Indeksy, optymalizator – lab5

Imię i Nazwisko:	Bartłomiej Jamiołkowski, Ada Bodziony

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

Ważne/wymagane są komentarze.

Zamieść kod rozwiązania oraz zrzuty ekranu pokazujące wyniki, (dołącz kod rozwiązania w formie tekstowej/źródłowej)

Zwróć uwagę na formatowanie kodu

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 tabelę Customer do swojej bazy danych:

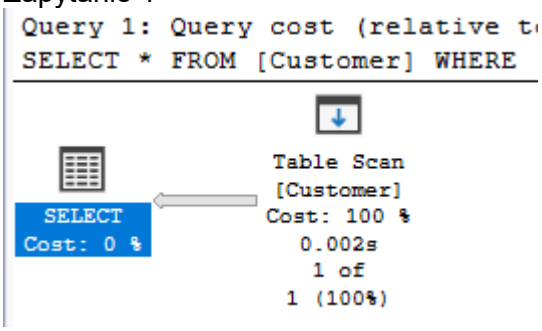
```
SELECT * INTO [Customer] FROM [AdventureWorks2017].[Sales].[Customer]
```

Wykonaj analizy zapytań:

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

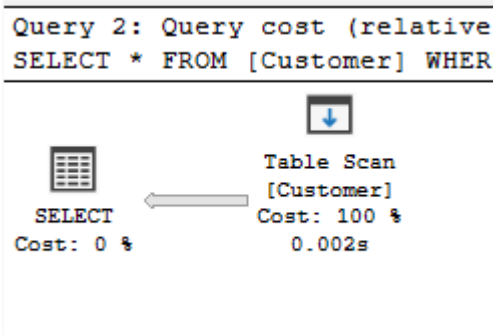
Zanotuj czas zapytania oraz jego koszt koszt:

Zapytanie 1



Czas: 0.002s
Koszt: 0.139158

Zapytanie 2



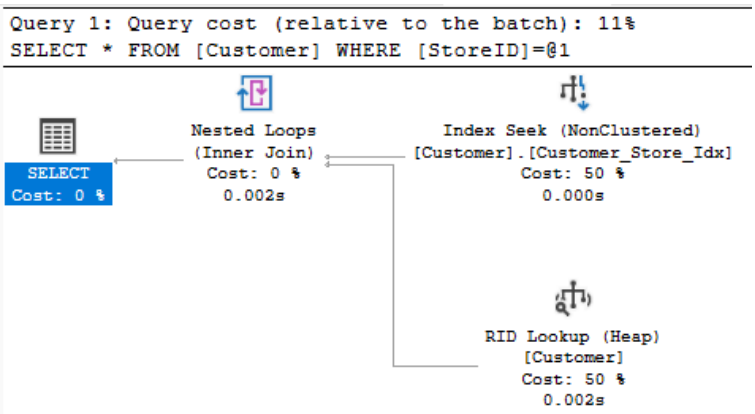
Czas: 0.002s
Koszt: 0.139158

Dodaj indeks:

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

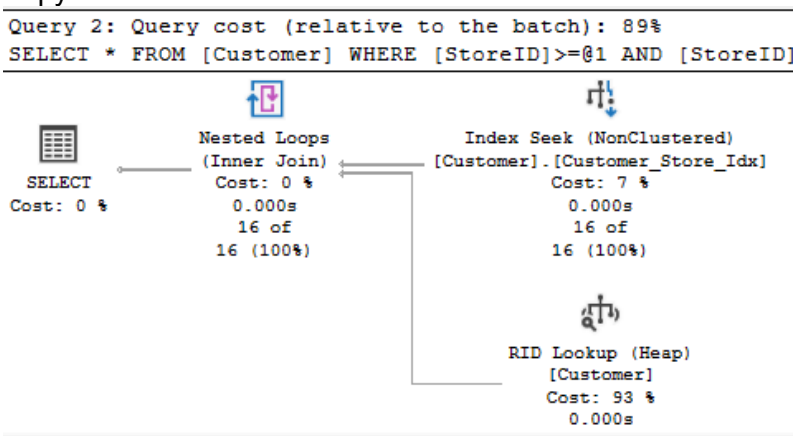
Jak zmienił się plan i czas? Czy jest możliwość optymalizacji?

