Indeksy, optymalizator
Lab 6-7

Imię i nazwisko:

- **Szymon Budziak**
- **Piotr Ludynia**

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

Swoje odpowiedzi wpisuj w miejsca oznaczone jako:

> Wyniki:

```
--  ...
```

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 lab6.

```
create database lab5
go

use lab5
go
```

Zamiast lab5 musieliśmy nazwać bazę lab6 ze względu na kolizję nazw z poprzednim laboratorium:

```
create database lab6
go

use lab6
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
```

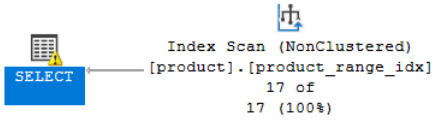
Sprawdź, czy indeks jest użyty w zapytaniu, który jest dopełnieniem zbioru:

```
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?

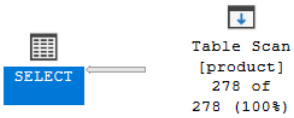
> W pierwszym zapytaniu indeks zostaje użyty. Zapytanie zwraca 17 wartości.

Execution Plan 1



> Jednak drugie zapytanie wykonuje się bez użycia indeksu. Wybierane jest 278 wartości.

Execution Plan 2



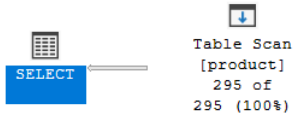
> Indeks jest użyty w zapytaniu, które zawiera warunek użyty przy tworzeniu go a w przeciwnym przypadku nie. Dzieje się tak, ponieważ optymalizator sprawdza, czy warunek z zapytania jest obecny w indeksie i na jego podstawie decyduje, czy użyje indeksu w wykonaniu zapytania. Wniosek: Tylko pierwsze zapytanie wykorzystuje indeks, drugie nie.

```
--  sprawdzany warunek:
--[...]
where productsubcategoryid >= 27 and productsubcategoryid <= 36
--[...]
```

> Przetestowaliśmy jeszcze jedno zapytanie. Zamieniamy koniunkcję na alternatywę. Cały zbiór będący wynikiem pierwszego zapytania jest w nim zawarty, jednak występują też tam wartości z poza niego (na przykład takie w których `productsubcategoryid` jest większe mniejsze niż 27 a `productsubcategoryid` dalej jest mniejsze bądź równe 36). W tym przypadku indeks również nie został użyty.

```
select name, productsubcategoryid, listprice
from product
where productsubcategoryid >= 27 or productsubcategoryid <= 36
```

Execution Plan 3



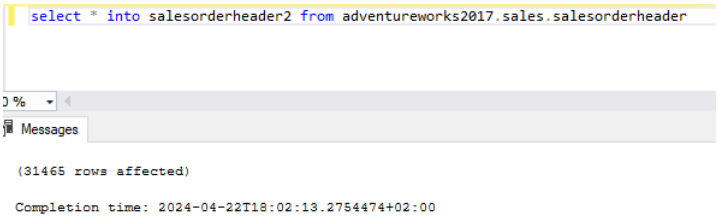
Zadanie 2 – indeksy klastrujące

Celem zadania jest poznanie indeksów klastrujących

Skopiuj ponownie tabelę SalesOrderHeader do swojej bazy danych:

```
select * into salesorderheader2 from adventureworks2017.sales.salesorderheader
```

Wynik



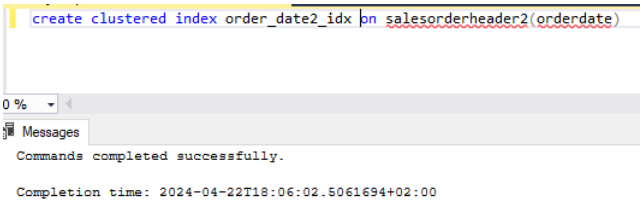
Wypisz tysiąc pierwszych zamówień:

```
select top 1000 * from salesorderheader2
order by orderdate
```

Stwórz indeks klastrujący według OrderDate:

```
create clustered index order_date2_idx on salesorderheader2(orderdate)
```

Wynik

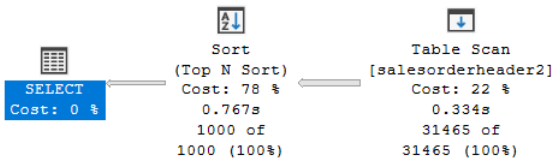


Wypisz ponownie tysiąc pierwszych zamówień. Co się zmieniło?

Wyniki dla 1000 pierwszych zamówień przed stworzeniem indeksu klastrującego:

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumb
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	SO43659	PO522145787
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	SO43660	PO18850127500
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	SO43661	PO18473189620
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	SO43662	PO18444174044
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	SO43663	PO18009186470
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	SO43664	PO16617121983
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	SO43665	PO16588191572
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	SO43666	PO16008173883
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	SO43667	PO15428132599
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	SO43668	PO14732180295
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	SO43669	PO14123169936
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	SO43670	PO14384116310
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	SO43671	PO13978119376
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	SO43672	PO13862153537
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	SO43673	PO13775141242
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	SO43674	PO12760141756
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	SO43675	PO12412186464

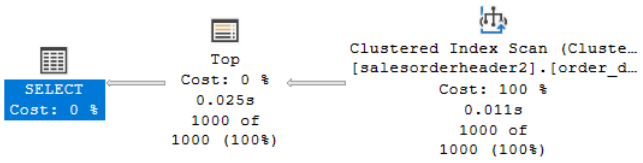
Execution Plan



Wyniki dla 1000 pierwszych zamówień po stworzeniu indeksu klastrującego:

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	Custom
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	SO43659	PO522145787	10-4020-000676	29825
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	SO43660	PO18850127500	10-4020-000117	29672
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	SO43661	PO18473189620	10-4020-000442	29734
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	SO43662	PO18444174044	10-4020-000227	29994
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	SO43663	PO18009186470	10-4020-000510	29565
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	SO43664	PO16617121983	10-4020-000397	29898
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	SO43665	PO16588191572	10-4020-000146	29580
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	SO43666	PO16008173883	10-4020-000511	30052
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	SO43667	PO15428132599	10-4020-000646	29974
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	SO43668	PO14732180295	10-4020-000514	29614
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	SO43669	PO14123169936	10-4020-000578	29747
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	SO43670	PO14384116310	10-4020-000504	29566
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	SO43671	PO13978119376	10-4020-000200	29890
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	SO43672	PO13862153537	10-4020-000119	30067
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	SO43673	PO13775141242	10-4020-000618	29844
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	SO43674	PO12760141756	10-4020-000083	29596

Execution Plan



> Możemy zauważyć, że wypisane zamówenia wyglądają tak samo, co nie jest zdziwieniem, ponieważ indeks klastrujący nie zmienia kolejności wierszy w tabeli, a jedynie sposób ich przechowywania. Zmienia się jednak plan wykonania zapytania:

> Użycie indeksu w drugim zapytaniu znacząco ogranicza czas ze względu na usunięcie sortowania. Przy tworzeniu indeksu klastrującego na kolumnę, pozbywamy się konieczności sortowania, czyli najbardziej kosztownej operacji dzięki czemu koszt zapytania jest znacznie mniejszy. Można z tego wywnioskować, że domyślnie tworzony jest indeks posortowany. W przeciwieństwie, w pierwszym przypadku, gdzie nie mamy jeszcze indeksu jest realizowana kosztowna operacja sortowania, która stanowi znaczą jego część.

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?

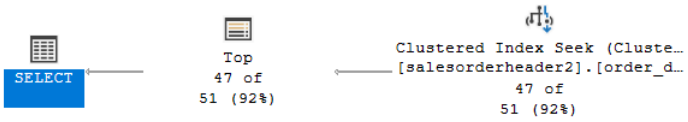
> Sprawdziliśmy 3 zapytania:

```
--query 3
select top 1000 * from salesorderheader2
where orderdate between '2010-10-01' and '2011-06-01'

