

Indeksy, optymalizator - lab4

Imię i Nazwisko: Daniel Kuc

Celem ćwiczenia jest zapoznanie się z planami wykonania zapytań (execution plans), oraz z budową i możliwością wykorzystaniem indeksów (cz. 2).

Swoje odpowiedzi wpisuj w **czzerwone pola**. Ważne/wymagane są komentarze. Preferowane są zrzuty ekranu, ale dołącz też polecenia formie tekstowej

Oprogramowanie - co jest potrzebne?

Do wykonania ćwiczenia potrzebne jest następujące oprogramowanie

- MS SQL Server,
- SSMS - SQL Server Management Studio
- przykładowa baza danych AdventureWorks2017.

Oprogramowanie dostępne jest na przygotowanej maszynie wirtualnej

Przygotowanie

Stwórz swoją bazę danych o nazwie **XYZ**. Jeśli jednak dzielisz z kimś serwer, to użyj swoich inicjałów:

```
CREATE DATABASE XYZ  
GO
```

```
USE XYZ  
GO
```

Dokumentacja

Obowiązkowo:

- <https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes>
- <https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide>
- <https://www.simple-talk.com/sql/performance/14-sql-server-indexing-questions-you-were-too-shy-to-ask/>

Materiały rozszerzające:

- <https://www.sqlshack.com/sql-server-query-execution-plans-examples-select-statement/>

Zadanie 1 – Indeksy klastrowane I nieklastrowane

Celem zadania jest poznanie indeksów **klastrowanych** i **nieklastrowanych**.

Skopiuj tablicę Customer do swojej bazy danych:





```
SELECT * INTO [Customer] FROM [AdventureWorks2017].[Sales].[Customer]
```

Wykonaj analizy zapytań:

```
SELECT * FROM Customer WHERE StoreID = 594
SELECT * FROM Customer WHERE StoreID BETWEEN 594 AND 610
```

Zanotuj czas zapytania, jego koszt:

Koszt: 0.139/0.139, Czas: 4ms, 6ms

Estimated query progress:100%	Query 1: Query cost (relative to the batch): 50% SELECT * FROM [Customer] WHERE [StoreID]=@1
 SELECT	 Table Scan [Customer] 1 of 1 (100%)
Estimated query progress:100%	Query 2: Query cost (relative to the batch): 50% SELECT * FROM [Customer] WHERE [StoreID]>=@1 AND [StoreID]<=@2
 SELECT	 Table Scan [Customer] 16 of 16 (100%)

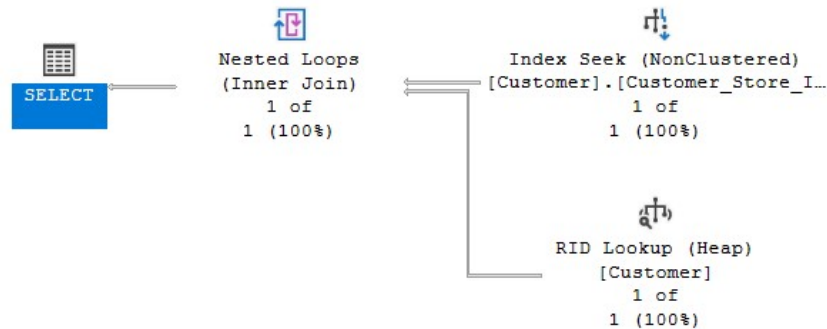
Dodaj indeks:

```
CREATE INDEX Customer_Store_Idx ON Customer(StoreID)
```

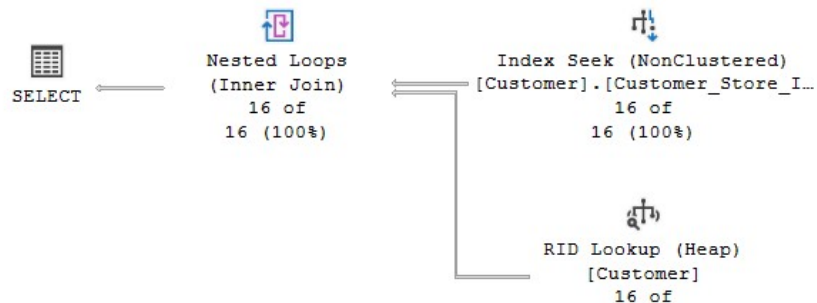
Jak zmienił się plan i czas? Czy jest możliwość optymalizacji?

