Encontrar y eliminar índices duplicados o superpuestos

Introducción

La indexación eficaz es la clave para que sus consultas se ejecuten rápidamente mientras consume la menor cantidad de recursos posible en el proceso. Cada índice que se agrega a una tabla aumentará la velocidad de las lecturas que ahora pueden utilizar ese índice, pero a costa de la velocidad siempre que ese índice deba actualizarse. Además, sus procesos de mantenimiento de índices (reconstrucción / reorganización) ahora tendrán un índice adicional para operar.

SQL Server no tiene ninguna protección contra los índices que duplican el comportamiento y, por lo tanto, es posible que una tabla tenga cualquier número de índices duplicados o superpuestos sin que usted sepa que están allí. Esto constituiría un drenaje innecesario de recursos que podría evitarse fácilmente. ¿Cómo podemos ver fácilmente nuestros índices actuales y determinar si existen duplicados? ¿Qué pasa con los índices que contienen listas de columnas superpuestas que se pueden combinar?

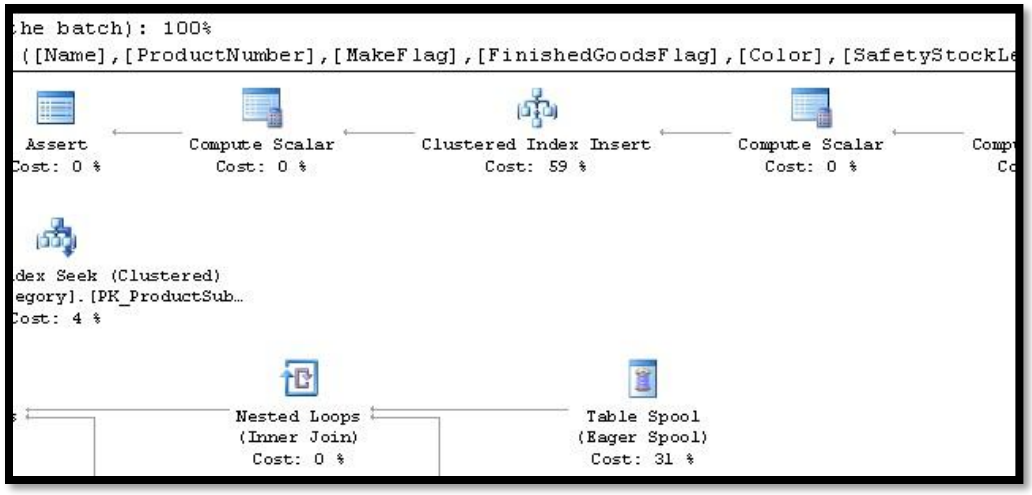
Problema

Para ilustrar el efecto de los índices duplicados, comenzaremos insertando una fila de datos en la tabla Production.Product en AdventureWorks:

```
SET STATISTICS IO ON
INSERT INTO Production.Product
        ( Name ,
          ProductNumber ,
          MakeFlag ,
          FinishedGoodsFlag ,
          Color ,
          SafetyStockLevel ,
          ReorderPoint ,
          StandardCost ,
          ListPrice ,
          Size ,
          SizeUnitMeasureCode ,
          WeightUnitMeasureCode ,
          Weight ,
          DaysToManufacture ,
          ProductLine ,
          Class ,
          Style ,
```

```
          ProductSubcategoryID ,
          ProductModelID ,
          SellStartDate ,
          SellEndDate ,
          DiscontinuedDate ,
          rowguid ,
          ModifiedDate
        )
VALUES  ( 'Test Product 1' , -- Name - Name
          N'12345' , -- ProductNumber - nvarchar(25)
          0 , -- MakeFlag - Flag
          0 , -- FinishedGoodsFlag - Flag
          N'Flurple' , -- Color - nvarchar(15)
          100 , -- SafetyStockLevel - smallint
          50 , -- ReorderPoint - smallint
          0 , -- StandardCost - money
          0 , -- ListPrice - money
          NULL , -- Size - nvarchar(5)
          NULL , -- SizeUnitMeasureCode - nchar(3)
          NULL , -- WeightUnitMeasureCode - nchar(3)
          7.77 , -- Weight - decimal
          12 , -- DaysToManufacture - int
          N'R' , -- ProductLine - nchar(2)
          N'L' , -- Class - nchar(2)
          N'U' , -- Style - nchar(2)
          2 , -- ProductSubcategoryID - int
          NULL , -- ProductModelID - int
          GETDATE() , -- SellStartDate - datetime
          GETDATE() , -- SellEndDate - datetime
          GETDATE() , -- DiscontinuedDate - datetime
          NEWID() , -- rowguid - uniqueidentifier
          GETDATE()  -- ModifiedDate - datetime
        )
```

