#CrecimientoProfesional
#AprendeConLosPioneros

Online | DMC Perú

DMC Perú | CAPACITACIÓN PROFESIONAL

SESIÓN XIII

Docente: Victor Gutierrez
Data Architect

Agenda

TÉCNICAS DE OPTIMIZACIÓN

- Diferencia entre constraints vs Índices.
- Optimización uso (Set Statistics, Non-Sargable, Forceseek, fill factor, comprensión datos).
- Forzados de operación (Loop join, merge join, hash join). Uso de optimizadores.
- Mejores prácticas en desarrollo de consultas.

Diferencia entre constraints vs Índices.

Índices

- Un índice es un conjunto de referencias ordenadas a filas de una tabla.
- Puede contener datos de una o más columnas de una tabla.
- Mejora el rendimiento de la recuperación de datos al reducir el número de páginas físicas a las que debe acceder la base de datos para leer una fila en la base de datos.
- Debido a que los índices almacenan datos en orden, también eliminan la necesidad de crear almacenamiento temporal para la cláusula ORDER BY si la columna relevante se incluye en un índice

Diferencia entre constraints vs Índices.

KEY

En una base de datos, una clave consta de una o más columnas de una tabla a las que se les han otorgado propiedades específicas. Al definir una tabla o índice, debe especificar la clave (primaria o extranjera).

Clave primaria

- Una tabla solo puede tener una clave primaria, y la clave debe contener solo valores únicos sin ningún valor NULL. La tabla que contiene la clave primaria se denomina tabla primaria.
- Se utiliza como ***referencia maestra*** para columnas definidas como claves foráneas en otras tablas.
- Las claves externas solo pueden contener valores definidos en la clave primaria a la que se refieren.

Clave externa

- Una clave externa asocia valores contenidos en una o más columnas de una tabla a claves primarias de otras tablas. La tabla que contiene la clave externa se denomina tabla secundaria.
- La tabla secundaria hace referencia a una tabla primaria, que debe contener una clave primaria. Una columna definida como una clave externa no puede contener valores NULL. Los valores en una columna de clave externa deben coincidir con todos los valores o con un subconjunto de los valores en la Clave primaria referenciada. Una clave externa no puede contener valores que no estén en la clave primaria a la que se refiere.

Diferencia entre constraints vs Índices.

Restricciones (Constraint)

Las restricciones son reglas que la base de datos aplica para mejorar la integridad de los datos. Puede especificar todas las siguientes restricciones en el nivel de columna o en el nivel de tabla

Restricción única

- Obliga a una columna a contener solo valores únicos, no permite valores NULL
- Aunque una tabla puede contener cualquier número de columnas únicas, solo una puede ser la clave principal.

Verificar restricción (Check Constraint)

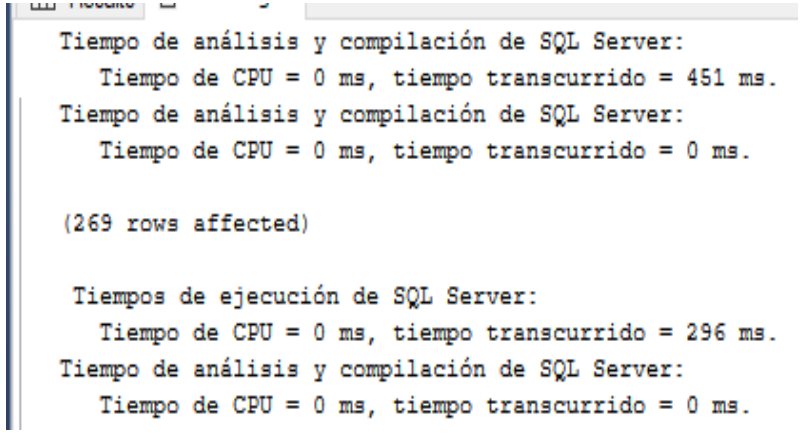
- Una restricción de verificación es una condición de búsqueda.
- Puede usar una restricción de verificación para asegurarse de que un valor que ingresa en una columna cumple con los criterios de la condición de búsqueda.
- Similar a las otras restricciones, puede definir una restricción de verificación al crear o alterar una tabla.

Optimización

SET STATISTICS TIME (Transact-SQL)

- Cuando SET STATISTICS TIME es ON, se muestran las estadísticas de tiempo de una instrucción. Cuando es OFF no se muestran las estadísticas de tiempo.
- La opción SET STATISTICS TIME se establece en tiempo de ejecución, no en tiempo de análisis.
- Microsoft SQL Server no puede calcular estadísticas precisas en el modo de fibra, que se activa al habilitar la opción de configuración agrupación ligera.
- La columna **cpu** de la tabla sysprocesses solo se actualiza cuando se ejecuta una consulta con SET STATISTICS TIME ON. Si SET STATISTICS TIME es OFF, se devuelve 0.
- Las opciones ON y OFF también afectan a la columna CPU en la vista Información del proceso para la actividad actual, en SQL Server Management Studio.

```
SET STATISTICS TIME ON;
GO
SELECT ProductID, StartDate, EndDate, StandardCost
FROM Production.ProductCostHistory
WHERE StandardCost < 500.00;
GO
SET STATISTICS TIME OFF;
GO
```



Optimización

SET STATISTICS IO (Transact-SQL)

- Muestra información relacionada con la cantidad de actividad de disco generada por las instrucciones Transact-SQL
- La opción SET STATISTICS IO se establece en tiempo de ejecución, no en tiempo de análisis.
- Cuando las instrucciones Transact-SQL recuperan columnas LOB, es posible que algunas operaciones de recuperación de LOB necesiten recorrer el árbol de LOB varias veces. Esto puede ocasionar que SET STATISTICS IO informe de un mayor número de lecturas lógicas del que cabría esperar. ***text ntext image varchar(max) nvarchar(max) varbinary(max)***

```
SET STATISTICS IO ON;
GO
SELECT *
FROM Production.ProductCostHistory
WHERE StandardCost < 500.00;
GO
SET STATISTICS IO OFF;
GO
```

```
(269 rows affected)
Tabla 'ProductCostHistory'.
Recuento de exámenes 1,
lecturas lógicas 5,
lecturas físicas 0,
lecturas anticipadas 0,
lecturas lógicas de LOB 0,
lecturas físicas de LOB 0,
lecturas anticipadas de LOB 0.
```

Optimización

SET STATISTICS IO (Transact-SQL)

Elemento de salida	Significado
Table	Nombre de la tabla.
Scan count	Número de búsquedas para recuperar todos los valores y generar el conjunto de datos final de la salida.
	El número es **0** si el índice usado es un índice único o un índice agrupado en una clave principal y está buscando un solo valor. **WHERE Primary_Key_Column = <value>.**
	El número es **1** cuando se busca un valor con un índice agrupado que no es único y que se define en una columna de clave de no principal. W**HERE Clustered_Index_Key_Column = <value>.**
	El número es **N** si N es el número de búsquedas empezó después de encontrar un valor de clave mediante la clave de índice.
logical reads	Número de páginas leídas de la caché de datos.
physical reads	Número de páginas leídas del disco.
read-ahead reads	Número de páginas llevadas a la caché por la consulta.
lob logical reads	Número de páginas leídas de la caché de datos.
lob physical reads	Número de páginas leídas del disco.
lob read-ahead reads	Número de páginas llevadas a la caché por la consulta.