--query 4
select top 1000 * from salesorderheader2
where orderdate between '2010-10-01' and '2011-06-01'
order by orderdate asc

--query 5
select top 1000 * from salesorderheader2
where orderdate between '2010-10-01' and '2011-06-01'
order by orderdate desc
```

Wyniki dla zapytania 3

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber
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	SO43659	PO522145787
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	SO43660	PO18850127500
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	SO43661	PO18473189620
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	SO43662	PO18444174044
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	SO43663	PO18009186470
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	SO43664	PO16617121983
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	SO43665	PO16588191572
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	SO43666	PO16008173883
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	SO43667	PO15428132599
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	SO43668	PO14732180295
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	SO43669	PO14123169936
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	SO43670	PO14384116310
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	SO43671	PO13978119376
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	SO43672	PO13862153537
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	SO43673	PO13775141242
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	SO43674	PO12760141756
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	SO43675	PO12412186464

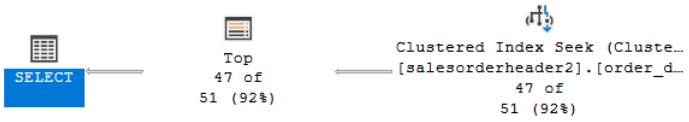
Execution Plan dla zapytania 3



Wyniki dla zapytania 4

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerI
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	SO43659	PO522145787	10-4020-000676	29825
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	SO43660	PO18850127500	10-4020-000117	29672
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	SO43661	PO18473189620	10-4020-000442	29734
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	SO43662	PO18444174044	10-4020-000227	29994
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	SO43663	PO18009186470	10-4020-000510	29565
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	SO43664	PO16617121983	10-4020-000397	29898
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	SO43665	PO16588191572	10-4020-000146	29580
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	SO43666	PO16008173883	10-4020-000511	30052
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	SO43667	PO15428132599	10-4020-000646	29974
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	SO43668	PO14732180295	10-4020-000514	29614
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	SO43669	PO14123169936	10-4020-000578	29747
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	SO43670	PO14384116310	10-4020-000504	29566
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	SO43671	PO13978119376	10-4020-000200	29890
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	SO43672	PO13862153537	10-4020-000119	30067

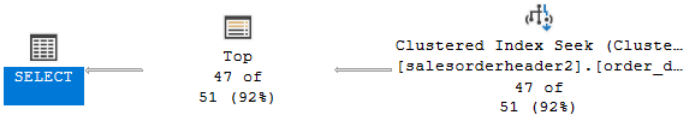
Execution Plan dla zapytania 4



Wyniki dla zapytania 5

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	C
1	43705	8	2011-06-01 00:00:00.000	2011-06-13 00:00:00.000	2011-06-08 00:00:00.000	5	1	SO43705	NULL	10-4030-011011	1
2	43704	8	2011-06-01 00:00:00.000	2011-06-13 00:00:00.000	2011-06-08 00:00:00.000	5	1	SO43704	NULL	10-4030-011005	1
3	43703	8	2011-06-01 00:00:00.000	2011-06-13 00:00:00.000	2011-06-08 00:00:00.000	5	1	SO43703	NULL	10-4030-016624	1
4	43702	8	2011-06-01 00:00:00.000	2011-06-13 00:00:00.000	2011-06-08 00:00:00.000	5	1	SO43702	NULL	10-4030-027645	2
5	43701	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	1	SO43701	NULL	10-4030-011003	1
6	43700	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	1	SO43700	NULL	10-4030-014501	1
7	43699	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	1	SO43699	NULL	10-4030-025863	2
8	43698	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	1	SO43698	NULL	10-4030-028389	2
9	43697	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	1	SO43697	NULL	10-4030-021768	2
10	43696	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43696	PO9947131800	10-4020-000603	2
11	43695	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43695	PO10179176559	10-4020-000027	2
12	43694	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43694	PO9657130250	10-4020-000315	2
13	43693	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43693	PO8120182325	10-4020-000485	2
14	43692	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43692	PO7859187017	10-4020-000221	2
15	43691	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43691	PO6409111675	10-4020-000292	2
16	43690	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43690	PO6235146326	10-4020-000431	2
17	43689	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43689	PO5626159507	10-4020-000166	2

Execution Plan dla zapytania 5



> Wyniki dwóch pierwszych zapytań wyglądają tak samo. Przy wykonywaniu zapytania pierwszego, - bez spacyfikowania kolejności, - dostajemy taki sam wynik jak przy kolejności rosnącej. Oznacza to, że właśnie w takiej kolejności posortowany jest indeks.

> Możemy zauważyć, że w drugim zapytaniu, indeks został wykorzystany, na co wskazuje operacja `Clustered Index Seek` w *Execution Plan dla zapytania 4*, jednak żadne dodatkowe sortowanie nie zostało wykonane. Jeśli chodzi o trzeci zapytanie, to również widzimy, że indeks został wykorzystany. Wskazuje na to operacja `Clustered Index Seek` w *Execution Plan dla zapytania 5*. Nie jest wykonywane dodatkowe sortowanie. Plany wykonania wszystkich trzech zapytań wyglądają tak samo.

> Wniosek: Oba indeksy działają poprawnie, dzięki czemy w obu przypadkach unikamy konieczności sortowania.

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
 )
```

Załóż 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 obecny indeks:

```
select productid, sum(unitprice), avg(unitprice), sum(orderqty), avg(orderqty)
from saleshistory
group by productid
order by productid
```

Załóż indeks typu ColumnStore:

```
create nonclustered columnstore index saleshistory_columnstore
 on saleshistory(unitprice, orderqty, productid)
```

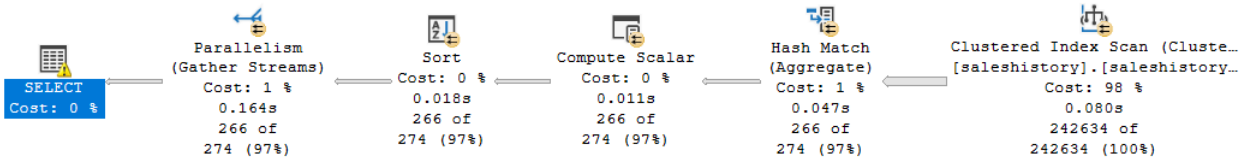
Sprawdź różnicę pomiędzy przetwarzaniem w zależności od indeksów. Porównaj plany i opisz różnicę.

Zapytanie z indeksem klastrowym

Wynik

	productid	(No column name)	(No column name)	(No column name)	(No column name)
1	707	190446.3596	30.8865	12532	2
2	708	183123.462	30.4495	13064	2
3	709	2126.67	5.656	2214	5
4	710	501.60	5.70	180	2
5	711	187655.7132	30.365	13486	2
6	712	51867.9142	7.6682	16622	2
7	713	42891.42	49.99	858	1
8	714	89598.859	36.7811	7272	2
9	715	114126.3424	34.901	13184	4
10	716	79979.0912	37.165	5960	2
11	717	354922.0452	814.0413	970	2
12	718	356216.839	813.2804	972	2
13	719	72303.7644	821.6336	218	2
14	722	147671.6584	188.3567	1880	2
15	723	20075.3356	193.032	258	2
16	725	145504.9176	194.5252	1982	2

Execution Plan



Estimated Subtree Cost

SELECT
Estimated operator progress: 100%

Actual Number of Rows for All Executions	266
Cached plan size	88 KB
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	6,18007
Estimated Number of Rows Per Execution	271
Estimated Number of Rows for All Executions	0

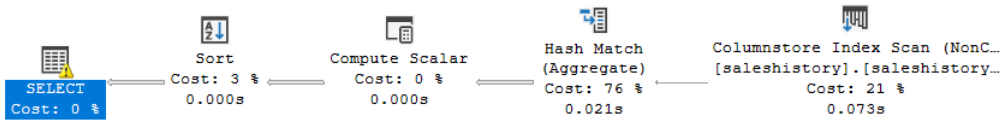
Statement
select productid, sum(unitprice), avg(unitprice), sum
(orderqty), avg(orderqty)
from saleshistory
group by productid
order by productid