El plan de ejecución es un poco complejo, ya que se están actualizando y comprobando muchas vistas, índices y restricciones, pero si nos acercamos solo a la inserción de índice agrupada, podemos ver las entrañas de la inserción:



Al pasar el cursor sobre el insert de índice agrupado, también podemos ver los detalles de qué objetos se actualizaron:

Clustered Index Insert

Insert rows in a clustered index.

Physical Operation	Clustered Index Insert
Logical Operation	Insert
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	1
Actual Number of Batches	0
Estimated Operator Cost	0.050005 (59%)
Estimated I/O Cost	0.05
Estimated CPU Cost	0.000005
Estimated Subtree Cost	0.0500065
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	1
Estimated Row Size	183 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	7

Object

[AdventureWorks].[Production].[Product].
[PK_Product_ProductID], [AdventureWorks].[Production].
[Product].[AK_Product_ProductNumber], [AdventureWorks].
[Production].[Product].[AK_Product_Name],
[AdventureWorks].[Production].[Product].
[AK_Product_rowguid], [AdventureWorks].[Production].
[Product].[NCI_Product_Weight]

Según los detalles en la parte inferior, podemos ver que se actualizan 5 índices:

- AK_Product_Name

- AK_Product_ProductNumber
- AK_Product_rowguid
- NCI_Product_Weight
- PK_Product_ProductID

Por último, aquí están las estadísticas de IO para la tabla Producto:

```
TABLE 'Product'. Scan COUNT 0, logical reads 11, physical reads 0, READ-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob READ-ahead reads 0.
```

El costo total del subárbol para esta operación es 0.085147 y hubo un total de 11 lecturas en la tabla. Tenga en cuenta que el inserto en sí comprendió el 59% del costo total del subárbol para toda la operación.

Ahora agregaremos un nuevo índice en Weight. Ya existe un índice en esta columna, por lo que lo estamos duplicando a propósito:

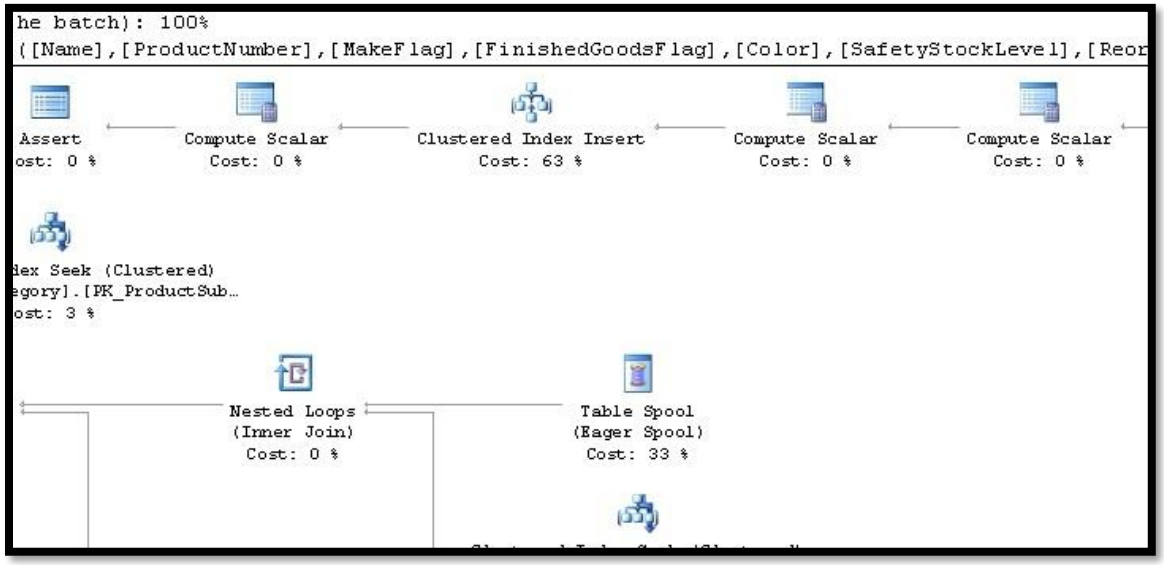
```
CREATE NONCLUSTERED INDEX NCI_Product_Weight_DUPE
 ON Production.Product (Weight ASC)
```