Optimización

Sargable *Search ARGument ABLE*

Es una palabra que concatena las tres palabras: búsqueda, argumento y capacidad, buscamos:

- Consumiendo menos recursos del sistema
- Acelerar el rendimiento de la consulta
- Utilizando índices de manera más efectiva

Índices

Index Scan es cuando SQL Server lee todos los datos en las páginas de índice. Es muy costoso para el motor de SQL Server.

Index Seek es cuando SQL Server lee solo datos coincidentes en las páginas de índice. Este método es más eficiente para el rendimiento de las consultas porque reducirá el consumo de IO y de tiempo.

Optimización

Sargable *Search ARGument ABLE*

WHERE Uso de Funciones

```
SELECT FirstName
FROM Dummy_PersonTable
where LEFT(FirstName,1)='K'
```

El uso de:

- **SUBSTRING**
- **LEFT**
- **LTRIM**
- **RTRIM**
- **User defined functions**

Afectan al Índice

```
SELECT FirstName
FROM Dummy_PersonTable
where FirstName like 'K%'
```

Defined Values	[AdventureWorks2016].[dI
Description	Scan a nonclustered index
Estimated CPU Cost	0,0221262
Estimated Execution Mode	Row
Estimated I/O Cost	0,0468287
Estimated Number of Executions	1
Estimated Number of Rows Per Execu	1187
Estimated Number of Rows to be Rea	19972
Estimated Operator Cost	0,0689549 (100%)

Defined Values	[AdventureWorks2016].[
Description	Scan a particular range c
Estimated CPU Cost	0,0015519
Estimated Execution Mode	Row
Estimated I/O Cost	0,0053472
Estimated Number of Executions	1
Estimated Number of Rows Per Execu	1268,06
Estimated Number of Rows to be Rea	1268,06
Estimated Operator Cost	0,0068991 (100%)

Optimización

Sargable *Search ARGument ABLE*

WHERE Uso de Funciones

```
SELECT ModifiedDate
FROM Dummy_PersonTable
where YEAR(ModifiedDate)=2009
```

Defined Values	[AdventureWorks2016].[dbo].[
Description	Scan a nonclustered index, en
Estimated CPU Cost	0,0221262
Estimated Execution Mode	Row
Estimated I/O Cost	0,0357176
Estimated Number of Executions	1
Estimated Number of Rows Per E	123,511
Estimated Number of Rows to be	19972
Estimated Operator Cost	0,0578438 (97%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	15 B
Estimated Subtree Cost	0,0578438
Forced Index	False
ForceScan	False
ForceSeek	False
Logical Operation	Index Scan

```
SELECT ModifiedDate
FROM Dummy_PersonTable
where ModifiedDate BETWEEN
'20090101' AND '20091231'
```

Defined Values	[AdventureWorks2016].[dbo].[D
Description	Scan a particular range of rows
Estimated CPU Cost	0,0003737
Estimated Execution Mode	Row
Estimated I/O Cost	0,003125
Estimated Number of Executions	1
Estimated Number of Rows Per E	197,027
Estimated Number of Rows to be	197,027
Estimated Operator Cost	0,0034987 (100%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	15 B
Estimated Subtree Cost	0,0034987
Forced Index	False
ForceScan	False
ForceSeek	False
Logical Operation	Index Seek

Optimización

Sargable *Search ARGument ABLE*

WHERE Uso de Funciones

```
SELECT MiddleName
FROM Dummy_PersonTable
where ISNULL(MiddleName,'E') ='E'
```

```
SELECT MiddleName
FROM Dummy_PersonTable
where  (MiddleName IS NULL OR MiddleName='E')
```

Defined Values	[AdventureWorks2016].[d
Description	Scan a nonclustered inde
Estimated CPU Cost	0,0221262
Estimated Execution Mode	Row
Estimated I/O Cost	0,0334954
Estimated Number of Executions	1
Estimated Number of Rows Per Execu	6156,67
Estimated Number of Rows to be Rea	19972
Estimated Operator Cost	0,0556216 (100%)

Defined Values	[AdventureWorks2016].[
Description	Scan a particular range
Estimated CPU Cost	0,0103155
Estimated Execution Mode	Row
Estimated I/O Cost	0,0171991
Estimated Number of Executions	1
Estimated Number of Rows for All Exe	9235
Estimated Number of Rows Per Execu	9235
Estimated Number of Rows to be Rea	9235
Estimated Operator Cost	0,0275146 (100%)

Optimización

Forceseek

Especifica que el optimizador de consultas use solamente una operación de búsqueda de índice como ruta de acceso a los datos de la tabla o la vista.

FORCESEEK [**(** *index_value* **(** *index_column_name* [**,** ... *n*] **))**]

index_value
Es el nombre o el valor de identificador del índice. No se puede especificar el identificador de índice 0 (montón).
Para devolver el nombre o el identificador del índice, consulte a la vista de catálogo sys.indexes.

index_column_name
Es el nombre de la columna de índice que se va a incluir en la operación de búsqueda
El optimizador también puede considerar columnas adicionales si es necesario.
Por ejemplo, si se especifica un índice no clúster, el optimizador puede elegir el uso de columnas de clave de índice clúster además de las columnas especificadas.