Zapytanie z indeksem ColumnStore

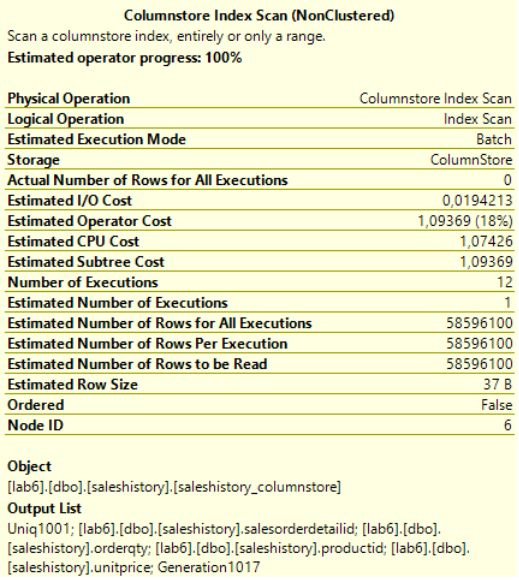
Wynik

	productid	(No column name)	(No column name)	(No column name)	(No column name)
1	707	190446.3596	30.8865	12532	2
2	708	183123.462	30.4495	13064	2
3	709	2126.67	5.656	2214	5
4	710	501.60	5.70	180	2
5	711	187655.7132	30.365	13486	2
6	712	51867.9142	7.6682	16622	2
7	713	42891.42	49.99	858	1
8	714	89598.859	36.7811	7272	2
9	715	114126.3424	34.901	13184	4
10	716	79979.0912	37.165	5960	2
11	717	354922.0452	814.0413	970	2
12	718	356216.839	813.2804	972	2
13	719	72303.7644	821.6336	218	2
14	722	147671.6584	188.3567	1880	2
15	723	20075.3356	193.032	258	2
16	725	145504.9176	194.5252	1982	2

Execution Plan



Estimated Subtree Cost



Columnstore Index Scan (NonClustered)

Scan a columnstore index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Columnstore Index Scan
Logical Operation	Index Scan
Estimated Execution Mode	Batch
Storage	ColumnStore
Actual Number of Rows for All Executions	0
Estimated I/O Cost	0,0194213
Estimated Operator Cost	1,09369 (18%)
Estimated CPU Cost	1,07426
Estimated Subtree Cost	1,09369
Number of Executions	12
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	58596100
Estimated Number of Rows Per Execution	58596100
Estimated Number of Rows to be Read	58596100
Estimated Row Size	37 B
Ordered	False
Node ID	6

Object
[lab6].[dbo].[saleshistory].[saleshistory_columnstore]
Output List
Uniq1001; [lab6].[dbo].[saleshistory].salesorderdetailid; [lab6].[dbo].[saleshistory].orderqty; [lab6].[dbo].[saleshistory].productid; [lab6].[dbo].[saleshistory].unitprice; Generation1017

> Indeks typu ColumnStore w przeciwieństwie do zwykłych indeksów ustawia kolejność pamięci kolumnami, a nie wierszami. Przez to, zapytanie, które bierze pod uwagę całość kolumny w celu pogrupowania lub obliczenia jej średnich wartości może wykonać się szybciej i prościej - bez dodania równoległości. Możemy jeszcze zauważyć, że Estimated Subtree Cost dla zwykłego indeksu to 6,18007 a Columnstore to 1,09369 co jest prawie 6 razy mniejsze.

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 ekranow)
- Komentarze do zapytań, ich wyników
- Sprawdzenie, co proponuje Database Engine Tuning Advisor (porównanie czy udało się Państwu znaleźć odpowiednie indeksy do zapytania)

> Wyniki:

Klastrowane (np. dla atrybutu nie będącego kluczem głównym)

> Pierwszy eksperyment, polega na stworzeniu indeksu klastrowego dla atrybutu, który nie jest kluczem głównym tabeli. W przypadku, gdy atrybut nie jest kluczem głównym, indeks klastrowy jest tworzony na podstawie klucza głównego. W przypadku, gdy atrybut jest kluczem głównym, indeks klastrowy jest tworzony na podstawie tego atrybutu.

Schemat tabeli

Tworzymy tabelę `Orders` z następującymi kolumnami:

- **OrderID**
- **CustomerID**
- **OrderDate**
- **OrderPrice**

Table zawierać będzie id zamówniea, id klienta, datę zamówienia oraz cenę zamówienia.

```
CREATE TABLE Orders (
  OrderID INT NOT NULL PRIMARY KEY,
  CustomerID INT NOT NULL,
  OrderDate DATE NOT NULL,
  OrderPrice DECIMAL(10, 2) NOT NULL
)
```

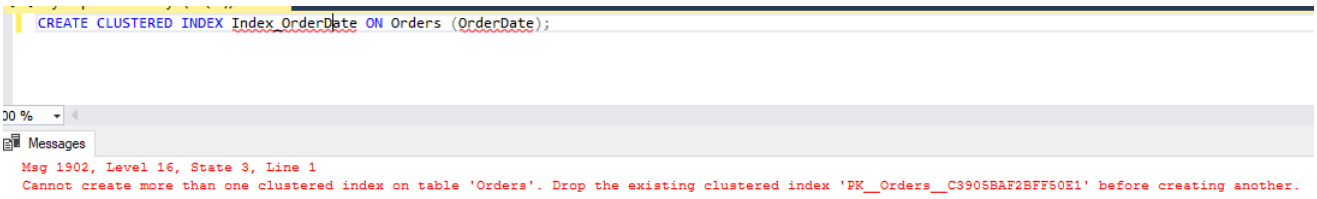
Wynik

Wypełniamy tabelę przykładowymi danymi z 100 000 rekordami:

```
DECLARE @count INT = 0

WHILE @count < 100000
BEGIN
  INSERT INTO Orders (OrderID, CustomerID, OrderDate, OrderPrice) VALUES (
    @count,
    @count % 100,
    DATEADD(DAY, @count % 365, '2020-01-01'),
    ROUND(RAND() * 100, 2)
  )
  SET @count = @count + 1
END
```

Dodajemy indeks klastrowany dla kolumny `OrderDate`, która nie jest kluczem głównym tabeli `Orders`:

```
CREATE CLUSTERED INDEX Index_OrderDate ON Orders (OrderDate);
```



> Otrzymujemy jednak błąd! Wynika to z faktu, że klucz główny jest już indeksem klastrowym. W takim przypadku, aby móc stworzyć indeks klastrowy dla innej kolumny, musimy usunąć indeks klastrowy dla klucza głównego. W naszym przypadku kluczem głównym jest `OrderID` i usuniemy go w taki sposób:

```
ALTER TABLE Orders
DROP CONSTRAINT PK__Orders__C3905BAF2BFF50E1;
```

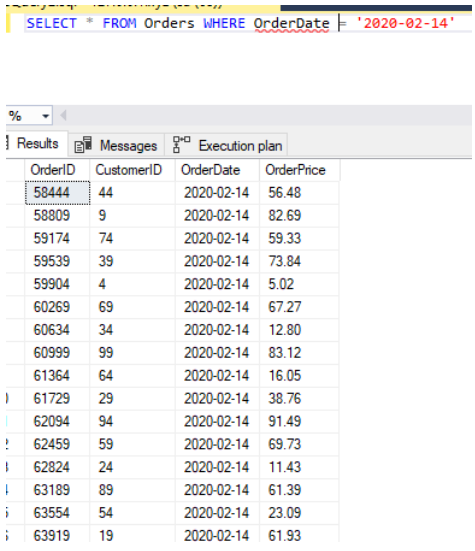
Teraz możemy dodać nasz indeks.