Con nuestro nuevo índice en su lugar, repetiremos la inserción anterior, ajustando algunas columnas para satisfacer índices únicos:

```
INSERT INTO Production.Product
        ( Name ,
          ProductNumber ,
          MakeFlag ,
          FinishedGoodsFlag ,
          Color ,
          SafetyStockLevel ,
          ReorderPoint ,
          StandardCost ,
          ListPrice ,
          Size ,
          SizeUnitMeasureCode ,
          WeightUnitMeasureCode ,
          Weight ,
          DaysToManufacture ,
          ProductLine ,
          Class ,
          Style ,
          ProductSubcategoryID ,
          ProductModelID ,
          SellStartDate ,
          SellEndDate ,
          DiscontinuedDate ,
          rowguid ,
          ModifiedDate
        )
VALUES  ( 'Test Product 2' , -- Name - Name
```

```
        N'12346' , -- ProductNumber - nvarchar(25)
        0 , -- MakeFlag - Flag
        0 , -- FinishedGoodsFlag - Flag
        N'Flurple' , -- Color - nvarchar(15)
        100 , -- SafetyStockLevel - smallint
        50 , -- ReorderPoint - smallint
        0 , -- StandardCost - money
        0 , -- ListPrice - money
        NULL , -- Size - nvarchar(5)
        NULL , -- SizeUnitMeasureCode - nchar(3)
        NULL , -- WeightUnitMeasureCode - nchar(3)
        7.77 , -- Weight - decimal
        12 , -- DaysToManufacture - int
        N'R' , -- ProductLine - nchar(2)
        N'L' , -- Class - nchar(2)
        N'U' , -- Style - nchar(2)
        2 , -- ProductSubcategoryID - int
        NULL , -- ProductModelID - int
        GETDATE() , -- SellStartDate - datetime
        GETDATE() , -- SellEndDate - datetime
        GETDATE() , -- DiscontinuedDate - datetime
        NEWID() , -- rowguid - uniqueidentifier
        GETDATE()  -- ModifiedDate - datetime
    )
```

Una mirada al plan de ejecución revela el mismo plan, pero con números ligeramente diferentes:



Pasando el cursor sobre la inserción, podemos verificar que nuestro nuevo índice se está actualizando:

Clustered Index Insert

Insert rows in a clustered index.

Physical Operation	Clustered Index Insert
Logical Operation	Insert
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	1
Actual Number of Batches	0
Estimated Operator Cost	0.060006 (63%)
Estimated I/O Cost	0.06
Estimated CPU Cost	0.000006
Estimated Subtree Cost	0.0600075
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	1
Estimated Row Size	183 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	7

Object

[AdventureWorks].[Production].[Product].
[PK_Product_ProductID], [AdventureWorks].[Production].
[Product].[AK_Product_ProductNumber], [AdventureWorks].
[Production].[Product].[AK_Product_Name],
[AdventureWorks].[Production].[Product].
[AK_Product_rowguid], [AdventureWorks].[Production].
[Product].[NCI_Product_Weight], [AdventureWorks].
[Production].[Product].[NCI_Product_Weight_DUPE]

Al final de la lista está nuestro duplicado recién creado. Una mirada a las estadísticas de IO sobre Production.Product. El producto muestra 2 lecturas adicionales que se necesitaban como parte del INSERT:

```
TABLE 'Product'. Scan COUNT 0, logical reads 13, physical reads 0, READ-
ahead reads 0, lob logical reads 0, lob physical reads 0, lob READ-ahead reads 0.
```

Esta vez, el inserto ocupa un total del 63% del costo total del subárbol, que ahora es 0.095148, un aumento del 11.7% sobre nuestro tiempo original. Nuestras lecturas también aumentaron un 18,2%. Esta tabla solo contiene 504 filas de forma predeterminada y, por lo tanto, es fácil inferir cómo estos costos incrementados se ampliarían a una tabla con miles o millones de filas y se convertirían en un problema costoso.

Solución

Hay un sin fin de vistas del sistema que usaremos para localizar índices duplicados:

- ***sys.schemas*** : contiene una fila para cada esquema en una base de datos.