Optimización

Forceseek

Sintaxis	Ejemplo	Descripción
Sin un índice o sugerencia INDEX	`FROM dbo.MyTable WITH (FORCESEEK)`	El optimizador de consultas sólo considera las operaciones de búsqueda de índice para tener acceso a la tabla o la vista mediante cualquier índice pertinente.
Combinado con una sugerencia INDEX	`FROM dbo.MyTable WITH (FORCESEEK, INDEX (MyIndex))`	El optimizador de consultas solo considera las operaciones de búsqueda de índice para tener acceso a la tabla o la vista mediante el índice especificado.
Parametrizado especificando un índice y columnas de índice	`FROM dbo.MyTable WITH (FORCESEEK (MyIndex (col1, col2, col3)))`	El optimizador de consultas solo considera las operaciones de búsqueda de índice para tener acceso a la tabla o la vista mediante el índice especificado usando al menos las columnas de índice especificadas.

Al especificar FORCESEEK con parámetros, limita el número de planes que el optimizador puede considerar en comparación con cuando se especifica FORCESEEK sin parámetros.

Optimización

Forceseek

En el ejemplo siguiente se usa la sugerencia FORCESEEK sin especificar un índice para obligar a que el optimizador de consultas realice una operación de búsqueda de índice en la tabla ***Sales.SalesOrderDetail***

```
SELECT *
FROM Sales.SalesOrderHeader AS h
INNER JOIN Sales.SalesOrderDetail AS d WITH (FORCESEEK)
    ON h.SalesOrderID = d.SalesOrderID
WHERE h.TotalDue > 100
AND (d.OrderQty > 5 OR d.LineTotal < 1000.00);
GO
```

Optimización

Forceseek

En el ejemplo siguiente se usa FORCESEEK con un índice para obligar a que el optimizador de consultas realice una operación de búsqueda de índice en el índice y la columna de índice especificados.

```
SELECT h.SalesOrderID, h.TotalDue, d.OrderQty
FROM Sales.SalesOrderHeader AS h
INNER JOIN Sales.SalesOrderDetail AS d
WITH (FORCESEEK (PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID (SalesOrderID)))
ON h.SalesOrderID = d.SalesOrderID
WHERE h.TotalDue > 100
AND (d.OrderQty > 5 OR d.LineTotal < 1000.00);
GO
```

Optimización

Compression

- La base de datos crece en el tiempo y a no ser que sea histórica el crecimiento no se va a detener.
- Cuantos más datos se encuentran en la base de datos, más tiempo (trabajo) se necesita el SQL Server para tratarlos
- Reduce la cantidad de espacio físico en disco requerido para almacenar datos y la cantidad de E/S de disco se guarda al realizar la compresión de datos de SQL Server.

Tipos de compresiones de datos de SQL Server:

- Compresión de datos a nivel de fila
- Compresión de datos a nivel de página

Optimización

Compression

Resumen
La compresión de datos a nivel de fila no afecta a los siguientes tipos de datos:

- Varchar,nvarchar,image,text,ntext
- XML, FILESTREAM, varbinary y sql_variant
- Date, time (ya extremadamente compactos)

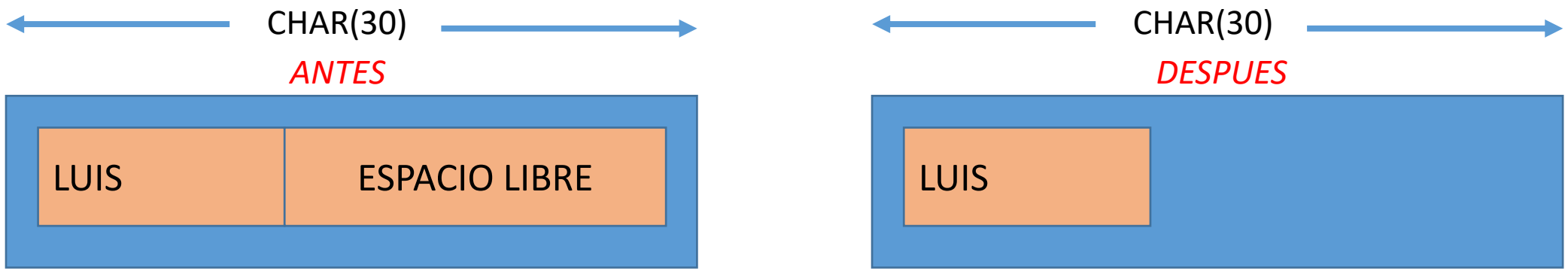
El beneficio se obtiene por tanto en otros tipos de datos:

- datetime, datetime2, datetimeoffset: ahorra 2 bytes si no almacena segundos
- char: solo ocupa lo necesario (como varchar)
- int, bigint,float, real,…: solo usa lo necesario
- binary: no almacena los ceros que puede evitar

Optimización

Compression

Row Compression: Ahorra espacio al cambiar los tipos de datos fijos en un formato de longitud variable



Optimización

Compression

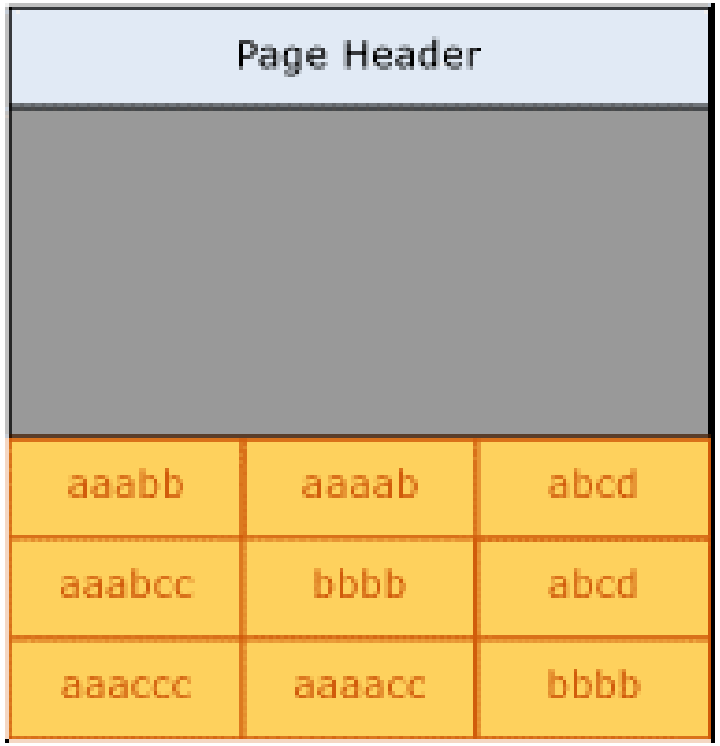
Page Compression: Utiliza compresión de filas y luego optimiza almacenamiento eliminando patrones repetidos de datos y reemplazándolos con referencia abreviada.