Przykładowe zapytania

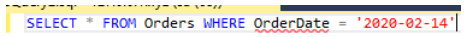
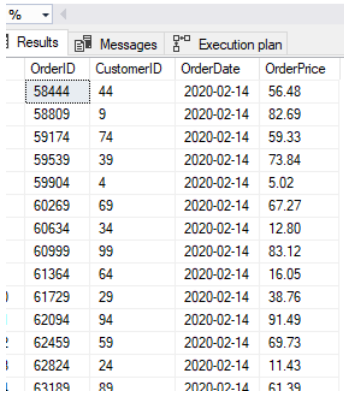
> Wykonujemy teraz przykładowe zapytanie. Zwróci nam on wszystkie dane dla zamówień z danego dnia. Zostanie wykonane porównanie działania zapytania z indeksem i bez niego. Przedstawione zostaną wyniki, plany wykonania oraz napisane zostaną wnioski.

```
SELECT * FROM Orders WHERE OrderDate = '2020-02-14'
```

Wyniki

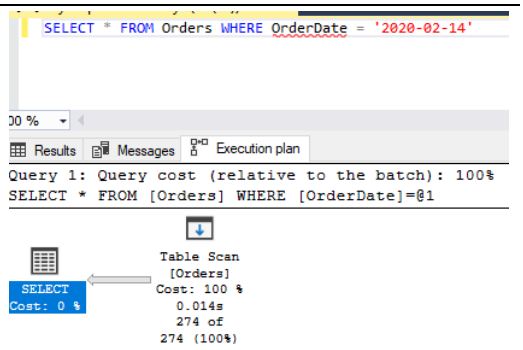
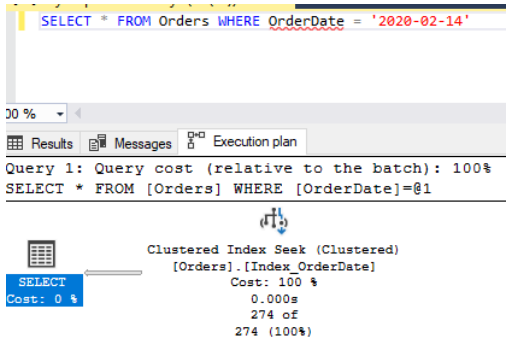
Zapytanie	Wynik
1	

OrderID	CustomerID	OrderDate	OrderPrice
58444	44	2020-02-14	56.48
58809	9	2020-02-14	82.69
59174	74	2020-02-14	59.33
59539	39	2020-02-14	73.84
59904	4	2020-02-14	5.02
60269	69	2020-02-14	67.27
60634	34	2020-02-14	12.80
60999	99	2020-02-14	83.12
61364	64	2020-02-14	16.05
61729	29	2020-02-14	38.76
62094	94	2020-02-14	91.49
62459	59	2020-02-14	69.73
62824	24	2020-02-14	11.43
63189	89	2020-02-14	61.39
63554	54	2020-02-14	23.09
63919	19	2020-02-14	61.93

Zapytanie	Wynik
	
2	

OrderID	CustomerID	OrderDate	OrderPrice
58444	44	2020-02-14	56.48
58809	9	2020-02-14	82.69
59174	74	2020-02-14	59.33
59539	39	2020-02-14	73.84
59904	4	2020-02-14	5.02
60269	69	2020-02-14	67.27
60634	34	2020-02-14	12.80
60999	99	2020-02-14	83.12
61364	64	2020-02-14	16.05
61729	29	2020-02-14	38.76
62094	94	2020-02-14	91.49
62459	59	2020-02-14	69.73
62824	24	2020-02-14	11.43
63189	89	2020-02-14	61.39

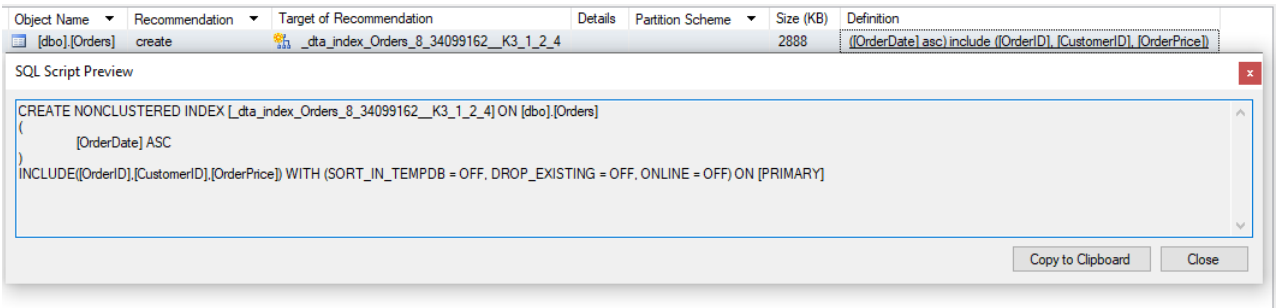
Plany wykonania

Zapytanie	Plan wykonania
1	
2	

Komantarz

> Możemy zauważyć, że w przypadku zapytania, które korzysta z indeksu klastrowanego, czas wykonania zapytania jest znacznie krótszy. W przypadku zapytania, które nie korzysta z indeksu klastrowanego, czas wykonania zapytania jest dłuższy, ponieważ wykonuje `Table Scan`. Zapytanie z indeksem wykonuje `Clustered Index Seek`. Dzięki indeksowi, koszt zapytania jest znacznie niższy, a liczba operacji wejścia/wyjścia jest mniejsza.

Rekomendacja Databse Engine Tuning Advisor



> Narzędzie sugeruje utworzenie indeksu nieklastrowego na OrderDate. W naszym przypadku usunęliśmy indeks klastrowy dla klucza głównego i zastąpiliśmy go nowym indeksem klastrowym dla OrderDate.

> Wniosek: Indeks klastrowany jest przydatny, gdy zapytania często odwołują się do kolumny, która jest indeksem klastrowym. W przypadku, gdy kolumna nie jest kluczem głównym, musimy usunąć indeks klastrowy dla klucza głównego, aby móc stworzyć indeks klastrowy dla innej kolumny.

Nieklastrowane

> Drugi eksperyment polega na użyciu indeksu nieklastrowego. Indeks nieklastrowy jest indeksem, który nie zmienia kolejności wierszy w tabeli. Indeks nieklastrowy jest tworzony na kolumnach, które nie są kluczami głównymi tabeli.

Schemat tabeli

Tworzymy tabelę `Products` z następującymi kolumnami:

- **ProductID**
- **ProductName**
- **CategoryID**
- **ProductPrice**

Tabela ta zawierać będzie id produktu, nazwę produktu, id kategori oraz cenę produktu.

```
CREATE TABLE Products (
  ProductID INT NOT NULL PRIMARY KEY,
  ProductName NVARCHAR(50) NOT NULL,
  CategoryID INT NOT NULL,
  ProductPrice DECIMAL(10, 2) NOT NULL
)
```

Wypełniamy tabelę przykładowymi danymi z 100 000 rekordami:

```
DECLARE @count INT = 0

WHILE @count < 100000
BEGIN
  INSERT INTO Products (ProductID, ProductName, CategoryID, ProductPrice) VALUES (
    @count,
    CONCAT('Product', @count),
    @count % 10,
    ROUND(RAND() * 100, 2)
  )
  SET @count = @count + 1
END
```

Wynik

```
select * from Products
```

%

Results Messages

ProductID	ProductName	CategoryID	ProductPrice
0	Product0	0	76.52
1	Product1	1	16.03
2	Product2	2	21.33
3	Product3	3	87.21
4	Product4	4	12.33
5	Product5	5	91.87
6	Product6	6	96.97
7	Product7	7	10.07
8	Product8	8	34.56
9	Product9	9	44.89
10	Product10	0	59.03
11	Product11	1	5.22
12	Product12	2	27.50
13	Product13	3	46.92
14	Product14	4	81.26

Dodajemy indeks nieklastrowany dla kolumny `CategoryID`:

```
CREATE NONCLUSTERED INDEX Index_CategoryID ON Products (CategoryID);
```