Koszt:0.006/0.051, Czaasy: 1ms, 2ms

Estimated query progress:100% Query 1: Query cost (relative to the batch): 11%
SELECT * FROM [Customer] WHERE [StoreID]=@1



Estimated query progress:100% Query 2: Query cost (relative to the batch): 89%
SELECT * FROM [Customer] WHERE [StoreID]>=@1 AND [StoreID]<=@2



Po dodaniu indeksu, pierwsze zapytanie będzie miało wydajniejszy plan wykonania, ponieważ baza danych będzie szybciej lokalizować wszystkie wiersze, w których wartość StoreID wynosi 594. Będzie to efektywniejsze niż przeszukiwanie całej tabeli.

Drugie zapytanie również skorzysta z indeksu, ale będzie musiało przeszukać więcej wierszy, ponieważ StoreID, przyjmuje kilka różnych wartości w przeciwieństwie do zapytania pierwszego. Jednakże, ze względu na wykorzystanie indeksu, wykonanie tego zapytania również będzie znacznie szybsze.

jeśli tabela Customer jest duża i zawiera wiele kolumn, to w przypadku pierwszego zapytania indeks nie zawsze będzie wykorzystany przez serwer bazy danych, ponieważ może być bardziej wydajne wykonanie pełnego przeszukania tabeli niż korzystanie z indeksu. W takim przypadku można rozważyć dodanie dodatkowego indeksu, które są często używane w zapytaniach jako filtracja.

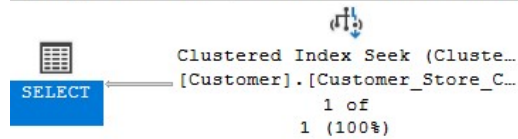
Dodaj indeks klastrowany:

```
CREATE CLUSTERED INDEX Customer_Store_Cls_Idx ON Customer (StoreID)
```

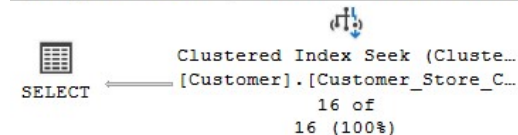
Czy zmienił się plan i czas? Skomentuj dwa podejścia w wyszukiwaniu krotek.

Koszt: 0.003/0.003, Czasy: 1ms, 1ms

Estimated query progress:100%	Query 1: Query cost (relative to the batch): 50% SELECT * FROM [Customer] WHERE [StoreID]=@1
-------------------------------	---



Estimated query progress:100%	Query 2: Query cost (relative to the batch): 50% SELECT * FROM [Customer] WHERE [StoreID]>=@1 AND [StoreID]<=@2
-------------------------------	--



Obydwa zapytania będą korzystać z nowego indeksu, a plan wykonania będzie zakładał, że dane zostaną pobrane z indeksu, a nie z tabeli. Dzięki temu operacje I/O będą "tanie", a zapytania zostaną wykonane dużo szybciej. (Dla nieco większych tabel, tutaj liczba rekordów nie jest pokazana)

Jednym podejściem do wyszukiwania krotek jest "naiwne" przeszukiwanie, gdzie serwer bazy danych przechodzi przez całą tabelę, aby znaleźć pasujące wiersze. To podejście jest bardzo kosztowne, zwłaszcza dla dużych tabel, ponieważ wymaga przeszukania każdego wiersza w tabeli. Innym podejściem jest wyszukiwanie wspomagane przez indeksy, gdzie baza danych korzysta z indeksu, aby znaleźć pasujące wiersze. Wyszukiwanie indeksowe jest dużo szybsze niż klasyczne, ponieważ baza danych musi przeszukać tylko małą część tabeli, która odpowiada kryteriom wyszukiwania definiowanym przez indeksy w bazie.