Optimización

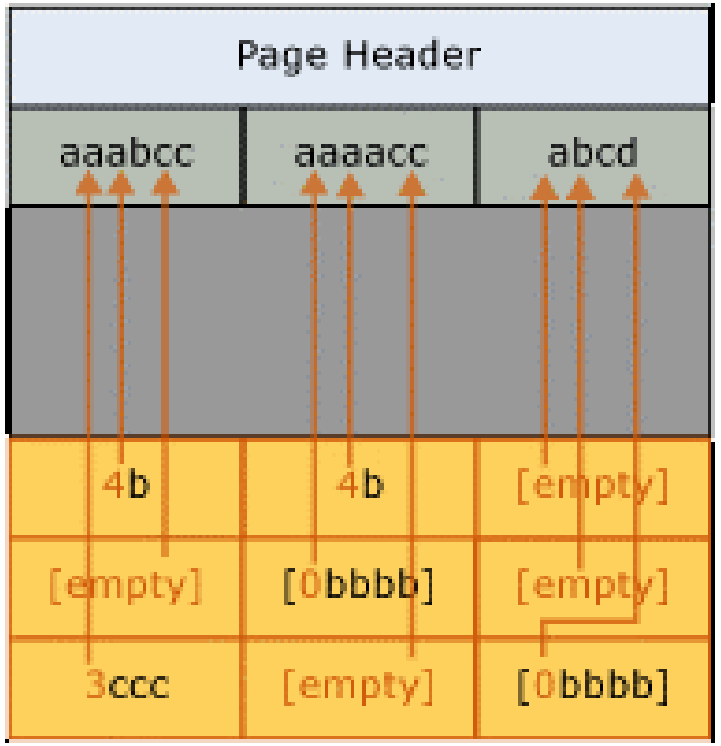
Compression

Page Compression:

La siguiente ilustración muestra una página de muestra de una tabla antes de la compresión de prefijo.



La siguiente ilustración muestra la misma página después de la compresión de prefijo. El prefijo se mueve al encabezado y los valores de la columna se cambian a referencias al prefijo.

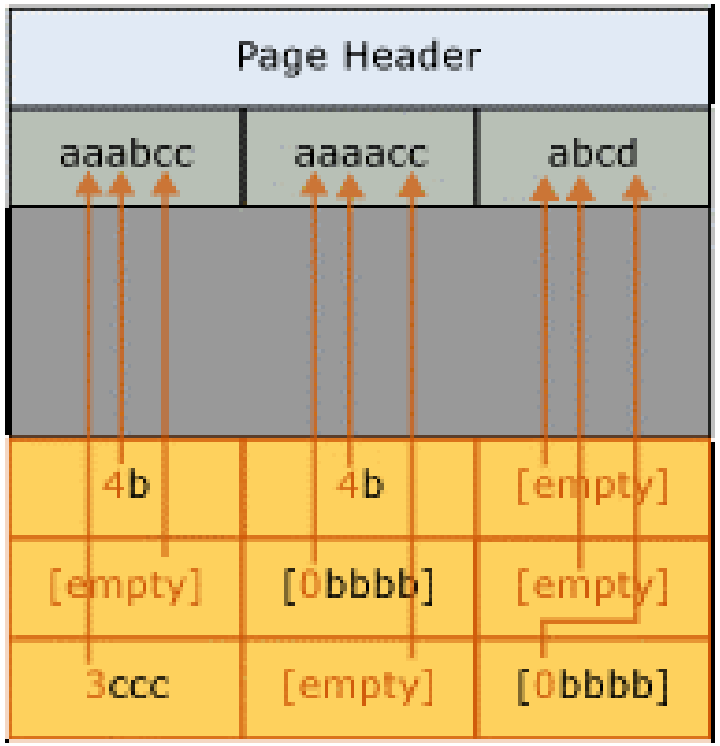


4b = los primeros cuatro caracteres del prefijo (aaab) están presentes para esa fila, y también el carácter b. Esto hace que el valor resultante sea aaabb, que es el valor original.

Optimización

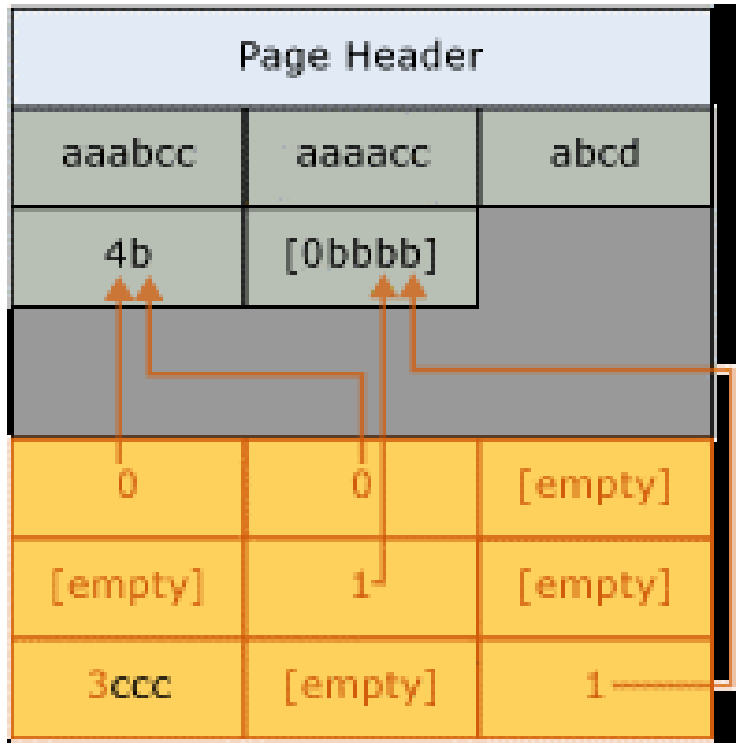
Compression

Page Compression:



Una vez completada la compresión del prefijo, se aplica la compresión del diccionario. La compresión de diccionario busca valores repetidos en cualquier lugar de la página. A diferencia de la compresión de prefijos, la compresión de diccionario no está restringida a una columna. La compresión de diccionario puede reemplazar los valores repetidos que ocurren en cualquier lugar de una página. La siguiente ilustración muestra la misma página después de la compresión del diccionario.