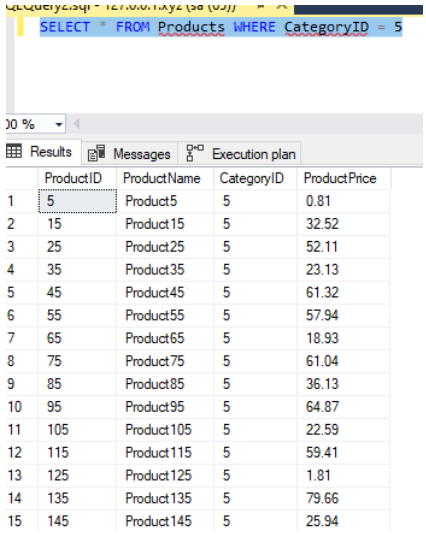
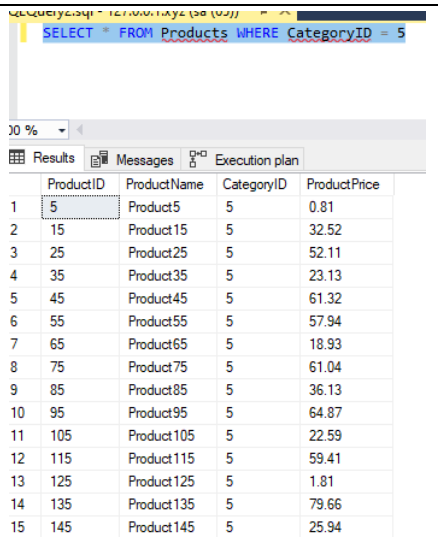
Przykładowe zapytania

> Wykonujemy teraz przykładowe zapytanie, które zwróci nam wszystkie dane dla kateogri równej 5.Zostanie wykonane porównanie działania zapytania z indeksem i bez niego. Przedstawione zostaną wyniki, plany wykonania oraz napisane zostaną wnioski.

```
SELECT * FROM Products WHERE CategoryID = 5
```

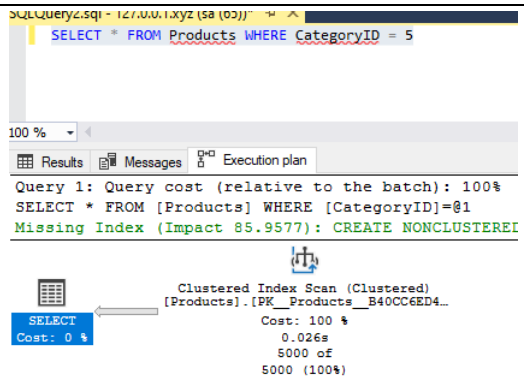
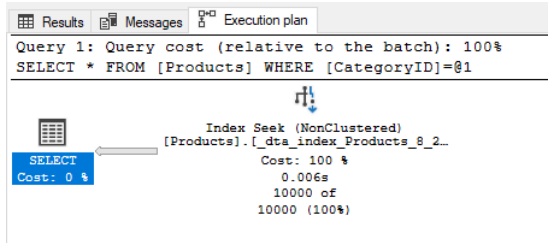
Wyniki

Zapytanie	Wynik

Zapytanie	Wynik
1	
2	

	ProductID	ProductName	CategoryID	ProductPrice
1	5	Product5	5	0.81
2	15	Product15	5	32.52
3	25	Product25	5	52.11
4	35	Product35	5	23.13
5	45	Product45	5	61.32
6	55	Product55	5	57.94
7	65	Product65	5	18.93
8	75	Product75	5	61.04
9	85	Product85	5	36.13
10	95	Product95	5	64.87
11	105	Product105	5	22.59
12	115	Product115	5	59.41
13	125	Product125	5	1.81
14	135	Product135	5	79.66
15	145	Product145	5	25.94

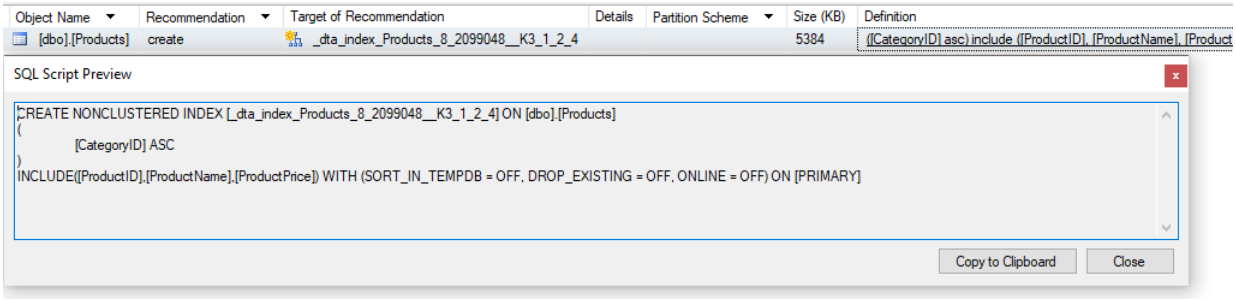
Plany wykonania

Zapytanie	Plan wykonania
1	
2	

Komantarz

> Użycie indeksu nieklastrowego przyspiesza nam działanie zapytania. Możemy zauważyć, że pierwsze zapytanie, czyli zapytanie bez indeksu wykonuje się o około 4 razy dłużej niż drugie, które ten indeks wykorzystuje. Zapytanie z indeksem wykonuje `Index Seek` natomiast to bez indeksu scana całej tabeli, co można zauważyć na planie wykonania.

Rekomendacja Databse Engine Tuning Advisor



> Narzędzie sugeruje utworzenie nieklastrowego indeksu na CategoryID, czyli takiego jaki został przez nas wcześniej stworzony.

Indeksy wykorzystujące kilka atrybutów, indeksy include

> Trzeci ekperyment polega na użyciu indeksu wykorzystującego kilka atrybutów, czyli indeksu include. Polega on na dodaniu do indeksu dodatkowych kolumn, które nie są kluczami indeksu, ale są dodawane do indeksu, aby zwiększyć jego wydajność.

Schemat tabeli

Tworzymy tabelę `Employees` z następującymi kolumnami:

- **EmployeeID**
- **Name**
- **Surname**
- **City**
- **Salary**

Tabela zawierać będzie rekordy z danymi takimi jak: id pracownika, imie oraz naziwsko pracownika, miasto zamieszkania i jego zarobki.

```
CREATE TABLE Employees (
  EmployeeID INT NOT NULL PRIMARY KEY,
  Name NVARCHAR(50) NOT NULL,
  Surname NVARCHAR(50) NOT NULL,
  City NVARCHAR(30) NOT NULL,
  Salary DECIMAL(10, 2) NOT NULL
)
```

Wypełniamy tabelę przykładowymi danymi z 100 000 rekordami:

```
DECLARE @count INT = 0

WHILE @count < 100000
BEGIN
  INSERT INTO Employees (EmployeeID, Name, Surname, City, Salary)
  VALUES (
    @count,
    CONCAT('Name', @count),
    CONCAT('Surname', @count),
    CONCAT('City', @count % 20),
    ROUND(RAND() * 10000, 2)
  )
  SET @count = @count + 1
END
```

Wynik

```
select * from Employees
```

Results | Messages

EmployeeID	Name	Surname	City	Salary
0	Name0	Surname0	City0	4867.27
1	Name1	Surname1	City1	6663.53
2	Name2	Surname2	City2	4436.37
3	Name3	Surname3	City3	7079.16
4	Name4	Surname4	City4	1323.85
5	Name5	Surname5	City5	632.46
6	Name6	Surname6	City6	5917.04
7	Name7	Surname7	City7	6304.95
8	Name8	Surname8	City8	5292.08
9	Name9	Surname9	City9	6246.03
10	Name10	Surname10	City10	9513.63
11	Name11	Surname11	City11	4007.02
12	Name12	Surname12	City12	4875.79
13	Name13	Surname13	City13	4185.46
14	Name14	Surname14	City14	2532.49
15	Name15	Surname15	City15	7372.56
16	Name16	Surname16	City16	8041.37

