

Indeksy, optymalizator – lab6-7

Imię i Nazwisko: Bartłomiej Jamiołkowski, Ada Bodziony

Swoje odpowiedzi wpisuj w **czerwone pola**. Preferowane są zrzuty ekranu, **wymagane** komentarze.

Oprogramowanie - co jest potrzebne?

Do wykonania ćwiczenia potrzebne są:

- **MS SQL Server** wersja co najmniej 2016,
- przykładowa baza danych **AdventureWorks2017**.

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/indexes/create-filtered-indexes>

–

Zadanie 1

Skopiuj tabelę Product do swojej bazy danych:

```
SELECT * INTO Product FROM [AdventureWorks2017].[Production].Product
```

Stwórz indeks z warunkiem przedziałowym :

```
CREATE NONCLUSTERED INDEX Product_Range_Idx
ON Product (ProductSubcategoryID, ListPrice) Include (Name)
WHERE ProductSubcategoryID >= 27 AND ProductSubcategoryID <= 36
```

Sprawdź, czy indeks jest użyty w zapytaniu:

```
SELECT Name, ProductSubcategoryID, ListPrice
FROM Product
WHERE ProductSubcategoryID >= 27 AND ProductSubcategoryID <= 36
```

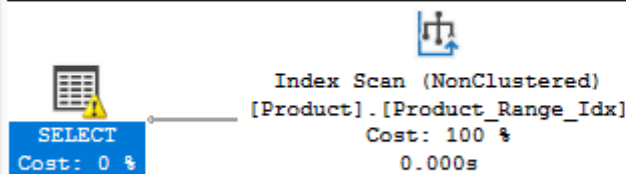
Sprawdź, czy indeks jest użyty w zapytaniu, który jest dopełnieniem zbioru:

```
SELECT Name, ProductSubcategoryID, ListPrice
FROM Product
WHERE ProductSubcategoryID < 27 OR ProductSubcategoryID > 36
```

Skomentuj oba zapytania. Czy indeks został użyty w którymś zapytaniu, dlaczego? Czy indeks nie został użyty w którymś zapytaniu, dlaczego? Jak działają indeksy z warunkiem?

Zapytanie 1

Query 1: Query cost (relative to the batch): 20%
SELECT [Name],[ProductSubcategoryID],[ListPrice]

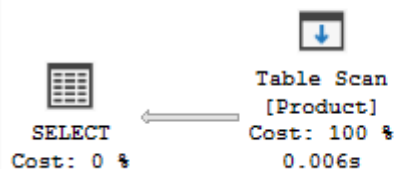


Zapytanie 1 ma czas: 0.000s oraz cost: 0.0033007. Stanowi 20% kosztu obu zapytań, czyli jest wydajniejsze.

Execution plan pokazuje, że indeks został użyty w zapytaniu 1. Stało się tak, ponieważ indeks z warunkiem przedziałowym to tak zwany Filtered Index. Ten index jest zoptymalizowanym, nieklastrowanym indeksem magazynu wierszy opartym na dysku. Szczególnie jest on dostosowany do zapytań wybierających z dobrze zdefiniowanego podzbioru danych. W tym zapytaniu klauzula WHERE zapytania jest podzbiorem klauzuli WHERE filtrowanego indeksu, dlatego korzysta z Filtered Index.

Zapytanie 2

Query 2: Query cost (relative to the batch): 80%
SELECT Name, ProductSubcategoryID, ListPrice FROM



Zapytanie 2 ma czas: 0.006s oraz cost: 0.013466. Stanowi 80% kosztu obu zapytań, czyli jest mniej wydajne.

Execution plan pokazuje, że indeks nie został użyty w zapytaniu 2. Stało się tak, ponieważ w tym zapytaniu klauzula WHERE zapytania nie jest podzbiorem klauzuli WHERE filtrowanego indeksu, dlatego Filtered Index nie jest tu wykorzystywany.

Indeks z warunkiem to zoptymalizowany, nieklastrowany indeks magazynu wierszy oparty na dysku, szczególnie dostosowany do zapytań wybierających z dobrze zdefiniowanego podzbioru danych. Używa predykatu filtru do indeksowania części wierszy w tabeli. Dobrze zaprojektowany indeks filtrowany może poprawić wydajność zapytań i zmniejszyć koszty utrzymania indeksu i przechowywania w porównaniu z indeksami pełnotabelowymi.