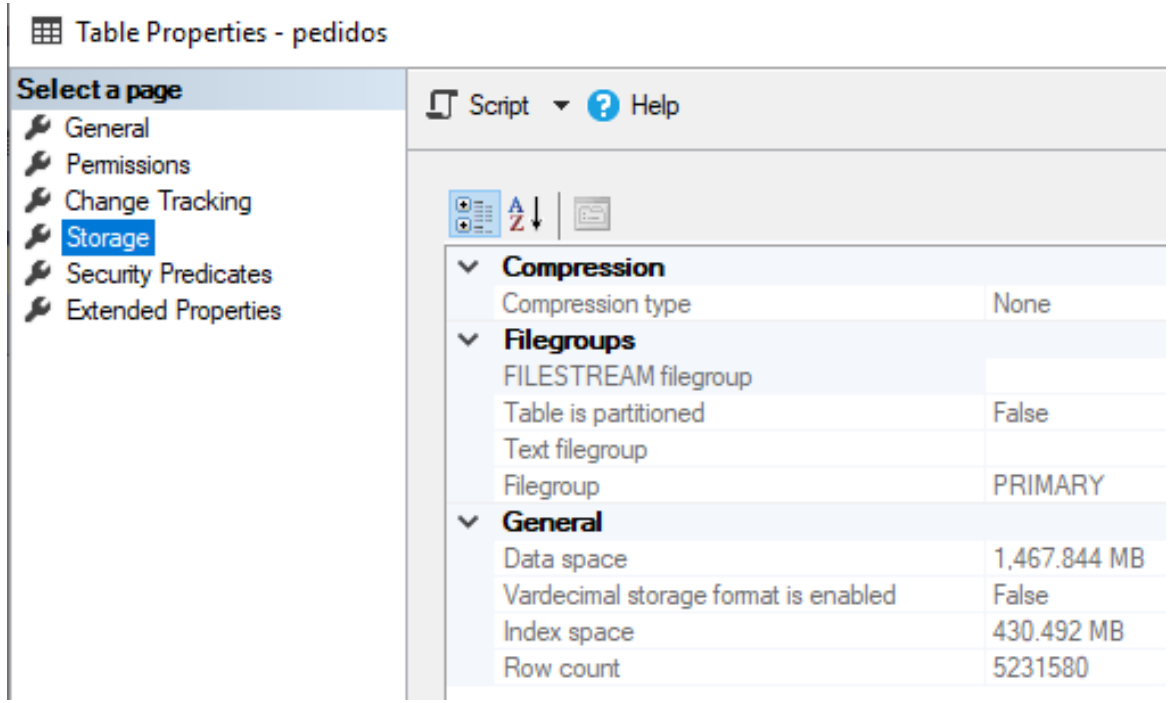
Tenga en cuenta que se ha hecho referencia al valor **4b** desde diferentes columnas de la página.



Optimización

Compression

Tabla a comprimir



Compression type	Row count	Current space	Requested compressed space
Row	5231580	1,467.852 MB	1,378.180 MB

Compression type	Row count	Current space	Requested compressed space
Page	5231580	1,467.852 MB	775.656 MB

Optimización

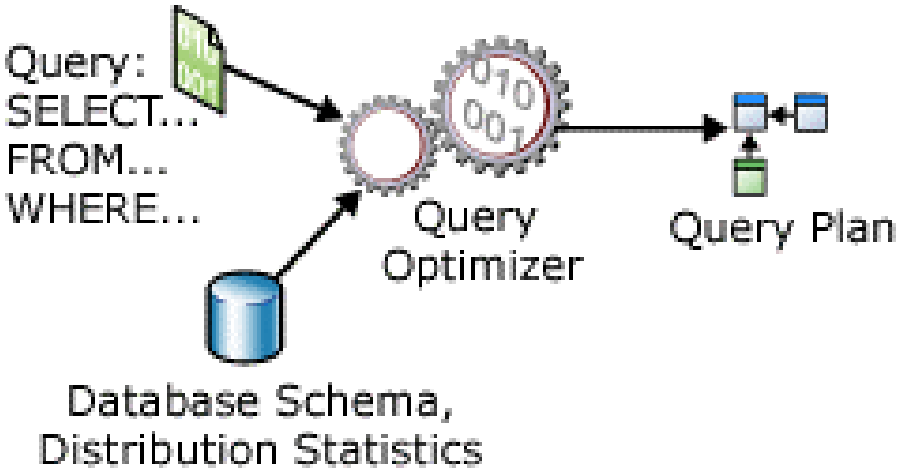
Forzados de Operaciones

SQL Server puede procesar las instrucciones Transact-SQL mediante dos modos de procesamiento distintos:

- Ejecución del modo de fila
- Ejecución del modo por lotes

Modo por lotes

- Las consultas procesan varias filas a la vez
- Cada columna dentro de un lote se almacena como un vector, el procesamiento por lotes se basa en vectores.
- Se usan algoritmos que se optimizan para las CPU de varios núcleos y el rendimiento de aumento de memoria que se encuentran en el hardware moderno.
- El modo por lotes funciona en los datos comprimidos siempre que sea posible
- El resultado es un mayor paralelismo y un rendimiento más rápido.



Optimización

Forzados de Operaciones

Nested Loops Join

Una unión de bucles anidados es una estructura lógica en la que un bucle (iteración) reside dentro de otro, es decir, para cada iteración del bucle externo, todas las iteraciones del bucle interno se ejecutan / procesan.

Una unión de Nested Loops funciona de la misma manera. Una de las tablas de unión se designa como la tabla externa y otra como la tabla interna. Para cada fila de la tabla externa, todas las filas de la tabla interna coinciden una por una si la fila coincide con ella se incluye en el conjunto de resultados; de lo contrario, se ignora. Luego se recoge la siguiente fila de la tabla externa y se repite el mismo proceso, y así sucesivamente.

El optimizador de SQL Server podría elegir una unión de bucles anidados cuando una de las tablas de unión es pequeña (considerada como la tabla externa) y otra es grande (considerada como la tabla interna que está indexada en la columna que está en la unión) y, por lo tanto, requiere un mínimo de E / S y la menor cantidad de comparaciones.