Zapytanie 1



Czas: 0.002s

Zapytanie 2



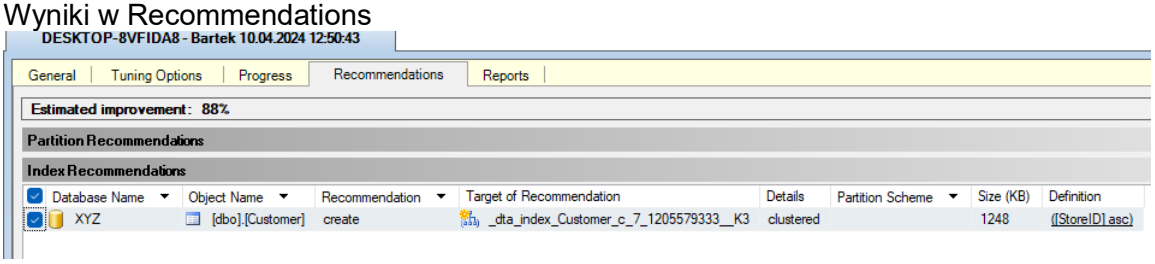
Czas: 0.000s

Po wprowadzeniu zmian zaokrąglony czas w zapytaniu 1 pozostał bez zmian, natomiast czas w zapytaniu 2 został skrócony.

W planach dotyczących zapytania 1 i zapytania 2 pojawiło się więcej operacji. Oba plany są do siebie podobne. Operacja Table Scan została zastąpiona operacjami: Nested Loops, Index Seek (Nonclustered) oraz RID Lookup.

Istnieje możliwość optymalizacji za pomocą Database Engine Tuning Advisor.

Wyniki w Recommendations



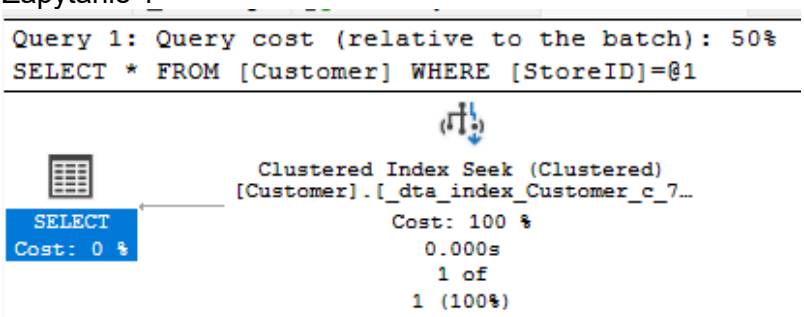
Implementacja tego wyniku znajduje się poniżej.

Dodaj indeks klastrowany:

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

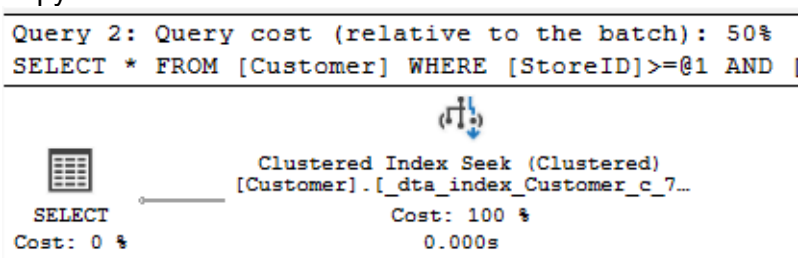
Czy zmienił się plan i czas? Skomentuj dwa podejścia w wyszukiwaniu krotek.

Zapytanie 1



Czas: 0.000s
Cost: 0.0032831

Zapytanie 2



Czas: 0.000s
Cost: 0.0032996

Po wprowadzeniu zmian czas w zapytaniu 1 zmalał do 0.000s . Najprawdopodobniej czas w zapytaniu 2 również zmalał, ale ze względu na zaokrąglenie do 3 miejsca po przecinku nie jest to już widoczne. Plany zapytań się zmieniły. Operacje Nested Loops, Index Seek (Nonclustered) oraz RID Lookup zostały zastąpione operacją Clustered Index Seek (Clustered).

Clustered Index sortuje i przechowuje wiersze danych tabeli lub widoku w kolejności opartej na kluczu Clustered Index. Clustered Index jest zaimplementowany jako struktura indeksu drzewa B, która obsługuje szybkie pobieranie wierszy na podstawie ich wartości klucza Clustered Index.

Nonclustered Index można zdefiniować w tabeli lub widoku za pomocą Clustered Index lub na stercie (Heap). Każdy wiersz indeksu w Nonclustered Index zawiera wartość klucza nieklastrowanego i lokalizator wierszy. Ten lokalizator wskazuje wiersz danych w Clustered Index lub stercie mający wartość klucza. Wiersze w indeksie są przechowywane w kolejności wartości kluczy indeksu, ale nie ma gwarancji, że wiersze danych będą w określonej kolejności, chyba że w tabeli zostanie utworzony Clustered Index.