Dodajemy indeks nieklastrowany dla kolumny `City` oraz include dla `Name`, `Surname` i `Salary`:

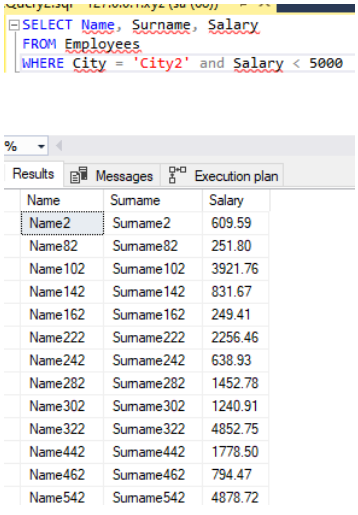
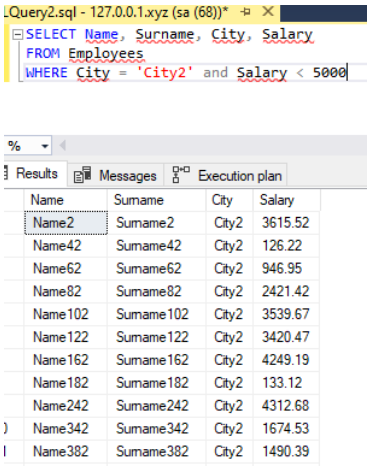
```
CREATE NONCLUSTERED INDEX Index_City_Salary
ON Employees (City) INCLUDE (Name, Surname, Salary);
```

Przykładowe zapytania

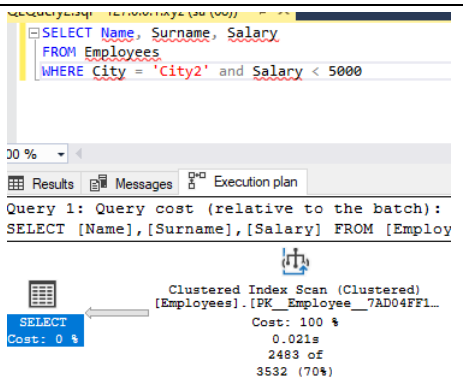
> Wykonujemy teraz przykładowe zapytanie. W zapytaniu tym chcemy otrzymać Imiona, Nazwiska, Miasto oraz zarobki osób z miasta City2 z zarobkami poniżej 5000. Sprawdzimy wyniki bez indeksu oraz dla indeksu. Przedstawione zostaną wyniki, plany wykonania oraz napisane zostaną wnioski.

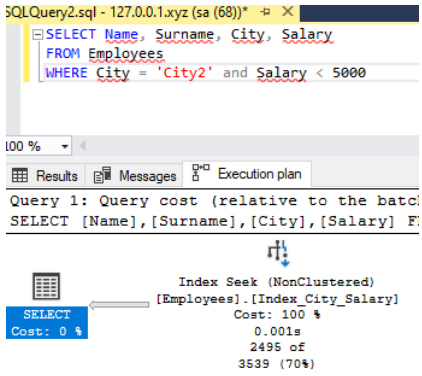
```
SELECT Name, Surname, Salary
FROM Employees
WHERE City = 'City2' and Salary < 5000
```

Wyniki

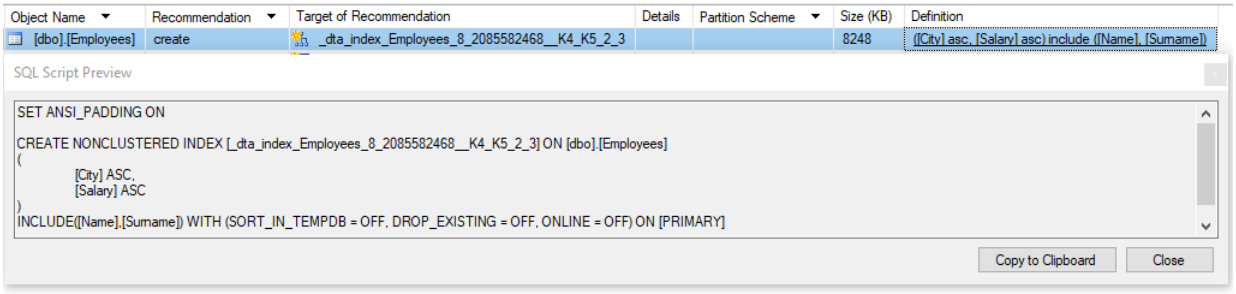
Zapytanie	Wynik
1	
2	

Plany wykonania

Zapytanie	Plan wykonania
1	

Zapytanie	Plan wykonania
2	

Rekomendacja Databse Engine Tuning Advisor



> Narzędzie sugeruje utworzeniu indeksu, który został przez nas już zdefiniowany, czyli indeks wykorzystujący kilka atrybutów.

> Wniosek: Indeksy wykorzystujące kilka atrybutów, wraz z include są przydatne, gdy zapytania obejmują wiele kolumn.

Komantarz

> W tym ekperymencie, możemy zauważyć, że w przypadku zapytania, które nie używa indeksu, wykonywane jest pełne skanowanie tabeli `Index Seek`. W przypadku zapytania, które korzysta z indeksu, wykonywany jest `Inde Scan`. Dzięki nim zapytania stają się bardziej wydajne, bo SZBD może w szybszy sposób uzsykać dostęp do danych, które spełniają kryterium z zapytania.

Filtered Index (Indeks warunkowy)

> Filtered Index, czyli Indeks warunkowy, którego używamy w 4 eksperymencie jest to indeks, który zawiera tylko wiersze, które spełniają określone warunki. Indeks warunkowy jest tworzony na podstawie warunku, który jest określony w klauzuli WHERE.

Schemat tabeli

Tworzymy tabelę `Customer` z następującymi kolumnami:

- **CustomerID**
- **Name**
- **City**
- **Mail**

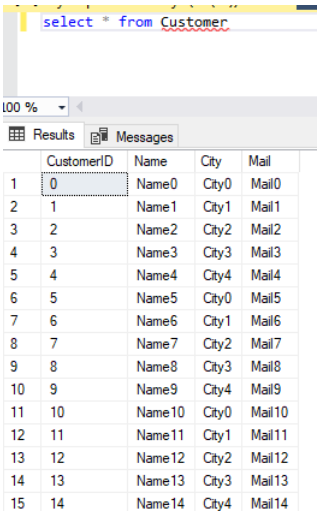
Tabela ta będzie przechowywać dane o klientach, ich imionach, miastach w których mieszkają oraz mailach.

```
CREATE TABLE Customer (
  CustomerID INT NOT NULL PRIMARY KEY,
  Name NVARCHAR(50) NOT NULL,
  City NVARCHAR(30) NOT NULL,
  Mail NVARCHAR(50) NOT NULL
)
```

Wypełniamy tabelę przykładowymi danymi z 100 000 rekordami:

```
DECLARE @count INT = 0

WHILE @count < 100000
BEGIN
  INSERT INTO Customer (CustomerID, Name, City, Mail) VALUES (
    @count,
    CONCAT('Name', @count),
    CONCAT('City', @count % 5),
    CONCAT('Mail', @count)
  )
  SET @count = @count + 1
END
```

Wynik



	CustomerID	Name	City	Mail
1	0	Name0	City0	Mail0
2	1	Name1	City1	Mail1
3	2	Name2	City2	Mail2
4	3	Name3	City3	Mail3
5	4	Name4	City4	Mail4
6	5	Name5	City0	Mail5
7	6	Name6	City1	Mail6
8	7	Name7	City2	Mail7
9	8	Name8	City3	Mail8
10	9	Name9	City4	Mail9
11	10	Name10	City0	Mail10
12	11	Name11	City1	Mail11
13	12	Name12	City2	Mail12
14	13	Name13	City3	Mail13
15	14	Name14	City4	Mail14

Dodajemy indeks warunkowy (filtered index) dla kolumny `City`, dla określonego miasta np. 'City3':

```
CREATE NONCLUSTERED INDEX Index_City
ON Customer (City, CustomerID, Name)
INCLUDE (Mail)
WHERE City = 'City3';
```

Przykładowe zapytania

> Wykonujemy teraz przykładowe zapytania. Pierwsze, które korzystać będzie z indeksu warunkowego zwróci nam wszystkie dane o Customerach z miasta `City3`. Drugie, które nie będzie korzystać z indeksu warunkowego, zwróci nam wszystkie dane o Customerach z miasta `City1`. Zostanie wykonane porównanie działania zapytania z indeksem i bez niego. Przedstawione zostaną wyniki, plany wykonania oraz napisane zostaną wnioski.