Optimización

Forzados de Operaciones

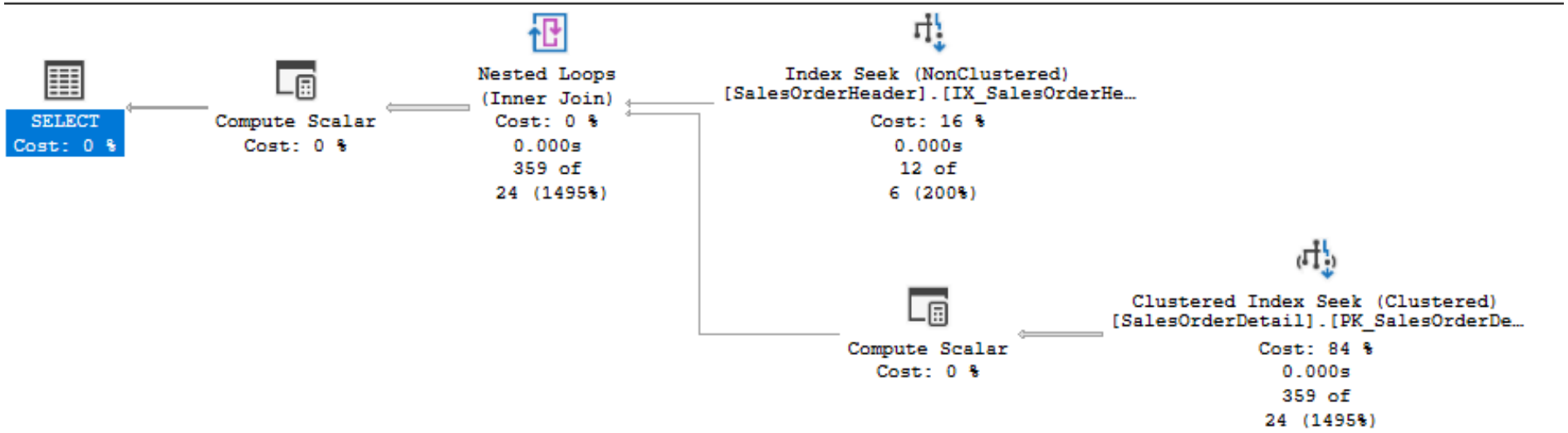
Nested Loops Join

- **native nested loops join**, en cuyo caso la búsqueda escanea toda la tabla o índice
- **index nested loops join** cuando la búsqueda puede utilizar un índice existente para realizar búsquedas
- **temporary index nested loops join** si el optimizador crea un índice temporal como parte del plan de consulta y lo destruye una vez que se completa la ejecución de la consulta.

Optimización

Nested Loops Join

```
SELECT H.CustomerID, H.SalesOrderID, D.ProductID, D.LineTotal
FROM Sales.SalesOrderHeader H
INNER JOIN Sales.SalesOrderDetail D ON H.SalesOrderID = D.SalesOrderID
WHERE H.CustomerID = 29825
```



Optimización

Nested Loops Join

```
SET STATISTICS PROFILE ON
SELECT H.CustomerID, H.SalesOrderID, D.ProductID, D.LineTotal
FROM Sales.SalesOrderHeader H
INNER JOIN Sales.SalesOrderDetail D ON H.SalesOrderID = D.SalesOrderID
WHERE H.CustomerID = 29825
SET STATISTICS PROFILE OFF
```

	Rows	Executes	StmtText	StmtId	NodeId	Par...	PhysicalOp	LogicalOp	Argument
1	312	1	SELECT H.CustomerID, H.SalesOrderID, D.ProductI...	1	1	0	NULL	NULL	NULL
2	0	0	|--Compute Scalar(DEFINE:([D].[LineTotal]=[Advent...	1	2	1	Compute Scalar	Compute Scalar	DEFINE:([D].[LineTotal]=[AdventureWorks
3	312	1	|--Nested Loops(Inner Join, OUTER REFEREN...	1	3	2	Nested Loops	Inner Join	OUTER REFERENCES:([H].[SalesOrderID
4	12	1	|--Index Seek(OBJECT:([AdventureWorks].[S...	1	4	3	Index Seek	Index Seek	OBJECT:([AdventureWorks].[Sales].[Sales
5	0	0	|--Compute Scalar(DEFINE:([D].[LineTotal]=i...	1	5	3	Compute Scalar	Compute Scalar	DEFINE:([D].[LineTotal]=isnull((CONVERT
6	312	12	|--Clustered Index Seek(OBJECT:([Adven...	1	6	5	Clustered Index Seek	Clustered Index Seek	OBJECT:([AdventureWorks].[Sales].[Sales

Optimización

Merge Join

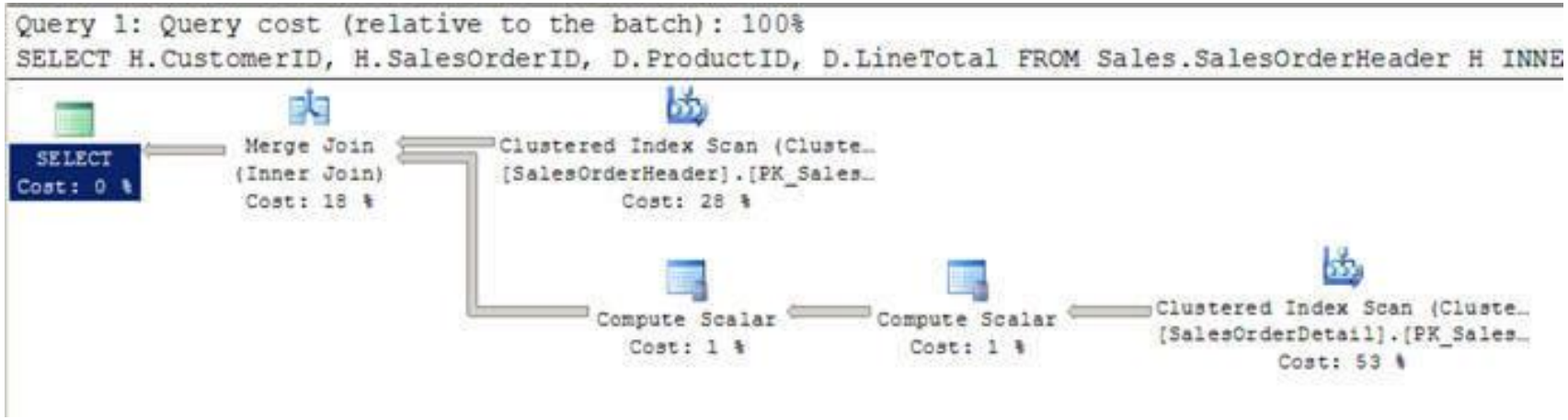
Requiere que ambas entradas se ordenen por el Columna de combinación / columnas de combinación (o ambas tablas de entrada tienen índices agrupados en la columna que une las tablas) y también requiere al menos una igualdad

- Es un operador de unión más eficiente y rápido ***si los datos ordenados*** se pueden obtener de un índice existente y realiza casi todas las operaciones de unión siempre que haya al menos un predicado de unión de igualdad involucrado.
- También admite múltiples predicados de unión de igualdad siempre que las tablas de entrada estén ordenadas en todas las claves de unión involucradas y estén en el mismo orden.