Zadanie 2 – Indeksy zawierające dodatkowe dane z kolumn

Celem zadania jest poznanie indeksów z przechowywaniem kolumn
Skopiuj tablicę Person do swojej bazy danych:

```
SELECT [BusinessEntityID]
      , [PersonType]
      , [NameStyle]
      , [Title]
      , [FirstName]
      , [MiddleName]
      , [LastName]
      , [Suffix]0
      , [EmailPromotion]
      , [rowguid]
      , [ModifiedDate]
INTO [Person]
FROM [AdventureWorks2017].[Person].[Person]
```

Wykonaj analizę planu dla trzech zapytań:

```

SELECT * FROM [Person] WHERE LastName = 'Agbonile'

SELECT * FROM [Person] WHERE LastName = 'Agbonile' AND FirstName =
'Osarumwense'

SELECT * FROM [Person] WHERE FirstName = 'Osarumwense'

```

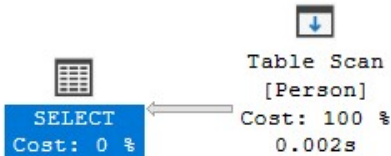
Co można o nich powiedzieć?

Mamy do czynienia z naiwnym, klasycznym, liniowym przeszukiwaniem/skanowaniem tabeli.

Koszty: 0.175, 0.175, 0.175

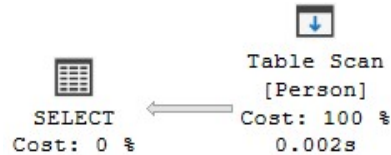
Query 1: Query cost (relative to the batch): 33%

```
SELECT * FROM [Person] WHERE [LastName]=@1
```



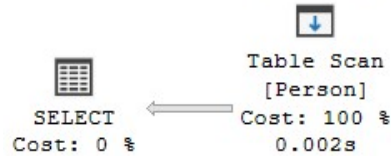
Query 2: Query cost (relative to the batch): 33%

```
SELECT * FROM [Person] WHERE [LastName]=@1 AND [FirstName]=@2
```



Query 3: Query cost (relative to the batch): 33%

```
SELECT * FROM [Person] WHERE [FirstName]=@1
```



Przygotuj indeks obejmujący te zapytanie:

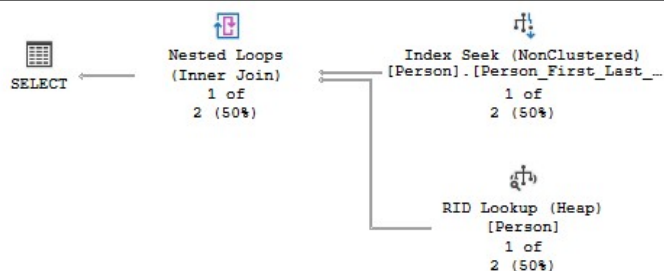
```

CREATE INDEX Person_First_Last_Name_Idx
ON Person(LastName, FirstName)

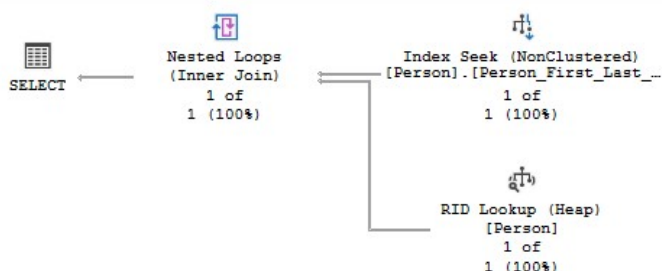
```

Sprawdź plan zapytania. Co się zmieniło?