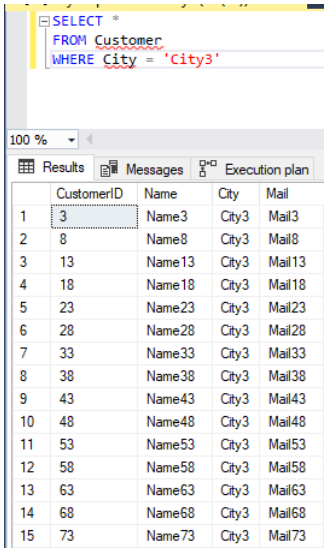
```
SELECT *
FROM Customer
WHERE City = 'City3'
```

```
SELECT *
FROM Customer
WHERE City = 'City1'
```

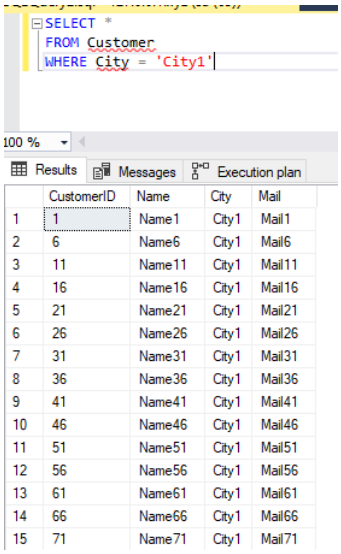
Wynik

Zapytanie **Wynik**

1



	CustomerID	Name	City	Mail
1	3	Name3	City3	Mail3
2	8	Name8	City3	Mail8
3	13	Name13	City3	Mail13
4	18	Name18	City3	Mail18
5	23	Name23	City3	Mail23
6	28	Name28	City3	Mail28
7	33	Name33	City3	Mail33
8	38	Name38	City3	Mail38
9	43	Name43	City3	Mail43
10	48	Name48	City3	Mail48
11	53	Name53	City3	Mail53
12	58	Name58	City3	Mail58
13	63	Name63	City3	Mail63
14	68	Name68	City3	Mail68
15	73	Name73	City3	Mail73

Zapytanie	Wynik
2	

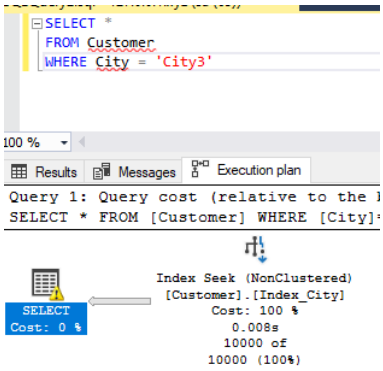
```
SELECT *
FROM Customer
WHERE City = 'City1'
```

100 %

Results | Messages | Execution plan

	CustomerID	Name	City	Mail
1	1	Name1	City1	Mail1
2	6	Name6	City1	Mail6
3	11	Name11	City1	Mail11
4	16	Name16	City1	Mail16
5	21	Name21	City1	Mail21
6	26	Name26	City1	Mail26
7	31	Name31	City1	Mail31
8	36	Name36	City1	Mail36
9	41	Name41	City1	Mail41
10	46	Name46	City1	Mail46
11	51	Name51	City1	Mail51
12	56	Name56	City1	Mail56
13	61	Name61	City1	Mail61
14	66	Name66	City1	Mail66
15	71	Name71	City1	Mail71

Plany wykonania

Zapytanie	Plan wykonania
1	

```
SELECT *
FROM Customer
WHERE City = 'City3'
```

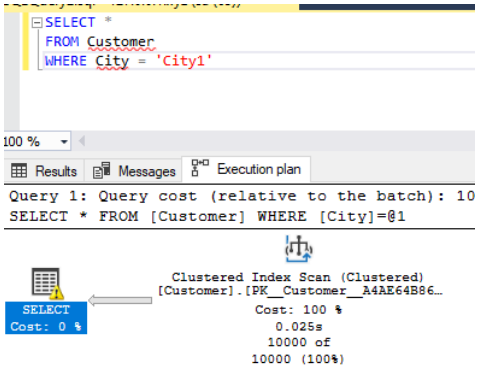
100 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the
SELECT * FROM [Customer] WHERE [City]
```

```
Index Seek (NonClustered)
[Customer].[Index_City]
Cost: 100 %
0.008s
10000 of
10000 (100%)
```

```
SELECT
Cost: 0 %
```

2	

```
SELECT *
FROM Customer
WHERE City = 'City1'
```

100 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 10
SELECT * FROM [Customer] WHERE [City]=@1
```

```
Clustered Index Scan (Clustered)
[Customer].[PK__Customer__A4AE64B86…
Cost: 100 %
0.025s
10000 of
10000 (100%)
```

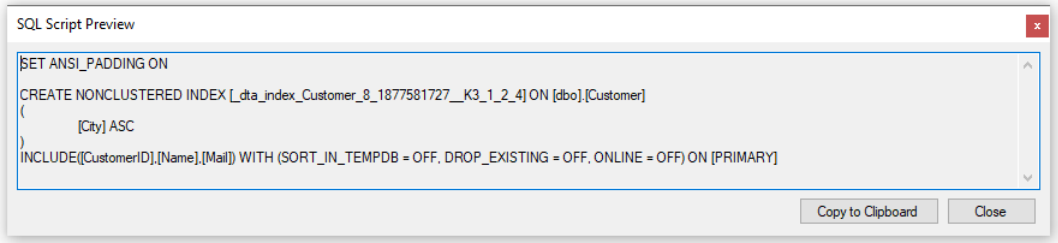
```
SELECT
Cost: 0 %
```

Komantarz

> W tym eksperymencie, możemy zauważyć, że tam gdzie jest użyty indeks warunkowy, zapytanie wykonuje się szybciej i to o wiele. Korzystamy tam z `Index Seek`. W przypadku zapytania, które korzysta z indeksu warunkowego, czas wykonania zapytania jest krótszy, ponieważ optymalizator korzysta z indeksu, który zawiera tylko interesujące nas wartości. W przypadku zapytania, które nie korzysta z indeksu warunkowego, czas wykonania zapytania jest dłuższy, ponieważ optymalizator musi przeszukać całą tabelę, wykonując scana tabeli, co można zaobserwować na planie wykonania zapytania, przedstawione jako `Clustered Index Scan`. Dzięki indeksowi, koszt zapytania jest znacznie niższy, a liczba operacji wejścia/wyjścia jest mniejsza.

Rekomendacja Databse Engine Tuning Advisor

Object Name	Recommendation	Target of Recommendation	Details	Partition Scheme	Size (KB)	Definition
[dbo].[Customer]	create	_dta_index_Customer_8_1877581727__K3_1_2_4			6656	([City] asc) include ([CustomerID], [Name], [Mail])



> Narzędzie sugeruje utworzenie identycznego indeksu, co wcześniej stworzony przez nas z dodatkową rekomendacją uporządkowania kategorii rosnąco.