Zadanie 2 – indeksy klastrujące

Celem zadania jest poznanie indeksów klastrujących.

Skopiuj ponownie tabelę SalesOrderHeader do swojej bazy danych:

```
SELECT * INTO [SalesOrderHeader2] FROM
[AdventureWorks2017].[Sales].[SalesOrderHeader]
```

Wypisz sto pierwszych zamówień:

```
SELECT TOP 1000 * FROM SalesOrderHeader2
ORDER BY OrderDate
```

Stwórz indeks klastrujący według OrderDate:

```
CREATE CLUSTERED INDEX Order_Date2_Idx ON SalesOrderHeader2 (OrderDate)
```

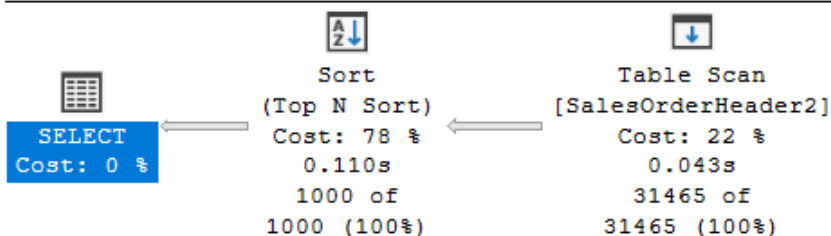
Wypisz ponownie sto pierwszych zamówień. Co się zmieniło?

Wypisanie pierwszych 1000 zamówień. Wynik:

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag
1	43659	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
2	43660	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
3	43661	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
4	43662	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
5	43663	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
6	43664	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
7	43665	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
8	43666	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
9	43667	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
10	43668	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
11	43669	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
12	43670	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
13	43671	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
14	43672	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
15	43673	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
16	43674	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
17	43675	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
18	43676	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
19	43677	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0
20	43678	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0

Oraz plan wykonania:

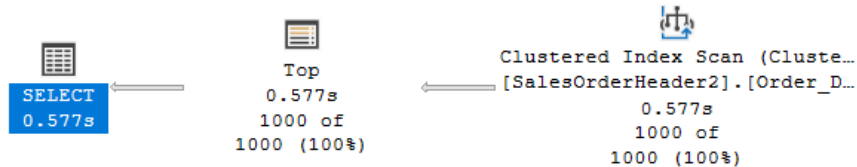
Query 1: Query cost (relative to the batch): 100%
SELECT TOP 1000 * FROM SalesOrderHeader2 ORDER BY OrderDate



Estimated Subtree Cost: 2,78791

Dodano indeks i powtórzono zapytanie:

Estimated query progress:100%	Query 1: Query cost (relative to the batch): 100% SELECT TOP 1000 * FROM SalesOrderHeader2 ORDER BY OrderDate
-------------------------------	--



Estimated Subtree Cost: 0,0233677

Jak widać indeks został wykorzystany w zapytaniu, skrócił koszt wykonania ponad 10-ciokrotnie. Zapytanie polega na wypisaniu posortowanych danych w zależności od OrderDate, indeks usprawnił odczyt 1000 kolumn z najmniejszymi wartościami OrderDate.

Sprawdź zapytanie:

```
SELECT TOP 1000 * FROM SalesOrderHeader2
WHERE OrderDate BETWEEN '2010-10-01' AND '2011-06-01'
```

Dodaj sortowanie według OrderDate ASC i DESC. Czy indeks działa w obu przypadkach. Czy wykonywane jest dodatkowo sortowanie?