Optimización

Merge Join

```
SELECT H.CustomerID, H.SalesOrderID, D.ProductID, D.LineTotal
FROM Sales.SalesOrderHeader H
INNER JOIN Sales.SalesOrderDetail D ON H.SalesOrderID = D.SalesOrderID
WHERE H.CustomerID > 100
```



Optimización

Merge Join

```
SET STATISTICS PROFILE ON
SELECT H.CustomerID, H.SalesOrderID, D.ProductID, D.LineTotal
FROM Sales.SalesOrderHeader H
INNER JOIN Sales.SalesOrderDetail D ON H.SalesOrderID = D.SalesOrderID
WHERE H.CustomerID > 100
SET STATISTICS PROFILE OFF
```

	Rows	Executes	StmtText	StmtId	NodeId	Par...	PhysicalOp	LogicalOp
1	111350	1	SELECT H.CustomerID, H.SalesOrderID, D.ProductI...	1	1	0	NULL	NULL
2	111350	1	|--Merge Join(Inner Join, MERGE:([H].[SalesOrderI...	1	2	1	Merge Join	Inner Join
3	30883	1	|--Clustered Index Scan(OBJECT:([AdventureW...	1	3	2	Clustered Index Scan	Clustered Index Scan
4	0	0	|--Compute Scalar(DEFINE:([D].[LineTotal]=[Ad...	1	4	2	Compute Scalar	Compute Scalar
5	0	0	|--Compute Scalar(DEFINE:([D].[LineTotal]=i...	1	5	4	Compute Scalar	Compute Scalar
6	121317	1	|--Clustered Index Scan(OBJECT:([Adven...	1	6	5	Clustered Index Scan	Clustered Index Scan

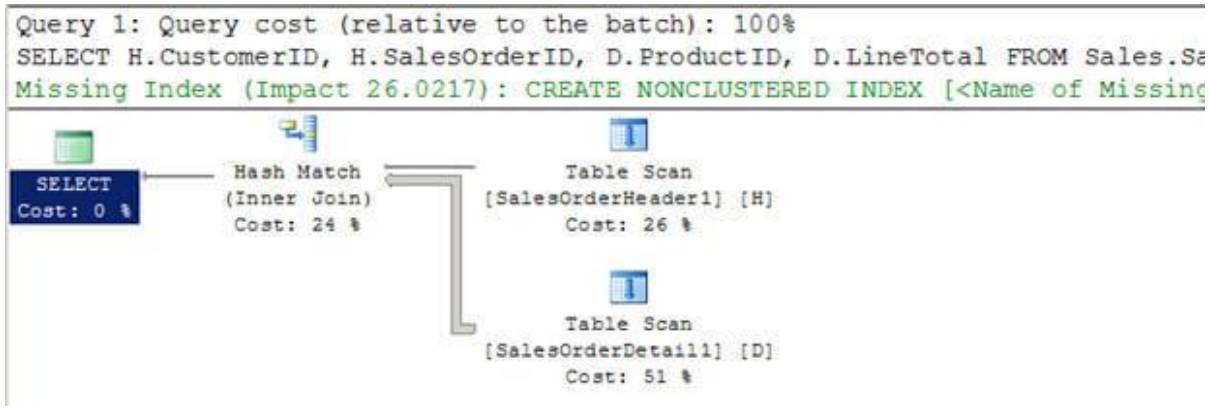
Optimización

Hash Join

Normalmente se usa cuando las tablas de entrada son bastante grandes y no existen índices adecuados en ellas. Una unión Hash se realiza en dos fases; la fase de construcción y la fase de sonda y, por lo tanto, la unión hash tiene dos entradas, es decir, entrada de construcción y entrada de sonda. La más pequeña de las entradas se considera como la entrada de compilación (para minimizar el requisito de memoria para almacenar la tabla hash discutida más adelante) y obviamente la otra es la entrada de la sonda.

```
SELECT * INTO Sales.SalesOrderHeader1 FROM Sales.SalesOrderHeader
SELECT * INTO Sales.SalesOrderDetail1 FROM Sales.SalesOrderDetail
GO

SELECT H.CustomerID, H.SalesOrderID, D.ProductID, D.LineTotal
FROM Sales.SalesOrderHeader1 H
INNER JOIN Sales.SalesOrderDetail1 D
ON H.SalesOrderID = D.SalesOrderID
WHERE H.CustomerID = 670
GO
```



Optimización

Hash Join

```
SET STATISTICS PROFILE ON
SELECT H.CustomerID, H.SalesOrderID, D.ProductID, D.LineTotal
FROM Sales.SalesOrderHeader1 H
INNER JOIN Sales.SalesOrderDetail1 D ON H.SalesOrderID = D.SalesOrderID
WHERE H.CustomerID = 29825
SET STATISTICS PROFILE OFF
```

	Rows	Executes	StmtText	StmtId	NodeId	Par...	PhysicalOp	LogicalOp	Argument
1	312	1	SELECT H.CustomerID, H.SalesOrderID, D.ProductI...	1	1	0	NULL	NULL	NULL
2	312	1	|--Hash Match(Inner Join, HASH:([H].[SalesOrderI...	1	2	1	Hash Match	Inner Join	HASH:([H].[S
3	12	1	|--Table Scan(OBJECT:([AdventureWorks].[Sal...	1	3	2	Table Scan	Table Scan	OBJECT:([Ac
4	121317	1	|--Table Scan(OBJECT:([AdventureWorks].[Sal...	1	4	2	Table Scan	Table Scan	OBJECT:([Ac