Zadanie 2 – Indeksy zawierające dodatkowe atrybuty (dane z kolumn)

Celem zadania jest poznanie indeksów z przechowujących dodatkowe atrybuty (dane z kolumn)

Skopiuj tabelę Person do swojej bazy danych:

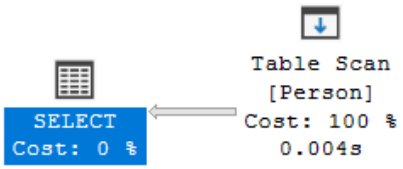
```
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
      ,[rowguid]
      ,[ModifiedDate]
INTO [Person]
FROM [AdventureWorks2017].[Person].[Person]
```

Wykonaj analizę planu dla trzech zapytań:

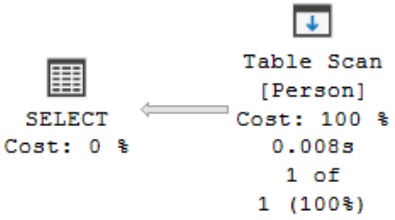
```
SELECT * FROM [Person] WHERE LastName = 'Agbonile'

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

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

Co można o nich powiedzieć?

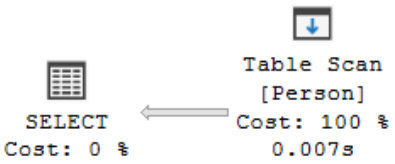
Query 1: Query cost (relative to the batch): 33%
SELECT * FROM [Person] WHERE [LastName]=@1



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



Zapytanie 1 – Estimated Subtree Cost 0,178585
Zapytanie 2 - Estimated Subtree Cost 0,178585
Zapytanie 3 - Estimated Subtree Cost 0,178585

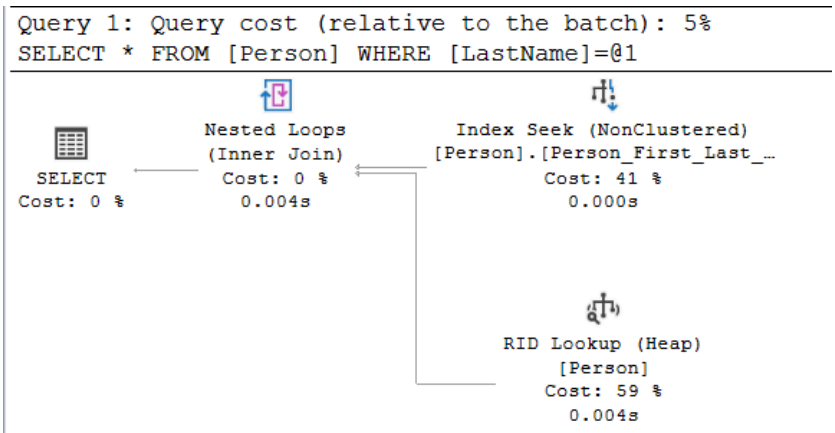
Koszty zapytań są identyczne, jak widać również czasy TableScan są bardzo zbliżone do siebie. Różnice w czasie wykonania nie przekraczają 0.004s.

Przygotuj indeks obejmujący te zapytania:

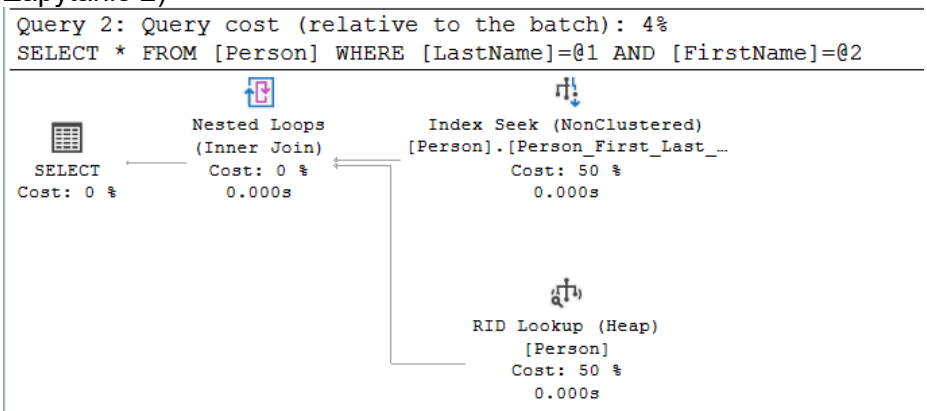
```
CREATE INDEX Person_First_Last_Name_Idx
ON Person(LastName, FirstName)
```

Sprawdź plan zapytania. Co się zmieniło?

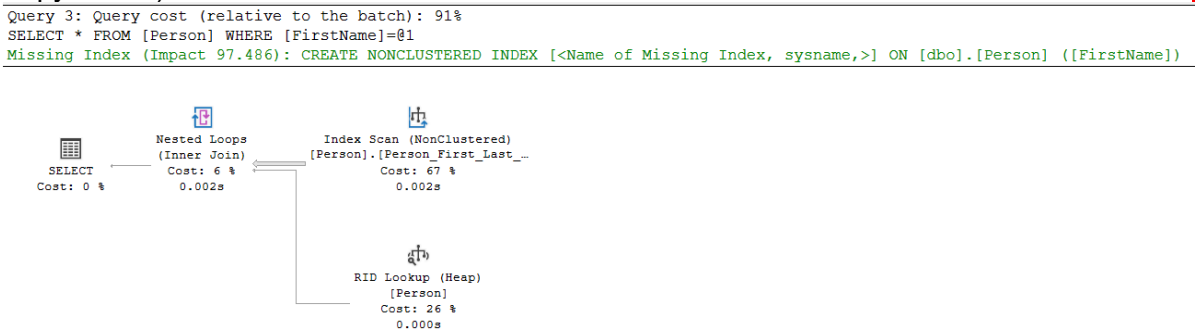
Dodano indeks z polecenia a następnie wywołano jeszcze raz zapytania:
Zapytanie 1)



Zapytanie 2)



Zapytanie 3)



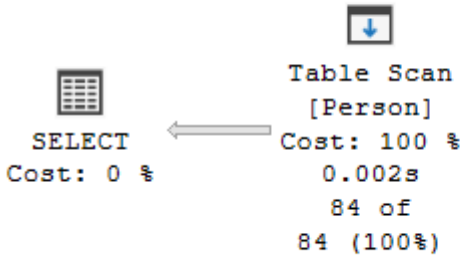
Czasy zapytań:
Zapytanie 1 – Estimated Subtree Cost 0,0080229
Zapytanie 2 - Estimated Subtree Cost 0,0065804
Zapytanie 3 - Estimated Subtree Cost 0,15377

Jak widać na planach zapytań każde z nich zostało wykonane z użyciem indeksu. W przypadku zapytań 1 i 2 – indeks pomógł znacząco obniżyć koszt zapytania. Natomiast w przypadku zapytania 3, czas obniżył się nieznacznie oraz otrzymaliśmy sugestię stworzenia indeksu zawierającego FirstName. Wynika to z tego, że kolejność kolumn w indeksie może wpłynąć na sposób, w jaki dane są porządkowane w węzłach drzewa B+, co z kolei może wpłynąć na efektywność indeksu – w tym przypadku spadek jest znaczny.

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?

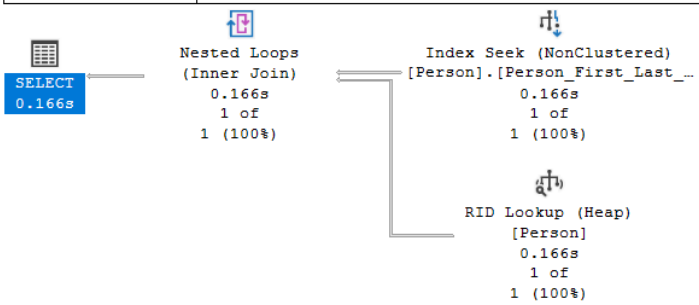
Zapytanie 1)