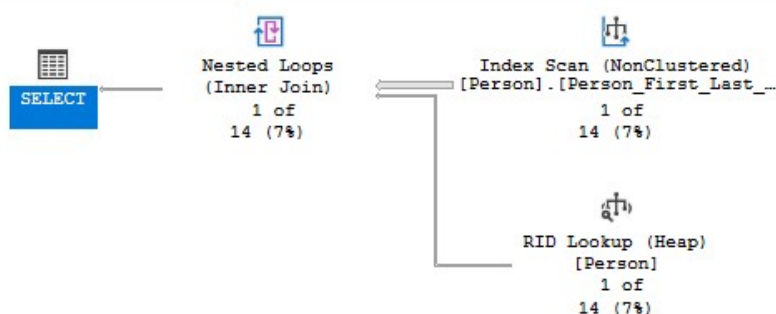
Estimated query progress:100%	Query 1: Query cost (relative to the batch): 5% SELECT * FROM [Person] WHERE [LastName]=@1
-------------------------------	---



Estimated query progress:100%	Query 2: Query cost (relative to the batch): 4% SELECT * FROM [Person] WHERE [LastName]=@1 AND [FirstName]=@2
-------------------------------	--



Estimated query progress:100%	Query 3: Query cost (relative to the batch): 91% SELECT * FROM [Person] WHERE [FirstName]=@1 Missing Index (Impact 97.4858): CREATE NONCLUSTE
-------------------------------	---



Koszty: 0.006, 0.006, 0.15








Dla pierwszego polecenia, zapytanie będzie korzystało z indeksu Person_First_Last_Name_Idx i wykonanie zapytania będzie szybsze niż przed dodaniem indeksu.

Dla drugiego polecenia, zapytanie również będzie korzystało z indeksu Person_First_Last_Name_Idx, a następnie będzie przeszukiwać indeks po kolumnie FirstName. Dzięki temu również czas wykonania zapytania powinien się skrócić.

Dla trzeciego polecenia, indeks Person_First_Last_Name_Idx nie będzie już aż tak pomocny, ponieważ kolumna FirstName nie jest pierwsza w indeksie. Zapytanie wciąż może skorzystać z indeksu, ale nie będzie to najbardziej optymalna droga. Sam Ssms podpowiada nam o stworzeniu kolejnego indeksu aby zwiększyć wydajność 3 zapytania.

Missing Index (Impact 97.4858): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Person] ([FirstName])

Przeprowadź ponownie analizę zapytań tym razem dla parametrów: FirstName = 'Angela' LastName = 'Price'. (Trzy zapytania, różna kombinacja parametrów). Czym różni się ten plan od zapytania o 'Osarumwense Agbonile' . Dlaczego tak jest?

Results	Messages	Live Query Statistics	Execution plan
Estimated query progress:100%	Query 1: Query cost (relative to the batch): 49% SELECT * FROM [Person] WHERE [LastName]=@1		
 Table Scan [Person] 84 of 84 (100%)			
 SELECT			
Estimated query progress:100%	Query 2: Query cost (relative to the batch): 2% SELECT * FROM [Person] WHERE [LastName]=@1 AND [FirstName]=@2		
 Nested Loops (Inner Join) 1 of 1 (100%)			
 Index Seek (NonClustered) [Person].[Person_First_Last_...] 1 of 1 (100%)			
 RID Lookup (Heap) [Person] 1 of			
Estimated query progress:100%	Query 3: Query cost (relative to the batch): 49% SELECT * FROM [Person] WHERE [FirstName]=@1 Missing Index (Impact 97.9211): CREATE NONCLUSTERED INDEX [<Na		
 Table Scan [Person] 50 of 50 (100%)			
 SELECT			

Koszt: 0.175, 0.006, 0.175

Pierwsze zapytanie będzie miało dość "duży" koszt, ponieważ będzie musiało naiwnie przejrzeć całą tabelę, aby znaleźć persone z nazwiskiem 'Price'. W takim przypadku indeks na kolumnie LastName może okazać się pomocny w przyspieszeniu zapytania.

Drugie zapytanie będzie miało niski koszt, ponieważ zapytanie będzie wyszukiwać po dwóch kolumnach, a indeks Person_First_Last_Name_Idx na obu kolumnach pozwoli na szybkie odnalezienie odpowiednich rekordów.