> Wnioski: Indeks warunkowy jest skuteczny gdy zapytanie pokrywa się z jego warunkami. Kiedy warunki nie są spełnione, może dojść do kosztownego pełnego skanowania tabeli.

Kolumnowe

> Indeksy Kolumnowe to indeksy, które przechowują dane w kolumnach, a nie w wierszach. Są one tworzone na kolumnach, które są często używane w zapytaniach, które wykonują operacje agregujące, takie jak SUM, AVG, COUNT, MAX, MIN. Tych indeksów użyjemy w eksperymencie 5.

Tworzymy tabelę `Orders` z następującymi klumnami:

- **OrderID**
- **CustomerID**
- **ProductID**
- **Quantity**
- **OrderPrice**

Tabela ta będzie przechowywać dane o zamówwieniach, identyfikatorach klientów, produktach, ilości produktów oraz ich cenie.

```
CREATE TABLE Orders (
  OrderID INT NOT NULL PRIMARY KEY,
  CustomerID INT NOT NULL,
  ProductID INT NOT NULL,
  Quantity DECIMAL(10, 2) NOT NULL,
  OrderPrice DECIMAL(10, 2) NOT NULL
)
```

Wypełniamy tabelę przykładowymi danymi z 100 000 rekordami:

```
DECLARE @count INT = 0

WHILE @count < 100000
BEGIN
  INSERT INTO Orders (OrderID, CustomerID, ProductID, Quantity, OrderPrice) VALUES (
    @count,
    @count % 100,
    @count % 10,
    ROUND(RAND() * 20, 2),
    ROUND(RAND() * 100, 2)
  )
  SET @count = @count + 1
END
```

Wynik

```
select * from Orders
```

100 %

Results | Messages

	OrderID	CustomerID	ProductID	Quantity	OrderPrice
1	0	0	0	3.15	66.77
2	1	1	1	3.96	45.97
3	2	2	2	0.22	14.85
4	3	3	3	14.03	64.86
5	4	4	4	0.76	97.33
6	5	5	5	16.10	9.19
7	6	6	6	4.73	73.59
8	7	7	7	11.58	8.45
9	8	8	8	2.78	89.42
10	9	9	9	0.03	79.18
11	10	10	0	3.77	97.68
12	11	11	1	11.52	58.99
13	12	12	2	13.12	20.70
14	13	13	3	6.37	39.29
15	14	14	4	0.10	16.48

Tworzymy indeks kolumnowy na kolumnę `OrderPrice`:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX Index_OrderPrice
ON Orders (OrderPrice);
```

Przykładowe zapytania

> Wykonujemy teraz przykładowe zapytanie, które zwróci nam ProductID oraz TotalSale. Pierwsze zapytanie nie będzie korzystać z indeksu kolumnowego. Drugie zapytanie będzie korzystać z indeksu kolumnowego. Zostanie wykonane porównanie działania zapytania z indeksem i bez niego. Przedstawione zostaną wyniki, plany wykonania oraz napisane zostaną wnioski.

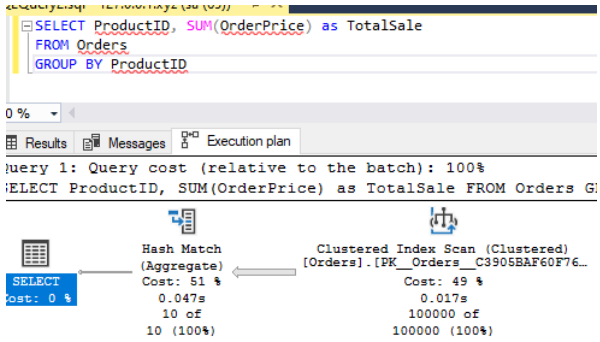
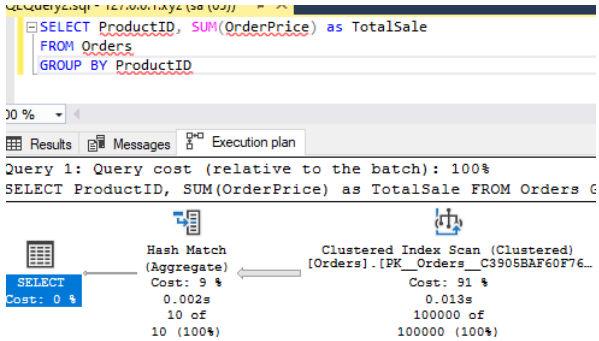
```
SELECT ProductID, SUM(OrderPrice) as TotalSale
FROM Orders
GROUP BY ProductID
```

Wyniki

Zapytanie	Wynik
1	`SELECT ProductID, SUM(OrderPrice) as TotalSale FROM Orders GROUP BY ProductID` Results | Messages | Execution plan ProductID / TotalSale: 0 504643.26; 9 503561.40; 3 501539.68; 6 499333.78; 7 501780.51; 1 497047.94; 4 500867.49; 5 499954.63; 2 502797.44; 8 498965.96
2	`SELECT ProductID, SUM(OrderPrice) as TotalSale FROM Orders GROUP BY ProductID` Results | Messages | Execution plan ProductID / TotalSale: 0 504643.26; 9 503561.40; 3 501539.68; 6 499333.78; 7 501780.51; 1 497047.94; 4 500867.49; 5 499954.63; 2 502797.44; 8 498965.96

Plany wykonania

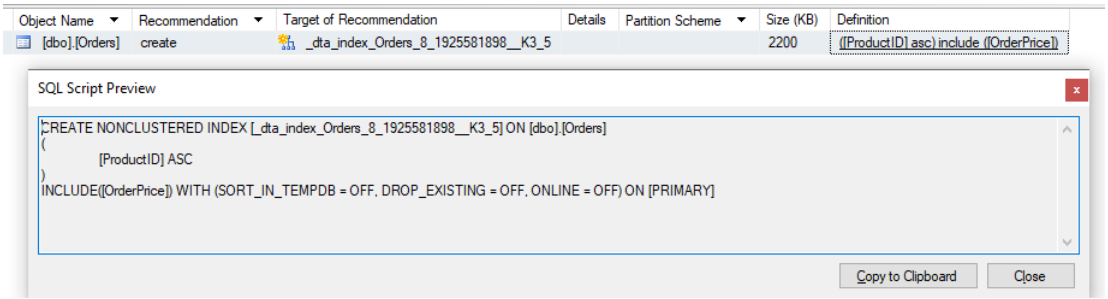
Zapytanie	Plan wykonania

Zapytanie	Plan wykonania
1	
2	

Komantarz

> W przypadku wykorzystania indeksu kolumnowego, zapytanie wykonuje się znacznie szybciej, ponieważ optymalizator korzysta z indeksu, który zawiera tylko interesujące nas wartości. W przypadku zapytania, które nie korzysta z indeksu kolumnowego, czas wykonania zapytania jest dłuższy, ponieważ optymalizator musi przeszukać całą tabelę, wykonując jej pełny skan, co można zaobserwować na planie wykonania zapytania. W przypadku zapytania, które korzysta z indeksu kolumnowego, 91% kosztów stanowiło `ClusteredIndex Scan`, bo `Hash Match` był już znany. W przypadku zapytania, które nie korzysta z indeksu kolumnowego, koszty się podzieliły prawie po 50 % pomiędzy `Hash Match` a `Clustered Index Scan`, co jest spowodowane nie znajomością `Hash Match` przez optymalizator.

Rekomendacja Databse Engine Tuning Advisor



> Narzędzie Database Engine Tuning Advisor sugeruje utworzenie indeksu, który ma kolumnę `OrderPrice` jako INCLUDE. Dzięki temu baza danych może efektywniej wykonywać operacje agregujące, takie jak suma wartości zamówień dla określonego przedziału czasowego, bo ma szybszy dostęp do tych danych.

> Wnioski: Indeks kolumnowy może znacznie usprawnić i poprawić wydajność zapytań. Doskonale nadaje się on do zapytań analitycznych, które wymagają szybkiego dostępu do dużej ilości danych i obliczeń agregujących.

Próbnie

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