- ***sys.tables*** : contiene una fila para cada tabla de usuario en una base de datos. Se puede unir a sys.schemas usando schema_id.
- ***sys.columns*** : contiene una fila para cada columna en una base de datos. Puede unirse a sys.tables usando object_id.
- ***sys.indexes*** : contiene una fila para cada índice en una base de datos. Puede unirse a sys.tables usando object_id.
- ***sys.index_columns*** : contiene una fila para cada columna de un índice. Puede unirse a sys.tables usando object_id y sys.indexes usando index_id.

Usando estas vistas, crearemos una consulta que nos mostrará una fila por índice, incluida una lista delimitada por comas de columnas clave y columnas de inclusión. Para mantener esto simple, filtraremos las tablas del sistema y cualquier cosa, excepto los índices agrupados y no agrupados. Las dos subconsultas correlacionadas toman todas las columnas de índice y las rellenan en las listas delimitadas por comas:

```
SELECT
       SCHEMA_DATA.name AS schema_name,
       TABLE_DATA.name AS table_name,
       INDEX_DATA.name AS index_name,
       STUFF((SELECT  ', ' + COLUMN_DATA_KEY_COLS.name + ' ' + CASE WHEN
INDEX_COLUMN_DATA_KEY_COLS.is_descending_key = 1 THEN 'DESC' ELSE 'ASC'
END -- Include column order (ASC / DESC)
                  FROM    sys.tables AS T
                          INNER JOIN sys.indexes INDEX_DATA_KEY_COLS
                                        ON T.object_id =
INDEX_DATA_KEY_COLS.object_id
                          INNER JOIN sys.index_columns
INDEX_COLUMN_DATA_KEY_COLS
                                        ON INDEX_DATA_KEY_COLS.object_id
= INDEX_COLUMN_DATA_KEY_COLS.object_id
                          AND INDEX_DATA_KEY_COLS.index_id =
INDEX_COLUMN_DATA_KEY_COLS.index_id
                          INNER JOIN sys.columns COLUMN_DATA_KEY_COLS
                                        ON T.object_id =
COLUMN_DATA_KEY_COLS.object_id
                          AND INDEX_COLUMN_DATA_KEY_COLS.column_id =
COLUMN_DATA_KEY_COLS.column_id
                  WHERE   INDEX_DATA.object_id =
INDEX_DATA_KEY_COLS.object_id
                          AND INDEX_DATA.index_id =
INDEX_DATA_KEY_COLS.index_id
                          AND INDEX_COLUMN_DATA_KEY_COLS.is_included_column
= 0
                  ORDER BY INDEX_COLUMN_DATA_KEY_COLS.key_ordinal
                  FOR XML PATH('')), 1, 2, '') AS key_column_list ,
   STUFF(( SELECT  ', ' + COLUMN_DATA_INC_COLS.name
                  FROM    sys.tables AS T
                          INNER JOIN sys.indexes INDEX_DATA_INC_COLS
```

```
                                                ON T.object_id =
INDEX_DATA_INC_COLS.object_id
                            INNER JOIN sys.index_columns
INDEX_COLUMN_DATA_INC_COLS
                                                ON INDEX_DATA_INC_COLS.object_id
= INDEX_COLUMN_DATA_INC_COLS.object_id
                            AND INDEX_DATA_INC_COLS.index_id =
INDEX_COLUMN_DATA_INC_COLS.index_id
                            INNER JOIN sys.columns COLUMN_DATA_INC_COLS
                                                ON T.object_id =
COLUMN_DATA_INC_COLS.object_id
                            AND INDEX_COLUMN_DATA_INC_COLS.column_id =
COLUMN_DATA_INC_COLS.column_id
                    WHERE   INDEX_DATA.object_id =
INDEX_DATA_INC_COLS.object_id
                            AND INDEX_DATA.index_id =
INDEX_DATA_INC_COLS.index_id
                            AND INDEX_COLUMN_DATA_INC_COLS.is_included_column
= 1
                    ORDER BY INDEX_COLUMN_DATA_INC_COLS.key_ordinal
                    FOR XML PATH('')), 1, 2, '') AS include_column_list,
        INDEX_DATA.is_disabled -- Check if index is disabled before
determining which dupe to drop (if applicable)
FROM sys.indexes INDEX_DATA
INNER JOIN sys.tables TABLE_DATA
ON TABLE_DATA.object_id = INDEX_DATA.object_id
INNER JOIN sys.schemas SCHEMA_DATA
ON SCHEMA_DATA.schema_id = TABLE_DATA.schema_id
WHERE TABLE_DATA.is_ms_shipped = 0
AND INDEX_DATA.type_desc IN ('NONCLUSTERED', 'CLUSTERED')
```