Trzecie zapytanie będzie miało koszt podobny do pierwszego, ponieważ będzie musiało przejrzeć całą tabelę, aby znaleźć persone z imieniem 'Angela'. W takim przypadku indeks na kolumnie FirstName może okazać się pomocny w przyspieszeniu zapytania.

Plan zapytań będzie różnił się od planu zapytania dotyczącego 'Osarumwense Agbonile', ponieważ różne wartości parametrów zapytań będą miały wpływ na koszt i strategię wykonania zapytania. Indeks na kolumnie LastName i FirstName pomoże w przyspieszeniu zapytania, które wyszukuje po obu tych kolumnach. Jednakże, indeks na jednej kolumnie nie będzie wystarczający do przyspieszenia zapytań, które wyszukują tylko po jednej kolumnie.

Zadanie 3

Skopiuj tablicę PurchaseOrderDetail do swojej bazy danych:

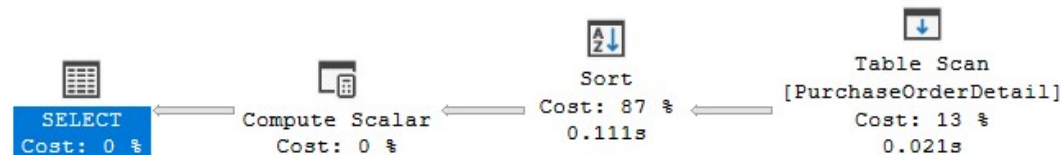
```
SELECT * INTO [PurchaseOrderDetail] FROM  
[AdventureWorks2017].[Purchasing].[PurchaseOrderDetail]
```

Wykonaj analizę zapytania:

```
SELECT RejectedQty, ((RejectedQty/OrderQty)*100) AS RejectionRate,  
ProductID, DueDate  
FROM PurchaseOrderDetail  
ORDER BY RejectedQty DESC, ProductID ASC
```

Która część zapytania ma największy koszt?

Query 1: Query cost (relative to the batch): 100%
SELECT RejectedQty, ((RejectedQty/OrderQty)*100) AS RejectionRate,



Największym kosztem wykonania danego zapytania jest sortowanie, która zajmuje 87% polecenia.

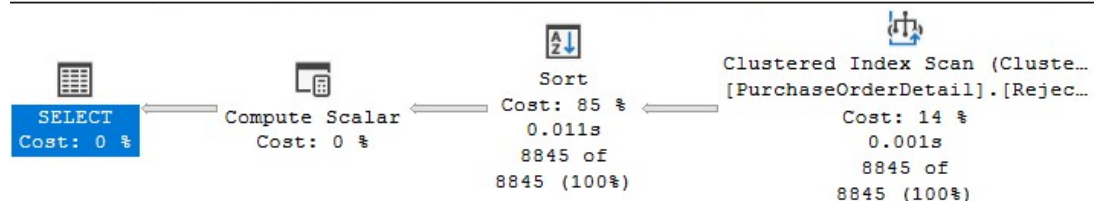
Jaki indeks można zastosować aby zoptymalizować koszt zapytania? Napisz go:

Pierwsza myśl (nie wiem czy koniecznie najlepsza):

```
CREATE CLUSTERED INDEX RejectedQt_idx_1 ON PurchaseOrderDetail (RejectedQty)
```

Wklej obrazek z planem:

Query 1: Query cost (relative to the batch): 100%
SELECT RejectedQty, ((RejectedQty/OrderQty)*100) AS RejectionRate, Prod



Zadanie 4

Celem zadania jest porównanie indeksów zawierających wszystkie kolumny z przechowywującym kolumny.

