Indeksy, optymalizator
Lab 6-7

Imię i nazwisko:

Mateusz Skowron, Bartłomiej Wiśniewski, Karol Wrona

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
```

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
```

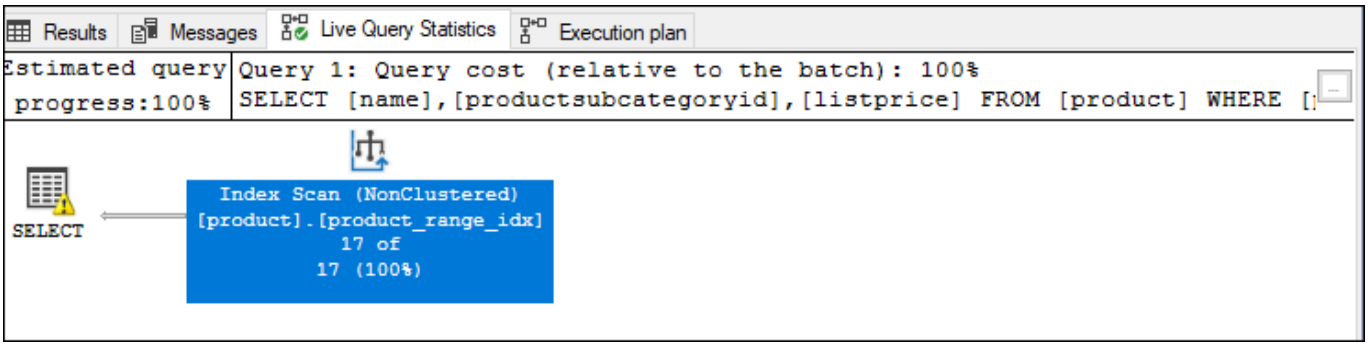
Sprawdź, czy indeks jest użyty w zapytaniu, który jest dopełnieniem zbioru:

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

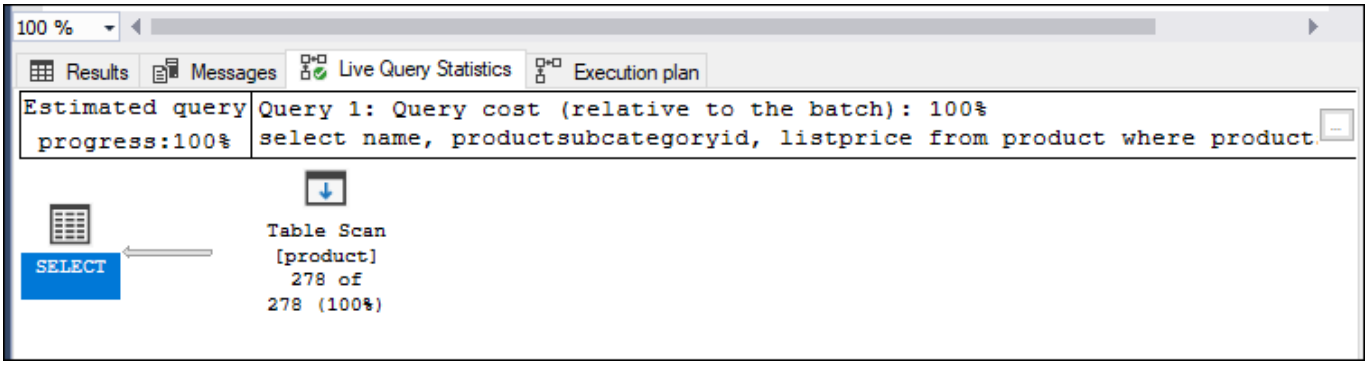
Skomentuj oba zapytania. Czy indeks został użyty w którymś zapytaniu, dlaczego? Czy indeks nie został użyty w którymś zapytaniu, dlaczego? Jak działają indeksy z warunkiem?