Zostały wykonane 3 poniższe zapytania:

```
-- Wersja bez sortowania
SELECT TOP 1000 * FROM SalesOrderHeader2
WHERE OrderDate BETWEEN '2010-10-01' AND '2011-06-01'
```

```
-- Wersja z sortowaniem DESC
SELECT TOP 1000 * FROM SalesOrderHeader2
WHERE OrderDate BETWEEN '2010-10-01' AND '2011-06-01'
order by OrderDate desc
```

```
-- Wersja z sortowaniem ASC
SELECT TOP 1000 * FROM SalesOrderHeader2
WHERE OrderDate BETWEEN '2010-10-01' AND '2011-06-01'
order by OrderDate asc
```

Plany wykonać:

Estimated query progress:100%	Query 1: Query cost (relative to the batch): 33% SELECT TOP 1000 * FROM SalesOrderHeader2 WHERE OrderI		
 SELECT 0.529s	 Top 0.529s 47 of 51 (92%)	 Clustered Index Seek (Cluste... [SalesOrderHeader2].[Order_D... 0.529s 47 of 51 (92%)	
Estimated query progress:100%	Query 2: Query cost (relative to the batch): 33% SELECT TOP 1000 * FROM SalesOrderHeader2 WHERE OrderI		
 SELECT 0.661s	 Top 0.661s 47 of 51 (92%)	 Clustered Index Seek (Cluste... [SalesOrderHeader2].[Order_D... 0.661s 47 of 51 (92%)	
Estimated query progress:100%	Query 3: Query cost (relative to the batch): 33% SELECT TOP 1000 * FROM SalesOrderHeader2 WHERE OrderI		
 SELECT 0.529s	 Top 0.529s 47 of 51 (92%)	 Clustered Index Seek (Cluste... [SalesOrderHeader2].[Order_D... 0.529s 47 of 51 (92%)	
Estimated Subtree Cost: 0,0040837 – równy dla wszystkich trzech Jak widać sortowanie desc i asc nie ma wpływu na wykorzystanie indeksu, ponieważ dane przechowywane w indeksie już są uporządkowane.			

Zadanie 3 – indeksy *column store*

Celem zadania jest poznanie indeksów typu column store–

Utwórz tabelę testową:

```
CREATE TABLE [dbo].[SalesHistory] (
    [SalesOrderID] [int] NOT NULL,
    [SalesOrderDetailID] [int] NOT NULL,
    [CarrierTrackingNumber] [nvarchar](25) NULL,
    [OrderQty] [smallint] NOT NULL,
    [ProductID] [int] NOT NULL,
    [SpecialOfferID] [int] NOT NULL,
    [UnitPrice] [money] NOT NULL,
    [UnitPriceDiscount] [money] NOT NULL,
    [LineTotal] [numeric](38, 6) NOT NULL,
    [rowguid] [uniqueidentifier] NOT NULL,
    [ModifiedDate] [datetime] NOT NULL
) ON [PRIMARY]
GO
```

Założ indeks:

```
CREATE CLUSTERED INDEX [SalesHistory_Idx]
ON [SalesHistory]([SalesOrderDetailID])
```

Wypełnij tablicę danymi:

(UWAGA! 'GO 100' oznacza 100 krotne wykonanie polecenia. Jeżeli podejrzewasz, że Twój serwer może to zbyt przeciążyć, zacznij od GO 10, GO 20, GO 50 (w sumie już będzie 80))

```
INSERT INTO SalesHistory
SELECT SH.*
FROM [AdventureWorks2017].[Sales].SalesOrderDetail SH
GO 100
```

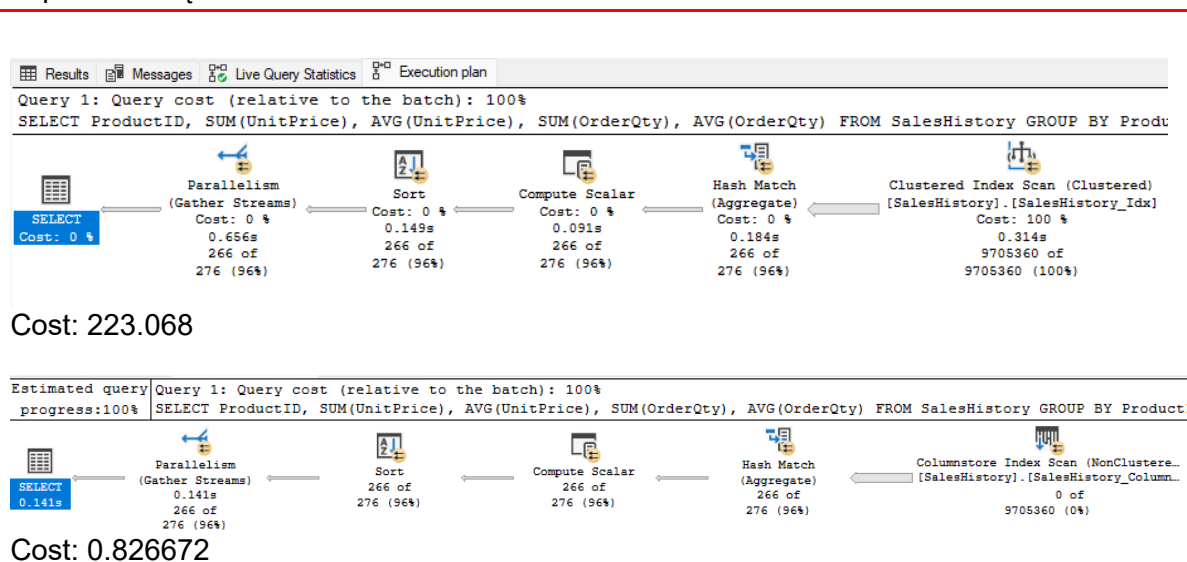
Sprawdź jak zachowa się zapytanie, które używa obecnego indeksu:

```
SELECT ProductID, SUM(UnitPrice), AVG(UnitPrice), SUM(OrderQty),
AVG(OrderQty)
FROM SalesHistory
GROUP BY ProductID
ORDER BY ProductID
```