Skopiuj tablicę Address do swojej bazy danych:

```
SELECT * INTO [Address] FROM [AdventureWorks2017].[Person].[Address]
```

W tej części będziemy analizować następujące zapytanie:

```
SELECT AddressLine1, AddressLine2, City, StateProvinceID, PostalCode  
FROM Address  
WHERE PostalCode BETWEEN N'98000' and N'99999'
```


Stwórz dwa indeksy:

```
CREATE INDEX Address_PostalCode_1
ON Address (PostalCode)
INCLUDE (AddressLine1, AddressLine2, City, StateProvinceID);
GO

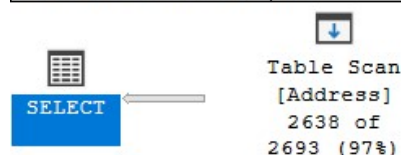
CREATE INDEX Address_PostalCode_2
ON Address (PostalCode, AddressLine1, AddressLine2, City,
StateProvinceID);
GO
```

Czy jest widoczna różnica w zapytaniach? Jeśli tak to jaka? Aby wymusić użycie indeksu użyj WITH(INDEX(Address_PostalCode_1)) po FROM:

Przed stworzeniem indeksów:

Koszt: 0.255

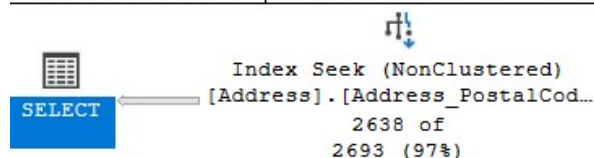
Estimated query progress:100%	Query 1: Query cost (re (@1 nvarchar
-------------------------------	--------------------------------------



Po stworzeniu indeksów:

Koszt: 0.025

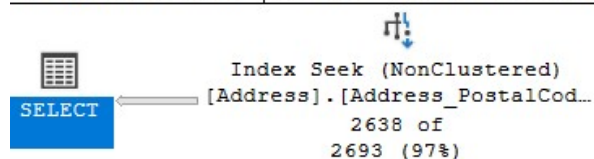
Estimated query progress:100%	Query 1: Query cost (re SELECT [AddressLine1], [
-------------------------------	--



Po wymuszeniu użycia indeksu pierwszego:

Koszt: 0.025

Estimated query progress:100%	Query 1: Query cost (re SELECT AddressLine1, A
-------------------------------	--



Znacząca różnica jest pomiędzy zapytaniem wykonanym przed dodaniem indeksów w bazie danych a po dodaniu indeksów(Znów kolejny raz dowiadujemy się, że "mądrę" indeksy działają).

Po wykonaniu polecenia dla indeksu Address_PostalCode_1 oraz Address_PostalCode_2 nie ma znaczącej różnicy.

Sprawdź rozmiar Indeksów:

```

SELECT i.[name] AS IndexName, SUM(s.[used_page_count]) * 8 AS IndexSizeKB
FROM sys.dm_db_partition_stats AS s
INNER JOIN sys.indexes AS i ON s.[object_id] = i.[object_id] AND
s.[index_id] = i.[index_id]
WHERE i.[name] = 'Address_PostalCode_1' OR i.[name] =
'Address_PostalCode_2'
GROUP BY i.[name]
GO

```

Który jest większy? Jak można skomentować te dwa podejścia? Które kolumny wpływają na to?

	IndexName	IndexSizeKB
1	Address_PostalCode_1	1784
2	Address_PostalCode_2	1808

Rozmiar drugiego indeksu (1808 KB) jest większy niż pierwszego indeksu (1784 KB).

Oba indeksy są tworzone na kolumnie PostalCode, ale różnią się tym, które kolumny są zawarte w indeksie. Pierwszy indeks zawiera tylko PostalCode i kolumny INCLUDE, podczas gdy drugi indeks zawiera wszystkie kolumny, które są w wyrażeniu SELECT.

Indeksy z większą liczbą kolumn mają większy rozmiar, ponieważ przechowują dodatkowe informacje. W tym przypadku, drugi indeks ma większy rozmiar ze względu na większą ilość kolumn w indeksie.