Optimización

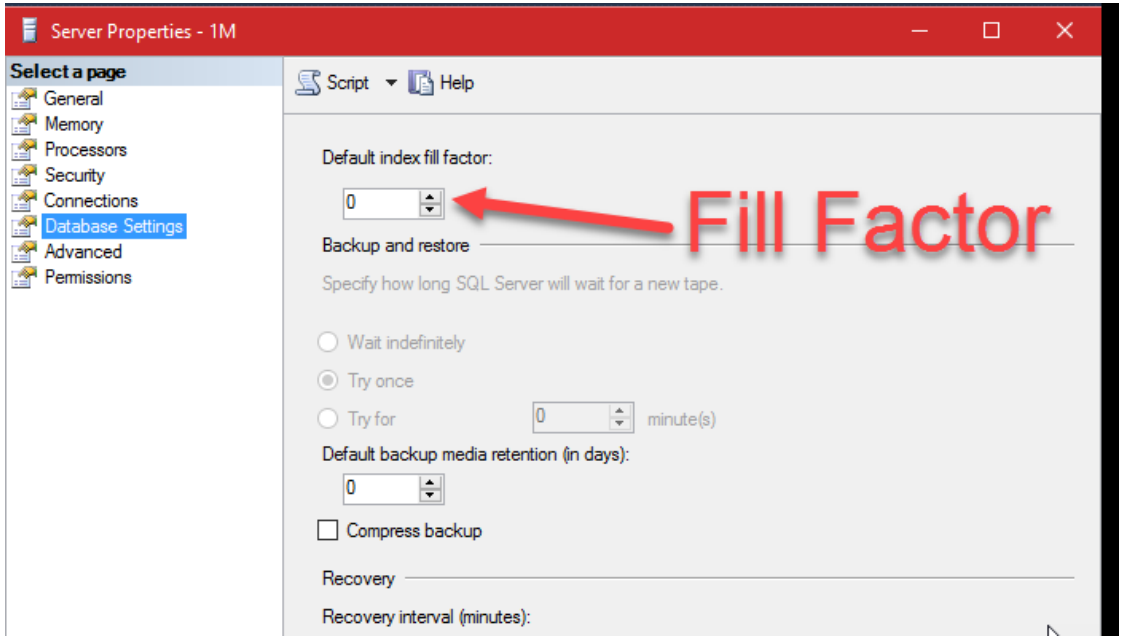
Resumen

- SQL Server hace un trabajo bastante bueno al decidir qué operador de combinación usar en cada condición.
- Comprender estas condiciones le ayuda a comprender qué se puede hacer en el ajuste del rendimiento.
- No se recomienda usar sugerencias de unión (usando la cláusula OPTION) para obligar a SQL Server a usar un operador de unión específico (a menos que no tenga otra salida), sino que puede usar otros medios como actualizar estadísticas, crear índices o volver a escribir su consulta.

Mantenimiento y Fragmentación

Fill Factor for an Index

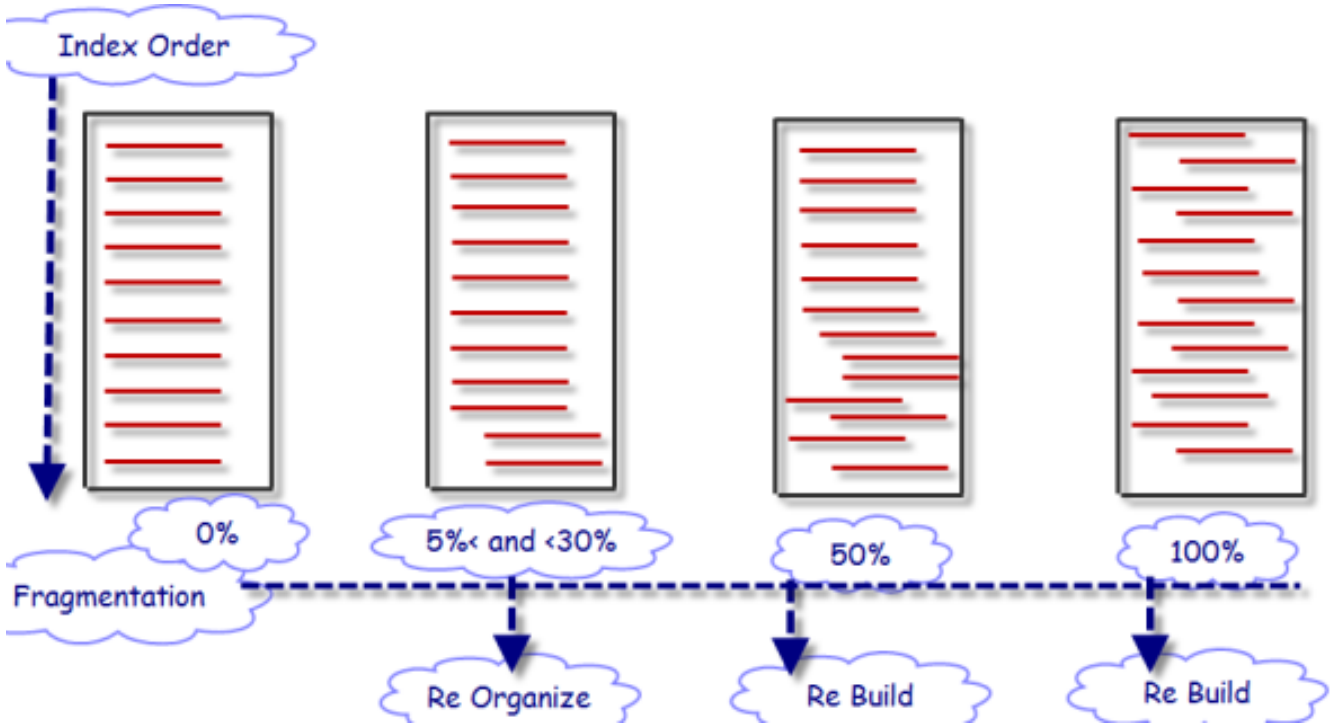
- La opción Fill Factor permite optimizar el almacenamiento y rendimiento de los datos de índice.
- Cuando se crea o se vuelve a generar un índice, el valor de factor de relleno determina el porcentaje de espacio en cada página que se tiene que rellenar con datos, por lo que se reserva el resto de cada página como espacio disponible para seguir creciendo.
- Si se especifica un valor de factor de relleno de 80, significa que el 20 por ciento de cada página de nivel de hoja se dejará vacío para proporcionar espacio para la expansión del índice a medida que se agreguen datos a la tabla
- El espacio vacío se reserva entre las filas de índice en lugar de al final del índice.
- El valor de factor de relleno es un porcentaje entre 1 y 100, y el valor predeterminado en el servidor es 0, lo que significa que las páginas de nivel de hoja se rellenan al máximo de su capacidad.



Mantenimiento y Fragmentación

Fill Factor for an Index

```
ALTER INDEX IX_Employee_OrganizationLevel_OrganizationNode
ON HumanResources.Employee
REBUILD WITH (FILLFACTOR = 80);
GO
```



DIIC Perú | CAPACITACIÓN **PROFESIONAL**

Online | DIIC Perú