El resultado de nuestra consulta se ve así:

	schema_name	table_name	index_name	key_column_list	include_column_list
35	Production	TransactionHistory	PK_TransactionHistory_TransactionID	TransactionID	NULL
36	Production	TransactionHistory	IX_TransactionHistory_ProductID	ProductID	NULL
37	Production	TransactionHistory	IX_TransactionHistory_ReferenceOrderID_Reference...	ReferenceOrderID, ReferenceOrderLineID	NULL
38	Person	BusinessEntity	PK_BusinessEntity_BusinessEntityID	BusinessEntityID	NULL
39	Person	BusinessEntity	AK_BusinessEntity_rowguid	rowguid	NULL
40	Production	ProductReview	PK_ProductReview_ProductReviewID	ProductReviewID	NULL
41	Production	ProductReview	IX_ProductReview_ProductID_Name	ProductID, ReviewerName	Comments
42	Person	BusinessEntityAddress	PK_BusinessEntityAddress_BusinessEntityID_Address...	BusinessEntityID, AddressID, AddressTypeID	NULL
43	Person	BusinessEntityAddress	AK_BusinessEntityAddress_rowguid	rowguid	NULL
44	Person	BusinessEntityAddress	IX_BusinessEntityAddress_AddressID	AddressID	NULL
45	Person	BusinessEntityAddress	IX_BusinessEntityAddress_AddressTypeID	AddressTypeID	NULL
46	Production	TransactionHistoryAr...	PK_TransactionHistoryArchive_TransactionID	TransactionID	NULL
47	Production	TransactionHistoryAr...	IX_TransactionHistoryArchive_ProductID	ProductID	NULL

Nuestra primera tarea será utilizar estos datos para localizar duplicados exactos. Esto se puede hacer construyendo sobre nuestra consulta existente para encontrar filas duplicadas y devolverlas solo para nuestra revisión:

```
;WITH CTE_INDEX_DATA AS (
```

```
	SELECT
		SCHEMA_DATA.name AS schema_name,
		TABLE_DATA.name AS table_name,
		INDEX_DATA.name AS index_name,
		STUFF((SELECT  ', ' + COLUMN_DATA_KEY_COLS.name + ' ' + CASE
WHEN INDEX_COLUMN_DATA_KEY_COLS.is_descending_key = 1 THEN 'DESC' ELSE
'ASC' END -- Include column order (ASC / DESC)
				FROM	sys.tables AS T
					INNER JOIN sys.indexes
INDEX_DATA_KEY_COLS
					ON T.object_id =
INDEX_DATA_KEY_COLS.object_id
					INNER JOIN
sys.index_columns INDEX_COLUMN_DATA_KEY_COLS
					ON
INDEX_DATA_KEY_COLS.object_id = INDEX_COLUMN_DATA_KEY_COLS.object_id
					AND
INDEX_DATA_KEY_COLS.index_id = INDEX_COLUMN_DATA_KEY_COLS.index_id
					INNER JOIN sys.columns
COLUMN_DATA_KEY_COLS
					ON T.object_id =
COLUMN_DATA_KEY_COLS.object_id
					AND
INDEX_COLUMN_DATA_KEY_COLS.column_id = COLUMN_DATA_KEY_COLS.column_id
				WHERE	INDEX_DATA.object_id =
INDEX_DATA_KEY_COLS.object_id
					AND INDEX_DATA.index_id =
INDEX_DATA_KEY_COLS.index_id
					AND
INDEX_COLUMN_DATA_KEY_COLS.is_included_column = 0
				ORDER BY
INDEX_COLUMN_DATA_KEY_COLS.key_ordinal
				FOR XML PATH('')), 1, 2, '') AS
key_column_list ,
		STUFF(( SELECT  ', ' + COLUMN_DATA_INC_COLS.name
				FROM	sys.tables AS T
					INNER JOIN sys.indexes
INDEX_DATA_INC_COLS
					ON T.object_id =
INDEX_DATA_INC_COLS.object_id
					INNER JOIN
sys.index_columns INDEX_COLUMN_DATA_INC_COLS
					ON
INDEX_DATA_INC_COLS.object_id = INDEX_COLUMN_DATA_INC_COLS.object_id
					AND
INDEX_DATA_INC_COLS.index_id = INDEX_COLUMN_DATA_INC_COLS.index_id
					INNER JOIN sys.columns
COLUMN_DATA_INC_COLS
					ON T.object_id =
COLUMN_DATA_INC_COLS.object_id
					AND
INDEX_COLUMN_DATA_INC_COLS.column_id = COLUMN_DATA_INC_COLS.column_id
```

```
                                    WHERE    INDEX_DATA.object_id =
INDEX_DATA_INC_COLS.object_id
                                             AND INDEX_DATA.index_id =
INDEX_DATA_INC_COLS.index_id
                                             AND
INDEX_COLUMN_DATA_INC_COLS.is_included_column = 1
                                    ORDER BY
INDEX_COLUMN_DATA_INC_COLS.key_ordinal
                                    FOR XML PATH('')), 1, 2, '') AS
include_column_list,
       INDEX_DATA.is_disabled -- Check if index is disabled before
determining which dupe to drop (if applicable)
       FROM sys.indexes INDEX_DATA
       INNER JOIN sys.tables TABLE_DATA
       ON TABLE_DATA.object_id = INDEX_DATA.object_id
       INNER JOIN sys.schemas SCHEMA_DATA
       ON SCHEMA_DATA.schema_id = TABLE_DATA.schema_id
       WHERE TABLE_DATA.is_ms_shipped = 0
       AND INDEX_DATA.type_desc IN ('NONCLUSTERED', 'CLUSTERED')
)
SELECT
       *
FROM CTE_INDEX_DATA DUPE1
WHERE EXISTS
(SELECT * FROM CTE_INDEX_DATA DUPE2
 WHERE DUPE1.schema_name = DUPE2.schema_name
 AND DUPE1.table_name = DUPE2.table_name
 AND DUPE1.key_column_list = DUPE2.key_column_list
 AND ISNULL(DUPE1.include_column_list, '') =
ISNULL(DUPE2.include_column_list, '')
 AND DUPE1.index_name <> DUPE2.index_name)
```