```
Query 2: Query cost (relative to the batch): 96%
SELECT * FROM [Person] WHERE [LastName]=@1
```



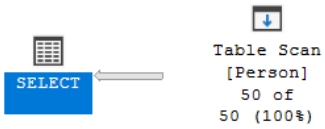
Zapytanie 2)

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



Zapytanie3)

Estimated query progress:100%	Query 2: Query cost (relative to the batch): 96% SELECT * FROM [Person] WHERE [FirstName]=@1 Missing Index (Impact 97.9456): CREATE NONCLUSTERED INDEX [<Nam



Zapytanie 1 – Estimated Subtree Cost 0,178585
Zapytanie 2 - Estimated Subtree Cost 0,0065804
Zapytanie 3 - Estimated Subtree Cost 0,178585

W przypadku zapytań 1 i 3 nie został wykorzystany utworzony index, co różni te plany wykonania od tych dla poprzednich warunków w klauzuli where. Jest to spowodowane zapewne tym, że w tabeli Person znajdują się po jednym rekordzie dla:
`LastName = 'Agbonile'` oraz `FirstName = 'Osarumwense'`
Oraz Natomiast rekordów z wartością `LastName = 'Price'` mamy 84, a z wartościami `FirstName = 'Angela'` – 50. Więc na podstawie tych statystyk na temat danych optymalizator zapytań może zdecydować się na użycie indeksu przykładowo w przypadku 'Agbonile', ale nie w przypadku 'Price', gdyż będzie to bardziej opłacalne.

Zadanie 3

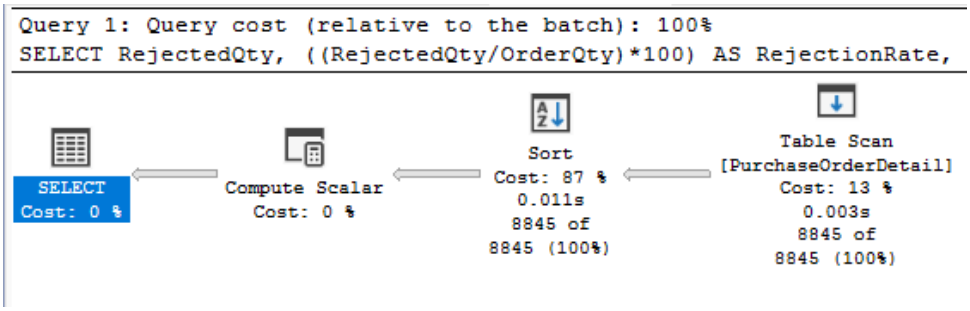
Skopiuj tabelę PurchaseOrderDetail do swojej bazy danych:

```
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
```

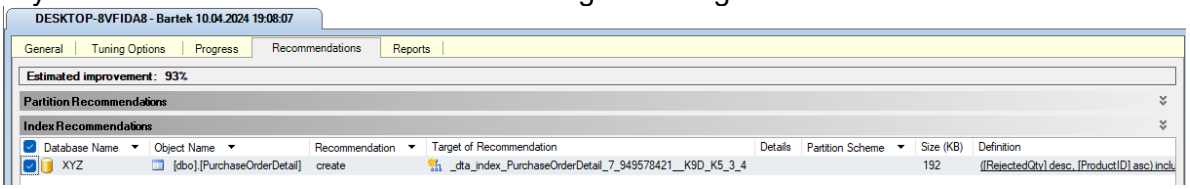
Która część zapytania ma największy koszt?



Największy koszt 87% ma operacja Sort.

Jaki indeks można zastosować aby zoptymalizować koszt zapytania? Przygotuj polecenie tworzące index.

Wyniki w Recommendations w Database Engine Tuning Advisor.

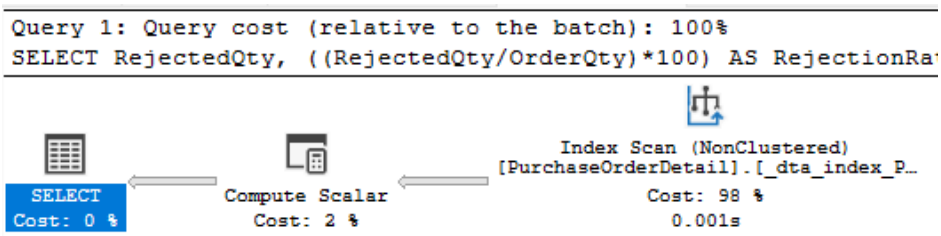


Do optymalizacji kosztu zapytania można zastosować Nonclustered Index.

Polecenie tworzące index