Założ indeks typu ColumnStore:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX SalesHistory_ColumnStore
ON SalesHistory(UnitPrice, OrderQty, ProductID)
```

Sprawdź różnicę pomiędzy przetwarzaniem w zależności od indeksów. Porównaj plany i opisz różnicę.



Różnica między Execution plans polega przede wszystkim na tym, że w pierwszym planie występuje operacja Clustered Index Scan (Clustered) a w drugim planie ta operacja jest zastąpiona Columnstore Index Scan (NonClustered). Kolejną widoczną różnicą są zmniejszone koszty oraz czasy w planie drugim w porównaniu z planem pierwszym. Dzieje się tak, ponieważ Columnstore index przechowuje dane i zarządza nimi, korzystając z przechowywania danych w oparciu o kolumny i przetwarzania zapytań w oparciu o kolumny. Użycie tych indeksów prowadzi do znacznego wzrostu wydajności zapytań.

Zadanie 4 – własne eksperymenty

Należy zaprojektować tabelę w bazie danych, lub wybrać dowolny schemat danych (poza używanymi na zajęciach), a następnie wypełnić ją danymi w taki sposób, aby zrealizować poszczególne punkty w analizie indeksów. Warto wygenerować sobie tabele o większym rozmiarze.

Do analizy, proszę uwzględnić następujące rodzaje indeksów:

- Klastrowane (np. dla atrybutu nie będącego kluczem głównym)
- Nieklastrowane
- Indeksy wykorzystujące kilka atrybutów, indeksy include
- Filtered Index (Indeks warunkowy)
- Kolumnowe

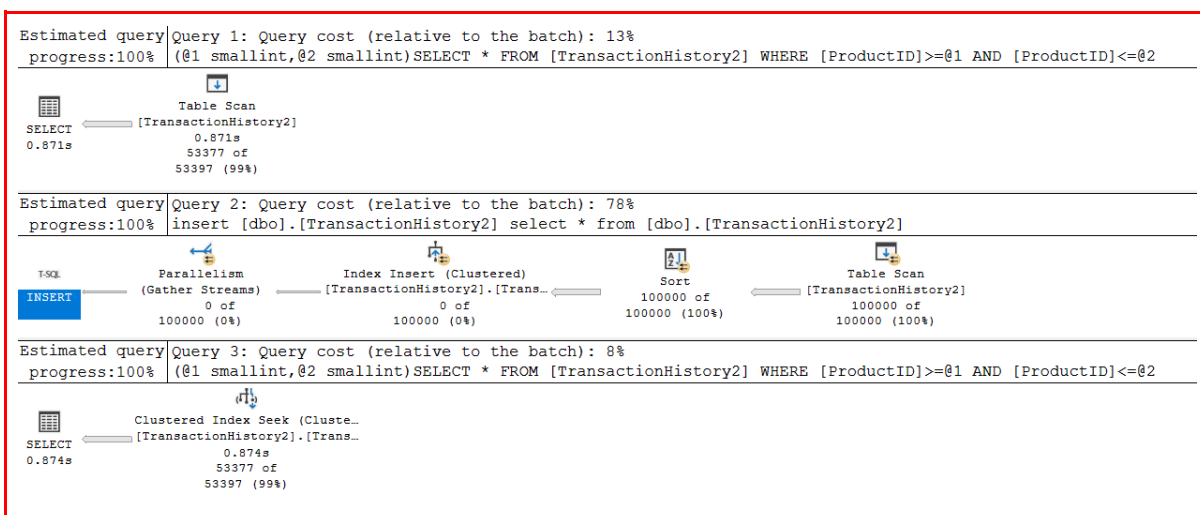
Analiza

Proszę przygotować zestaw zapytań do danych, które:

- wykorzystują poszczególne indeksy
- które przy wymuszeniu indeksu działają gorzej, niż bez niego (lub pomimo założonego indeksu, tabela jest w pełni skanowana)

Odpowiedź powinna zawierać:

- Schemat tabeli
- Opis danych (ich rozmiar, zawartość, statystyki)
- Trzy indeksy:
 - Opis indeksu
 - Przygotowane zapytania, wraz z wynikami z planów (zrzuty ekranów)
 - Komentarze do zapytań, ich wyników
 - Sprawdzenie, co proponuje Database Engine Tuning Advisor (porównanie czy udało się Państwu znaleźć odpowiednie indeksy do zapytania)



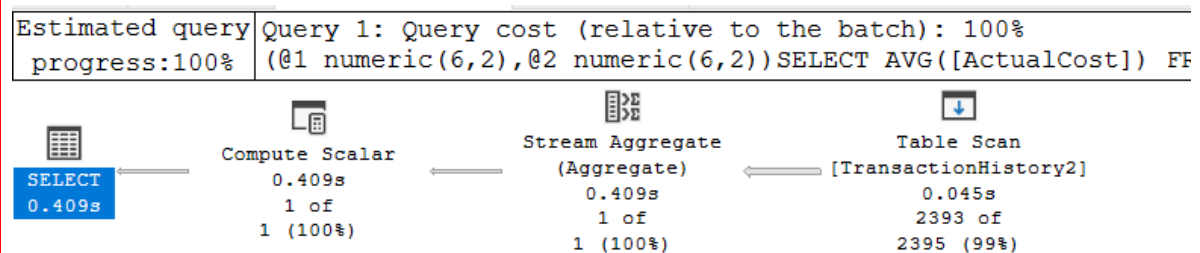
```
CREATE CLUSTERED INDEX TransactionHistory2_Idx ON TransactionHistory2(ProductId)
select *
from TransactionHistory2
where ProductID between 700 and 900
```

2. Indeksy kolumnowe analiza

a) Działanie na zapytaniach agregujących dane. Testowane zapytanie:

```
select avg(ActualCost)
from TransactionHistory2
where ActualCost between 1000.00 and 1500.00
```

Plan i koszt wykonania bez indeksów:



Actual Number of Rows for All Executions	1
Cached plan size	64 KB
CardinalityEstimationModelVersion	150
CompileCPU	5
CompileMemory	360
CompileTime	5
CompletionEstimate	1
Degree of Parallelism	1
ElapsedTime	0
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	1
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,0243808
Memory Grant	1088 KB

Estimated Subtree Cost: 0,72309

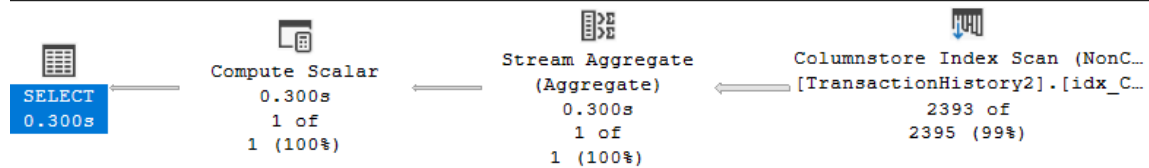
Następnie dodano dwa indeksy, kolejno kolumnowy i tradycyjny:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX idx_ColumnActualCost ON TransactionHistory2
(ActualCost);
```

```
CREATE NONCLUSTERED INDEX idx_ActualCost ON TransactionHistory2 (ActualCost);
```