El CTE es exactamente la misma consulta que creamos anteriormente. Encapsulamos esta lógica en un CTE por separado, ya que copiar y pegar las declaraciones STUFF grandes una y otra vez sería complicado y engorroso. Si bien podríamos usar la función de ventana ROW_NUMBER para devolver duplicados, queremos explícitamente ver todas las versiones del duplicado, no solo la que consideramos prescindible. Unir el CTE a sí mismo nos permite ubicar todos los conjuntos de filas que no son únicos y devolver todos los datos de cada uno. Los resultados se ven así:

	schema_name	table_name	index_name	key_column_list	include_column_list
1	Production	Document	UQ__Document__F73921F730F848ED	rowguid	NULL
2	Production	Document	AK_Document_rowguid	rowguid	NULL
3	Production	Product	NCI_Product_Weight	Weight	NULL
4	Production	Product	NCI_Product_Weight_DUPE	Weight	NULL

NCI_Product_Weight_DUPE, el índice que creamos anteriormente está aquí como se esperaba. También atrapamos a otro incauto en la tabla Production.Document. Ahora que hemos localizado 2 conjuntos de duplicados, podemos soltar una copia de cada uno

para eliminar la funcionalidad duplicada. No eliminaremos estos índices todavía, ya que serán necesarios para realizar más pruebas a continuación.

Nuestra solución hasta ahora solo identificará índices que son duplicados exactos entre sí. También estamos interesados en encontrar índices superpuestos: un escenario donde dos índices comparten la misma secuencia de índices en orden. Por ejemplo, creemos otro índice nuevo en Adventureworks:

```
CREATE NONCLUSTERED INDEX NCI_Product_Weight_OVERLAP
 ON Production.Product(Weight, ProductModelID ASC)
```

Este nuevo índice es similar al que duplicamos anteriormente, pero tiene ProductModelID como una columna de clave adicional. SQL Server usará ambos índices para diferentes consultas, pero podríamos eliminar los índices de Weight y confiar únicamente en este nuevo para manejar consultas de Weight o consultas de Weight & ProductModelID. Para las consultas solo en Weight, el nuevo índice sería un poco más caro de usar, ya que ProductModelID también debe devolverse, pero la diferencia es muy pequeña en comparación con la carga de mantener los otros índices indefinidamente.