> Wyniki:

Execution Plan dla zapytania 1:



Execution Plan dla zapytania 2:



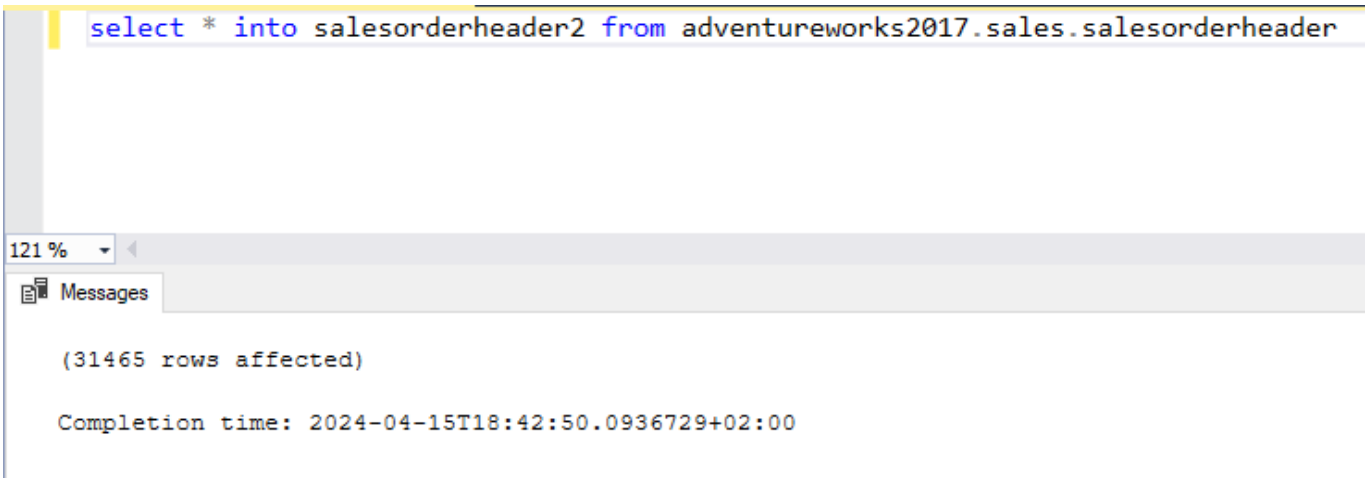
Tylko pierwsze zapytanie wykorzystuje indeks. W drugim zapytaniu warunek jest przeciwieństwem warunku indeksu, dlatego indeks nie jest używany. Indeks z warunkiem działa tylko wtedy, gdy warunek jest spełniony. W przeciwnym przypadku indeks nie jest używany.

Zadanie 2 – indeksy klastrujące

Celem zadania jest poznanie indeksów klastrujących![]
(file:////Users/rm/Library/Group%20Containers/UBF8T346G9.Office/TemporaryItems/msohtmlclip/clip_image001.jpg)

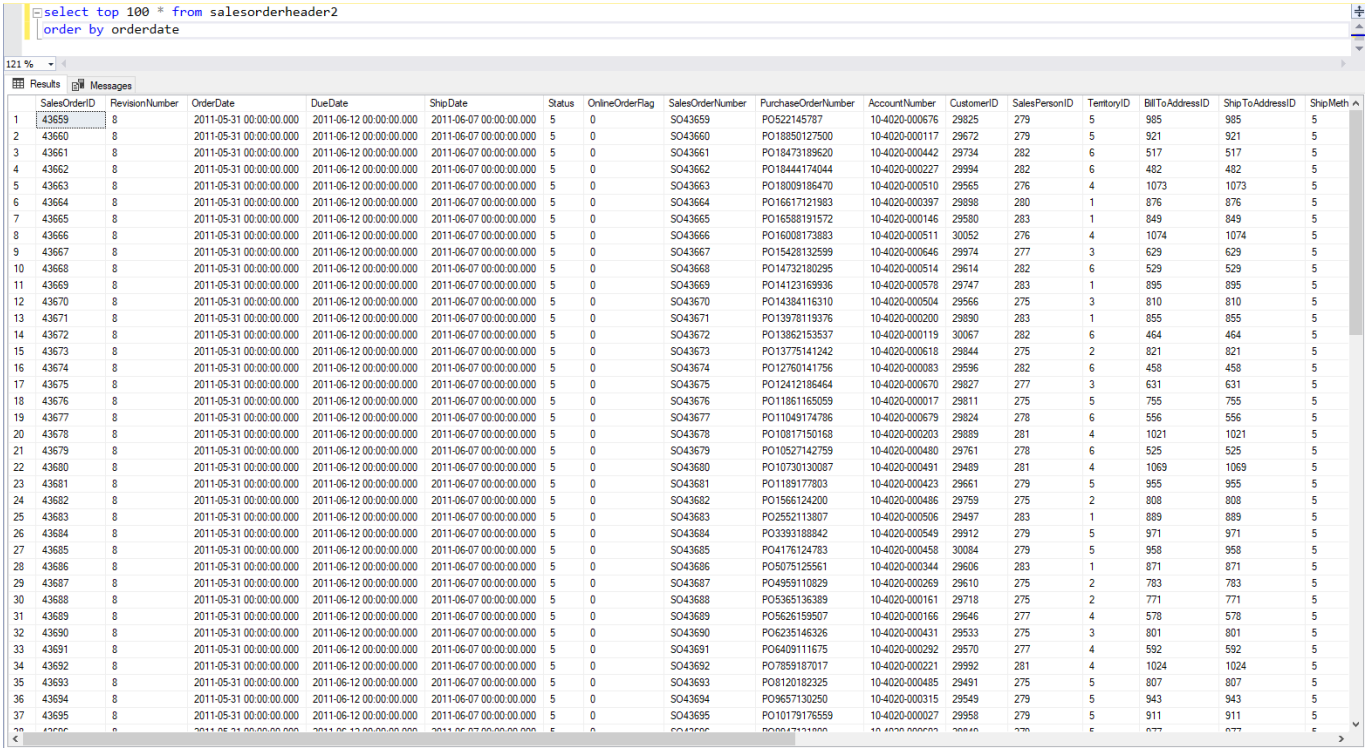
Skopiuj ponownie tabelę SalesOrderHeader do swojej bazy danych:

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



Wypisz sto pierwszych zamówień:

```
select top 100 * from salesorderheader2
order by orderdate
```

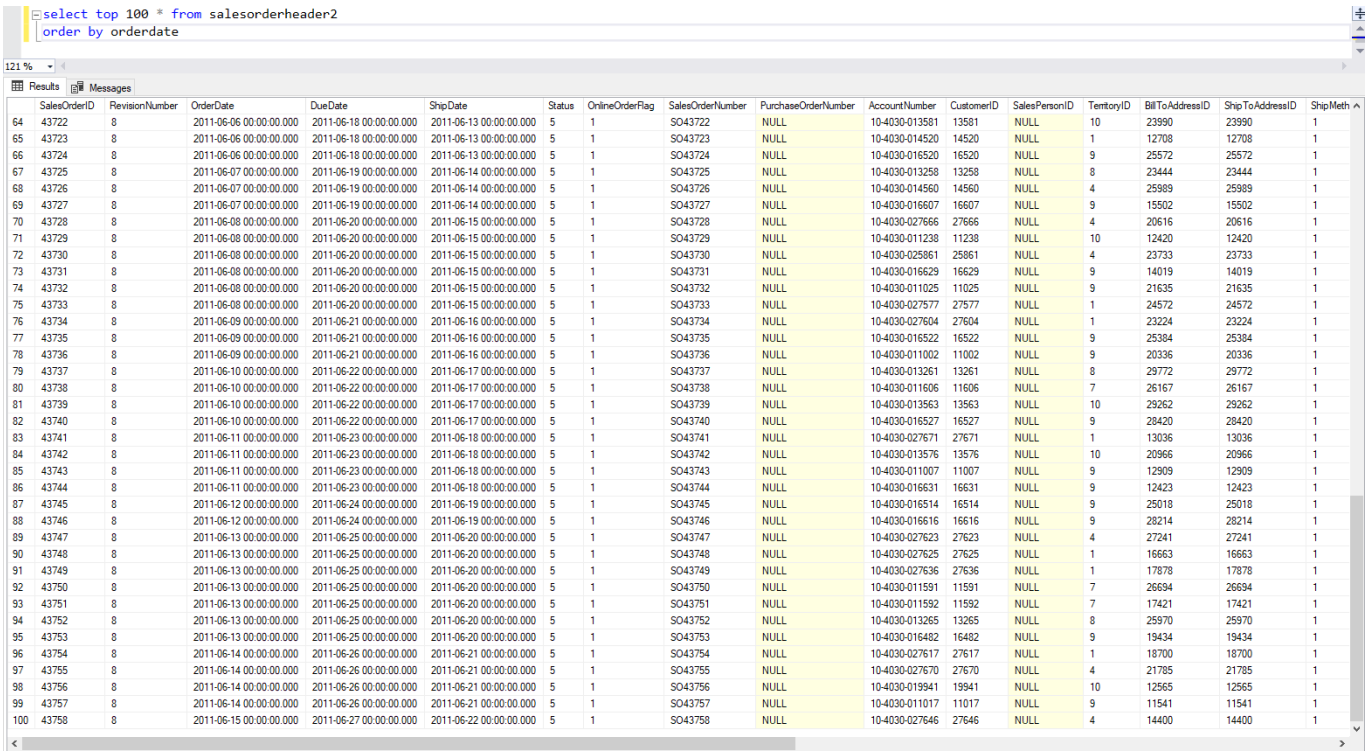


```
select top 100 * from salesorderheader2
order by orderdate
```