Podejście z wykorzystaniem indeksu z kolumnami INCLUDE jest stosowane w celu optymalizacji zapytań i redukcji rozmiaru indeksu. Indeksy z kolumnami INCLUDE zawierają dodatkowe kolumny, które nie są używane w indeksie, ale są używane w zapytaniach, co zwiększa wydajność zapytań. Natomiast podejście z drugim indeksem, który zawiera wszystkie kolumny z wyrażenia SELECT, jest stosowane, gdy chcemy mieć kompletny indeks na wszystkie kolumny, aby uniknąć odczytu z oryginalnej tabeli.

W tym przypadku, wykorzystanie pierwszego indeksu z kolumnami INCLUDE powinno być wystarczające dla tego zapytania.

Zadanie 5 – Indeksy z filtrami

Celem zadania jest poznanie indeksów z filtrami.

Skopiuj tablicę BillOfMaterials do swojej bazy danych:

```

SELECT * INTO BillOfMaterials
FROM [AdventureWorks2017].[Production].BillOfMaterials

```

W tej części analizujemy zapytanie:

```

SELECT ProductAssemblyID, ComponentID, StartDate
FROM BillOfMaterials
WHERE EndDate IS NOT NULL
      AND ComponentID = 327
      AND StartDate >= '2010-08-05'

```

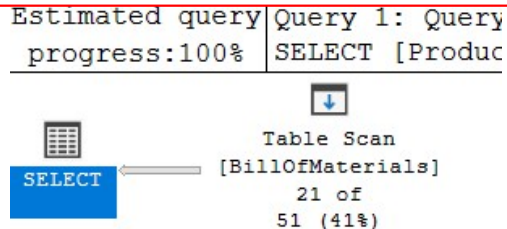
Zastosuj indeks:

```
CREATE NONCLUSTERED INDEX BillofMaterials_Cond_Idx
ON BillofMaterials (ComponentID, StartDate)
WHERE EndDate IS NOT NULL
```

Sprawdź czy działa. Wykonaj plan poniższego zapytania:

```
SELECT ProductAssemblyID, ComponentID, StartDate
FROM BillofMaterials
WHERE ComponentID = 327
AND StartDate > '2010-08-05'
```

Czy indeks został użyty? Dlaczego?



Indeks zawiera filtr WHERE EndDate IS NOT NULL, co oznacza, że indeks obejmuje tylko te wiersze, które spełniają ten warunek. Jednak w przypadku tego zapytania, filtr ten nie jest używany, ponieważ zapytanie używa tylko kolumn ComponentID i StartDate, a nie kolumny EndDate. Z tego powodu baza danych musi przeskanować całą tabelę, aby znaleźć pasujące wiersze, co jest kosztowne dla wydajności zapytania. W tym przypadku najlepszym rozwiązaniem byłoby stworzenie indeksu obejmującego kolumny ComponentID i StartDate, bez dodatkowego filtru WHERE.

Dygresja:

Ciekawym faktem, który można zauważyć to, że dla pierwszego polecenia tj.

```
SELECT ProductAssemblyID, ComponentID, StartDate
FROM BillofMaterials
WHERE EndDate IS NOT NULL
AND ComponentID = 327
AND StartDate >= '2010-08-05'
```

też indeks nie jest używany, ponieważ optymalizator wie, że lepiej przeskanować całą tabelę niż skorzystać z indeksu, w którym nie ma informacji o ProductAssemblyID zaciągania tej informacji za każdym razem.

Spróbuj wymusić indeks. Co się stało, dlaczego takie zachowanie?

Pojawia się błąd:

Query processor could not produce a query plan because of the hints defined in this query. Resubmit the query without specifying any hints and without using SET FORCEPLAN.

Polecenie wywołuje błąd, ponieważ indeks BillofMaterials_Cond_Idx zawiera klauzulę WHERE EndDate IS NOT NULL, która określa warunek, który musi zostać spełniony przez każdą krotkę w tabeli, aby została uwzględniona w indeksie. Jednak zapytanie nie zawiera tego warunku, więc indeks nie może być użyty do przyspieszenia wyszukiwania.

Punktacja

zadanie	pkt
1	2
2	2

3	2
4	2
5	2
razem	10