Para detectar tanto índices duplicados como índices superpuestos, podemos modificar la consulta que escribimos anteriormente usando alguna manipulación de cadenas:

```
;WITH CTE_INDEX_DATA AS (
       SELECT
              SCHEMA_DATA.name AS schema_name,
              TABLE_DATA.name AS table_name,
              INDEX_DATA.name AS index_name,
              STUFF((SELECT  ', ' + COLUMN_DATA_KEY_COLS.name + ' ' + CASE
WHEN INDEX_COLUMN_DATA_KEY_COLS.is_descending_key = 1 THEN 'DESC' ELSE
'ASC' END -- Include column order (ASC / DESC)
                                    FROM    sys.tables AS T
                                           INNER JOIN sys.indexes
INDEX_DATA_KEY_COLS
                                           ON T.object_id =
INDEX_DATA_KEY_COLS.object_id
                                           INNER JOIN
sys.index_columns INDEX_COLUMN_DATA_KEY_COLS
                                           ON
INDEX_DATA_KEY_COLS.object_id = INDEX_COLUMN_DATA_KEY_COLS.object_id
                                           AND
INDEX_DATA_KEY_COLS.index_id = INDEX_COLUMN_DATA_KEY_COLS.index_id
                                           INNER JOIN sys.columns
COLUMN_DATA_KEY_COLS
                                           ON T.object_id =
COLUMN_DATA_KEY_COLS.object_id
                                           AND
INDEX_COLUMN_DATA_KEY_COLS.column_id = COLUMN_DATA_KEY_COLS.column_id
                                    WHERE    INDEX_DATA.object_id =
INDEX_DATA_KEY_COLS.object_id
```

```
                                                AND INDEX_DATA.index_id =
INDEX_DATA_KEY_COLS.index_id
                                                AND
INDEX_COLUMN_DATA_KEY_COLS.is_included_column = 0
                                    ORDER BY
INDEX_COLUMN_DATA_KEY_COLS.key_ordinal
                                    FOR XML PATH(''), 1, 2, '') AS
key_column_list ,
          STUFF(( SELECT  ', ' + COLUMN_DATA_INC_COLS.name
                                    FROM    sys.tables AS T
                                                INNER JOIN sys.indexes
INDEX_DATA_INC_COLS
                                                ON T.object_id =
INDEX_DATA_INC_COLS.object_id
                                                INNER JOIN
sys.index_columns INDEX_COLUMN_DATA_INC_COLS
                                                ON
INDEX_DATA_INC_COLS.object_id = INDEX_COLUMN_DATA_INC_COLS.object_id
                                                AND
INDEX_DATA_INC_COLS.index_id = INDEX_COLUMN_DATA_INC_COLS.index_id
                                                INNER JOIN sys.columns
COLUMN_DATA_INC_COLS
                                                ON T.object_id =
COLUMN_DATA_INC_COLS.object_id
                                                AND
INDEX_COLUMN_DATA_INC_COLS.column_id = COLUMN_DATA_INC_COLS.column_id
                                    WHERE   INDEX_DATA.object_id =
INDEX_DATA_INC_COLS.object_id
                                                AND INDEX_DATA.index_id =
INDEX_DATA_INC_COLS.index_id
                                                AND
INDEX_COLUMN_DATA_INC_COLS.is_included_column = 1
                                    ORDER BY
INDEX_COLUMN_DATA_INC_COLS.key_ordinal
                                    FOR XML PATH(''), 1, 2, '') AS
include_column_list,
       INDEX_DATA.is_disabled -- Check if index is disabled before
determining which dupe to drop (if applicable)
       FROM sys.indexes INDEX_DATA
       INNER JOIN sys.tables TABLE_DATA
       ON TABLE_DATA.object_id = INDEX_DATA.object_id
       INNER JOIN sys.schemas SCHEMA_DATA
       ON SCHEMA_DATA.schema_id = TABLE_DATA.schema_id
       WHERE TABLE_DATA.is_ms_shipped = 0
       AND INDEX_DATA.type_desc IN ('NONCLUSTERED', 'CLUSTERED')
)
SELECT
       *
FROM CTE_INDEX_DATA DUPE1
WHERE EXISTS
(SELECT * FROM CTE_INDEX_DATA DUPE2
 WHERE DUPE1.schema_name = DUPE2.schema_name
 AND DUPE1.table_name = DUPE2.table_name
```

```
 AND (DUPE1.key_column_list LIKE LEFT(DUPE2.key_column_list,
LEN(DUPE1.key_column_list)) OR DUPE2.key_column_list LIKE
LEFT(DUPE1.key_column_list, LEN(DUPE2.key_column_list)))
 AND DUPE1.index_name <> DUPE2.index_name)
```