Następnie wykonano ponownie powyższe zapytanie:

Estimated query progress:100% Query 1: Query cost (relative to the batch): 100%
select avg(ActualCost) from TransactionHistory2 where ActualCo:

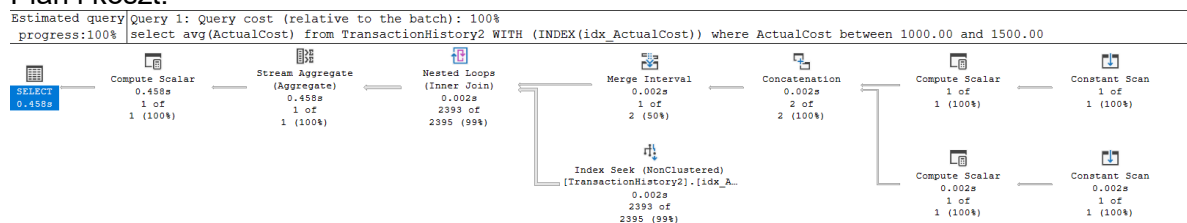


Estimated Subtree Cost: 0,0243808

Jak widać z dostępnych indeksów został wykorzystany ten kolumnowy. A także dzięki zastosowaniu indeksu zmniejszył się znacznie koszt zapytania. Jest to oczekiwane działanie, ze względu na to, sposób przechowywania danych, który powoduje, że optymalizują się wykonania zapytań zawierających agregacje.

Wymuśmy teraz działanie indeksu tradycyjnego:

Plan i koszt:



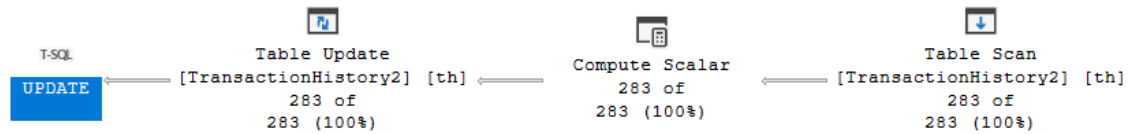
Actual Number of Rows for All Executions	1
Cached plan size	40 KB
CardinalityEstimationModelVersion	150
CompileCPU	1
CompileMemory	184
CompileTime	1
CompletionEstimate	1
ElapsedTime	458
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	1
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0,0117989

Jak się okazuje koszt wykonania zapytania jest niższy niż w przypadku indeksu kolumnowego. Jednak bez wymuszenia tego indeksu, zapytanie korzysta z wersji kolumnowej. Prawdopodobnie wpływa na to czas zapytania, który jest korzystniejszy dla wersji kolumnowej.

b) Przykład działania update

Bez indeksów:

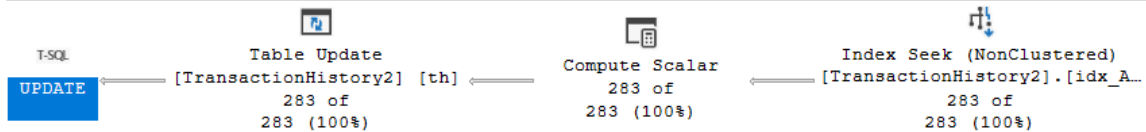
Estimated query progress:100%	Query 1: Query cost (relative to the batch): 100% update th set Quantity = 5 from TransactionHistory2 th where ActualC
-------------------------------	---



Estimated Subtree Cost: 0,740466

Indeks tradycyjny:

Estimated query progress:100%	Query 1: Query cost (relative to the batch): 100% update th set Quantity = 5 from TransactionHistory2 th where ActualC
-------------------------------	---



Estimated Subtree Cost: 0,0224069

W sytuacji gdy mamy na bazie jedynie kolumnową wersję tego indeksu, optymalizator nie bierze go pod uwagę. Niestety nie mamy możliwości w zapytaniach update wymuszenia indeksu do użycia przy wykonywaniu. Ale możemy się spodziewać, że koszt zapytania byłby znacząco większy ze względu na dodatkowe nakłady obliczeniowe przeznaczone na aktualizację indeksów, spowodowane inną, kolumnową strukturą

Punktacja

zadanie	pkt
1	2
2	2
3	2
4	10
razem	16