```
use [XYZ]
go

CREATE NONCLUSTERED INDEX [_dta_index_PurchaseOrderDetail_7_949578421__K9D_K5_3_4]
ON [dbo].[PurchaseOrderDetail]
(
	[RejectedQty] DESC,
	[ProductID] ASC
)
INCLUDE([DueDate],[OrderQty]) WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLI-
NE = OFF) ON [PRIMARY]
go
```

Ponownie wykonaj analizę zapytania:



Po wprowadzeniu zmian plan zapytań zmienił się. Operacje Sort i Table Scan zostały zastąpione operacją Index Scan (Nonclustered).

Zadanie 4

Celem zadania jest porównanie indeksów zawierających wszystkie kolumny oraz indeksów przechowujących dodatkowe dane (dane z kolumn).

Skopiuj tabelę Address do swojej bazy danych:

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

W tej części będziemy analizować następujące zapytanie:

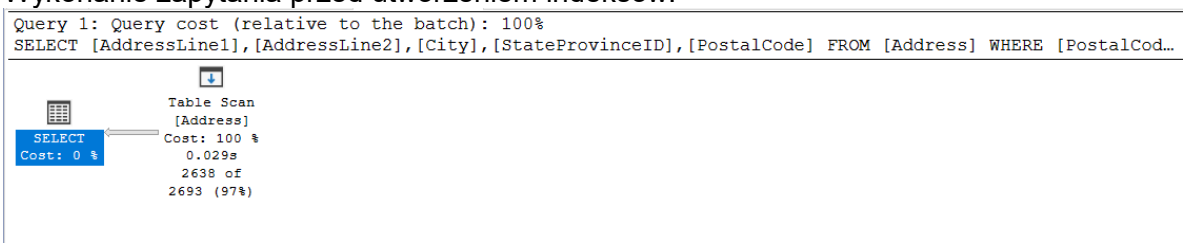
```
SELECT AddressLine1, AddressLine2, City, StateProvinceID, PostalCode
FROM Address
WHERE PostalCode BETWEEN N'98000' and N'99999'
```

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:

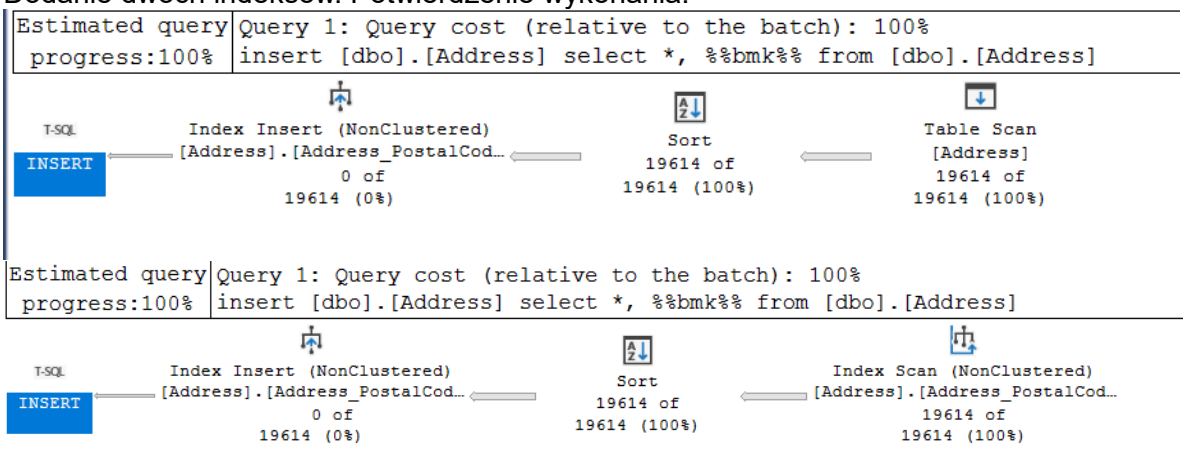
Wykonanie zapytania przed utworzeniem indeksów:



Misc	
Actual Number of Rows for All Executions	2638
Cached plan size	24 KB
CardinalityEstimationModelVersion	150
CompileCPU	1
CompileMemory	184
CompileTime	1
CompletionEstimate	1
ElapsedTime	176
Estimated Number of Rows for All Execution	0
Estimated Number of Rows Per Execution	2693,25
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,278191

Zapytanie wykonało się bez wykorzystania indeksu.

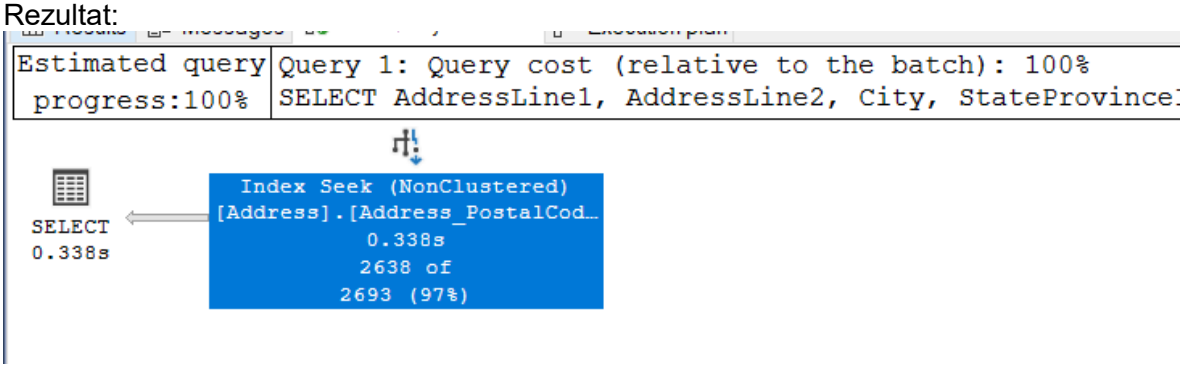
Dodanie dwóch indeksów. Potwierdzenie wykonania:



Wykonujemy zapytanie gdzie wymuszamy zawarcie indeksu Address_PostalCode_1:

```
SELECT AddressLine1, AddressLine2, City, StateProvinceID, PostalCode
FROM Address
WITH(INDEX(Address_PostalCode_1))
WHERE PostalCode BETWEEN N'98000' and N'99999'
```

Rezultat:

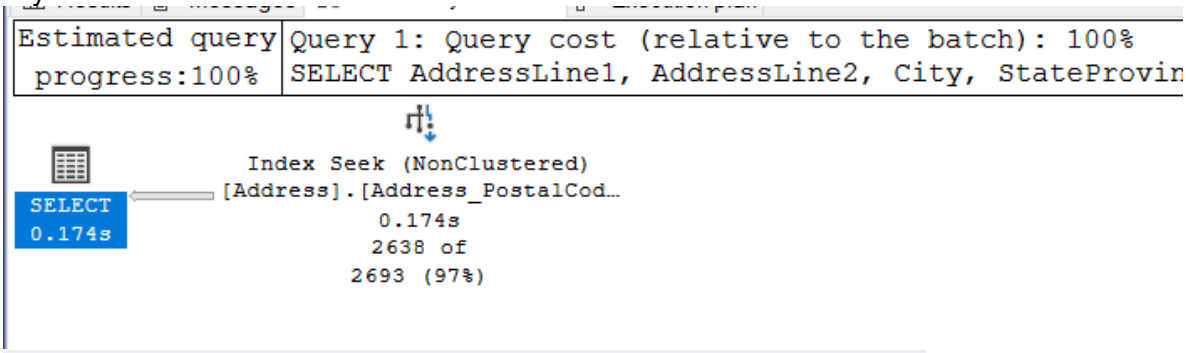


Misc	
Actual Number of Rows for All Executions	2638
Cached plan size	24 KB
CardinalityEstimationModelVersion	150
CompileCPU	3
CompileMemory	176
CompileTime	5
CompletionEstimate	1
ElapsedTime	338
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	2693,25
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,0284668

Wykonujemy zapytanie gdzie wymuszamy zawarcie indeksu Address_PostalCode_2:

```
SELECT AddressLine1, AddressLine2, City, StateProvinceID, PostalCode
FROM Address
WITH(INDEX(Address_PostalCode_2))
WHERE PostalCode BETWEEN N'98000' and N'99999'
```

Wynik:



Misc	
Actual Number of Rows for All Executions	2638
Cached plan size	24 KB
CardinalityEstimationModelVersion	150
CompileCPU	3
CompileMemory	176
CompileTime	3
CompletionEstimate	1
ElapsedTime	174
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	2693,25
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,0284668

Wykorzystanie drugiego z indeksów jest przyspieszyło dwukrotnie zapytanie (Estimated Subtree Cost pozostał taki sam)
Sprawdzono też zapytanie bez wymuszania indeksu i optymalizator również wybrał wydajniejszą opcję czyli z indeksem Address_PostalCode_2.
W przypadku indeksu z include wartości w nim zawarte znajdują się w liściach B+ drzewa. Natomiast indeks, gdzie wszystkie kolumny należą do klucza, zawiera je także w każdym węźle. Co może wpływać negatywnie na rozmiar indeksu.
Obydwa indeksy pokrywają zapytanie – zawierają wszystkie kolumny do których się odwołuje. Jednak w tym konkretnym przypadku jeśli chodzi o czas wykonania okazał się

index bez include.

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 na to wpływają?

	IndexName	IndexSizeKB
1	Address_PostalCode_1	1784
2	Address_PostalCode_2	1808

Tak jak napisano wyżej:
W przypadku indeksu Address_Postal_Code_1 (include) wartości w nim zawarte znajdują się w liściach B+ drzewa. Natomiast indeks Address_Postal_Code_1 , gdzie wszystkie kolumny należą do klucza, zawiera je także w każdym węźle, co zwiększa rozmiar indexu. Co ważne sumarycznie patrząc indeksy zawierają ten sam zbiór kolumn.

Zadanie 5 – Indeksy z filtrami

Celem zadania jest poznanie indeksów z filtrami.
Skopiuj tabelę 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
```

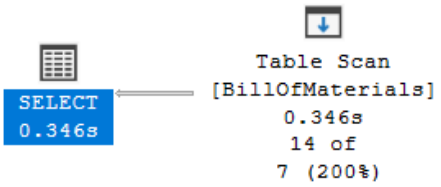
Sprawdź czy działa.
Przeanalizuj plan dla poniższego zapytania:

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

Czy indeks został użyty? Dlaczego?

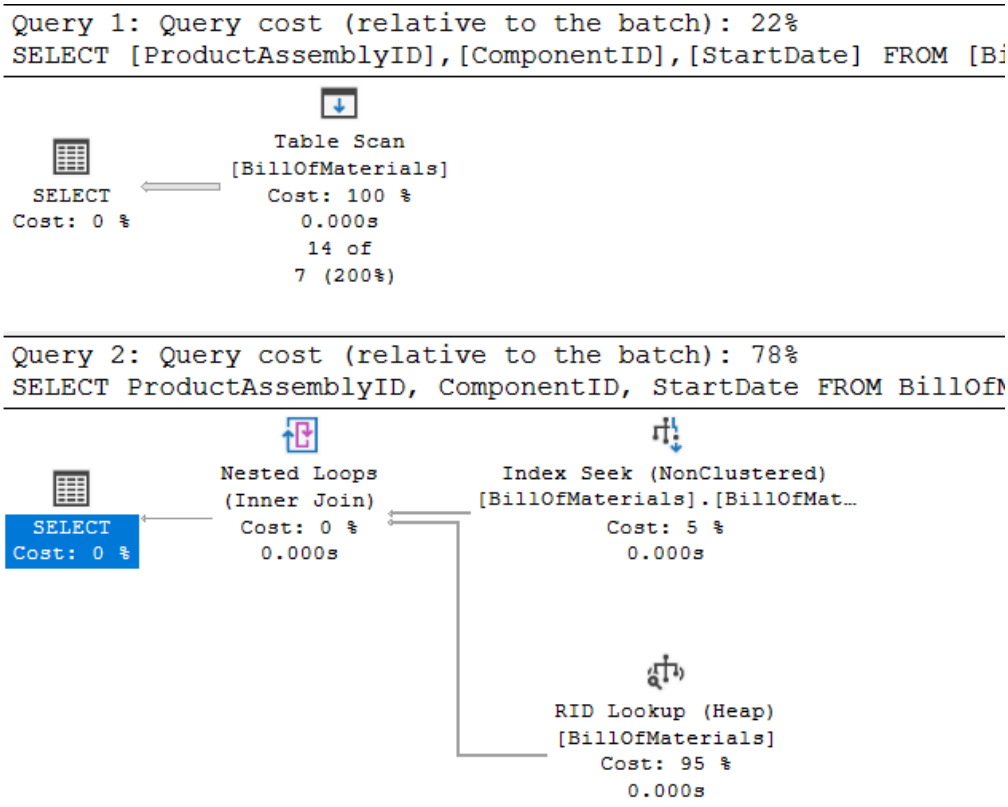
Wykonanie zapytania po dodaniu indeksu:

Estimated query progress:100%	Query 1: Query cost (relative to the batch): 100% SELECT ProductAssemblyID, ComponentID, StartDate FROM



Misc	
Actual Number of Rows for All Executions	14
Cached plan size	24 KB
CardinalityEstimationModelVersion	150
CompileCPU	7
CompileMemory	352
CompileTime	7
CompletionEstimate	1
ElapsedTime	346
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	6,6439
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,020303

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

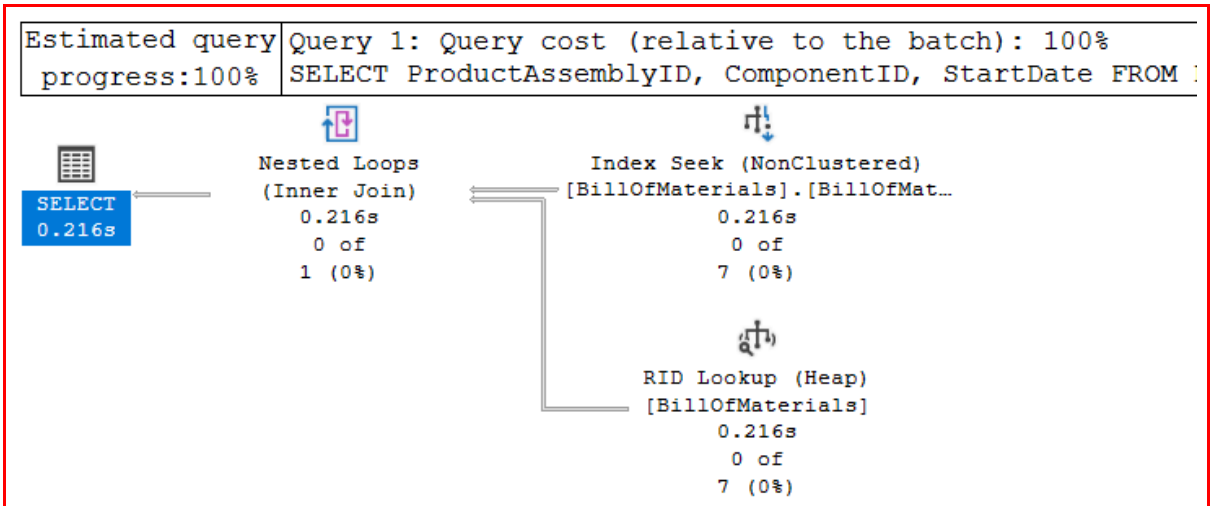


Bez indeksu – Estimated Subtree Cost - 0,020303
Z indeksem – Estimated Subtree Cost - 0,0723578

Porównanie planów zapytań bez wykorzystania indeksu i z indeksem. Jak widać wykorzystanie indeksu w tym przypadku zwiększyło dość znacząco koszt. Możliwe, że indeksowana kolumna ComponentId nie ogranicza w tym przypadku do oczekiwanej pod kątem wydajności, ilości rekordów. A co za tym idzie dokonuje się przeszukanie większego zakresu indeksu, co może okazać się bardziej kosztowne niż zapytanie bez jego wykorzystania. Dla ComponentID = 327 mamy 95 rekordów w tabeli BillOfMaterials, spróbujmy wykonać zapytanie dla ComponentId z mniejszą ilością rekordów w tej tabeli, np. 490 – wartość dla 11 rekordów.

```
SELECT ProductAssemblyID, ComponentID, StartDate
FROM BillOfMaterials
WHERE EndDate IS NOT NULL
    AND ComponentID = 490
    AND StartDate >= '2010-08-05'
```

Dla powyższego zapytania, gdzie zmieniono jedynie warunek na ComponentID, serwer zdecydował się na wykorzystanie indeksu przy wykonaniu zapytania:



Co wskazuje, że w tym przypadku wykorzystanie indeksu już było korzystniejsze.

Punktacja

zadanie	pkt
1	2
2	2
3	2
4	2
5	2
razem	10