Solo se necesitaron dos ajustes para cambiar nuestra consulta para buscar datos similares, en lugar de datos idénticos. Primero, eliminamos la verificación de igualdad en la lista de columnas incluidas. Por el bien de esta búsqueda, ignoraremos por completo las columnas incluidas. El conjunto de resultados generalmente será lo suficientemente pequeño como para que podamos escanearlos manualmente y ver qué columnas incluidas (si las hay) se superponen y se pueden combinar. El segundo cambio fue reescribir la igualdad de las columnas clave en una comparación de cadenas utilizando el operador LEFT. Aquí, estamos comparando cada lista de columnas clave en nuestro CTE con el segmento más a la izquierda del resto para determinar dónde tenemos partes repetidas. El resultado de nuestra consulta se ve así:

	schema_name	table_name	index_name	key_column_list	include_column_list
1	Production	Document	UQ__Document__F73921F730F848ED	rowguid	NULL
2	Production	Document	AK_Document_rowguid	rowguid	NULL
3	Production	Product	NCI_Product_Weight	Weight	NULL
4	Production	Product	NCI_Product_Weight_DUPE	Weight	NULL
5	Production	Product	NCI_Product_Weight_OVERLAP	Weight, ProductModelID	NULL

Además de las 4 filas que se devolvieron anteriormente, también obtenemos NCI_Product_Weight_OVERLAP, el nuevo índice que creamos anteriormente.

Por último, eliminaremos los índices que creamos en nuestro ejemplo:

```
DROP INDEX Production.Product.NCI_Product_Weight_DUPE
DROP INDEX Production.Product.NCI_Product_Weight_OVERLAP
```

Al ejecutar nuestra última consulta, podemos verificar una última vez si hay índices duplicados o superpuestos:

	schema_name	table_name	index_name	key_column_list	include_column_list
1	Production	Document	UQ__Document__F73921F730F848ED	rowguid	NULL
2	Production	Document	AK_Document_rowguid	rowguid	NULL

Todo lo que queda es un único índice duplicado, que dejaremos que Microsoft limpie a su gusto en una versión futura de AdventureWorks.

Nota del autor:

He modificado este artículo desde su publicación para agregar alguna funcionalidad y comentarios adicionales:

1. Se agregó is_disabled a la salida de la consulta. Si un índice está deshabilitado, querríamos saberlo antes de eliminar los duplicados. ¡Soltar el índice habilitado y dejar atrás el deshabilitado podría ser desastroso! ¡Gracias a grzegorz.mozejko por la idea!
2. Se agregó SQL adicional a las columnas de clave para que se genere la orden (ASC o DESC). Esto asegura que los índices que cubren diferentes órdenes de columna se traten como índices distintos y no se identifiquen como duplicados. Gracias a Carlo Romagnano por señalar esto.
3. Dejé de lado cualquier SQL generado para la eliminación automática, ya que agregar y eliminar índices nunca debe hacerse a la ligera y nunca sin suficiente tiempo e investigación.
4. Hay muchas otras formas de personalizar estos scripts para adaptarlos a su propio entorno o las necesidades de su empresa. Siéntase libre de meterse con ellos, reutilizarlos y compartirlos para que finalmente podamos encontrar formas más nuevas y mejores de manejar índices en SQL Server.

Conclusión

Los índices que duplican la funcionalidad entre sí desperdician valiosos recursos de SQL Server. Ser capaces de localizar de manera eficiente índices duplicados o superpuestos nos permitirá planificar un curso de acción que finalmente eliminará toda la lógica duplicada. Se debe tener cuidado al realizar cambios; Siempre pruebe primero en un entorno de desarrollo y confirme que sus cambios no eliminen inadvertidamente ninguna columna de índice no duplicada. Tener estas herramientas a su disposición le permitirá dedicar más tiempo a limpiar y verificar la eficiencia de su entorno de base de datos, y menos tiempo a tratar de identificar estos problemas en primer lugar.