121 %

Results Messages

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerID	SalesPersonID	TerritoryID	BillToAddressID	ShipToAddressID	ShipMeth
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	279	5	985	985	5
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	279	5	921	921	5
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	282	6	517	517	5
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	282	6	482	482	5
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	276	4	1073	1073	5
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	280	1	876	876	5
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	283	1	849	849	5
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	276	4	1074	1074	5
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	277	3	629	629	5
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	282	6	529	529	5
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	283	1	895	895	5
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	275	3	810	810	5
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	283	1	855	855	5
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	282	6	464	464	5
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	275	2	821	821	5
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	282	6	458	458	5
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	10-4020-000670	29827	277	3	631	631	5
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	SO43676	PO11861165059	10-4020-000017	29811	275	5	755	755	5
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	SO43677	PO11049174786	10-4020-000679	29824	278	6	556	556	5
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	SO43678	PO10817150168	10-4020-000203	29889	281	4	1021	1021	5
21	43679	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43679	PO10527142759	10-4020-000480	29761	278	6	525	525	5
22	43680	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43680	PO10730130087	10-4020-000491	29489	281	4	1069	1069	5
23	43681	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43681	PO1189177803	10-4020-000423	29661	279	5	955	955	5
24	43682	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43682	PO1566124200	10-4020-000486	29759	275	2	808	808	5
25	43683	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43683	PO2552113807	10-4020-000506	29497	283	1	889	889	5
26	43684	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43684	PO3393188842	10-4020-000549	29912	279	5	971	971	5
27	43685	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43685	PO4176124783	10-4020-000458	30084	279	5	958	958	5
28	43686	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43686	PO5075125561	10-4020-000344	29606	283	1	871	871	5
29	43687	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43687	PO4959110829	10-4020-000269	29610	275	2	783	783	5
30	43688	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43688	PO5365136389	10-4020-000161	29718	275	2	771	771	5
31	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	29646	277	4	578	578	5
32	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	29533	275	3	801	801	5
33	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	29570	277	4	592	592	5
34	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	29992	281	4	1024	1024	5
35	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	29491	275	5	807	807	5
36	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	29549	279	5	943	943	5
37	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	29958	279	5	911	911	5

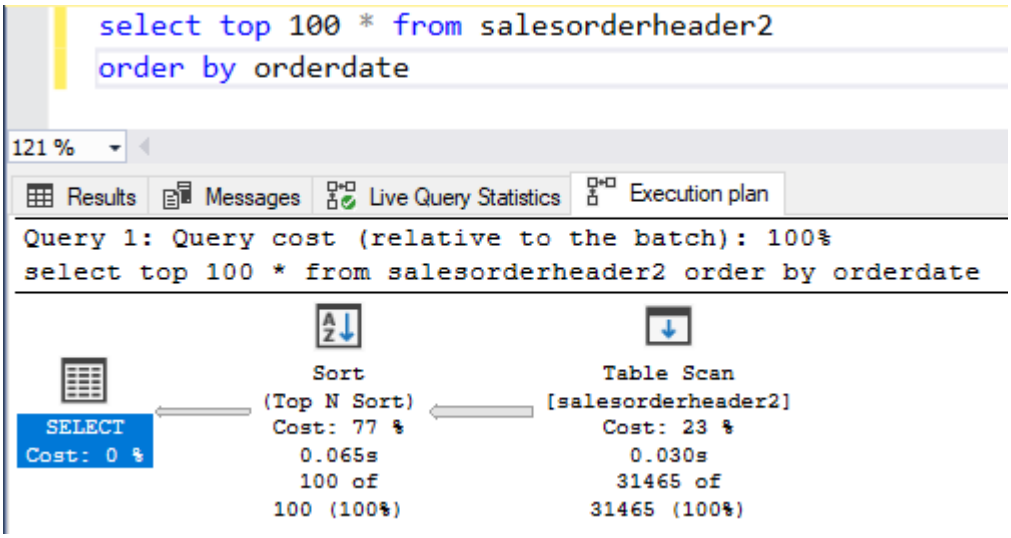


```
select top 100 * from salesorderheader2
order by orderdate
```

121 %

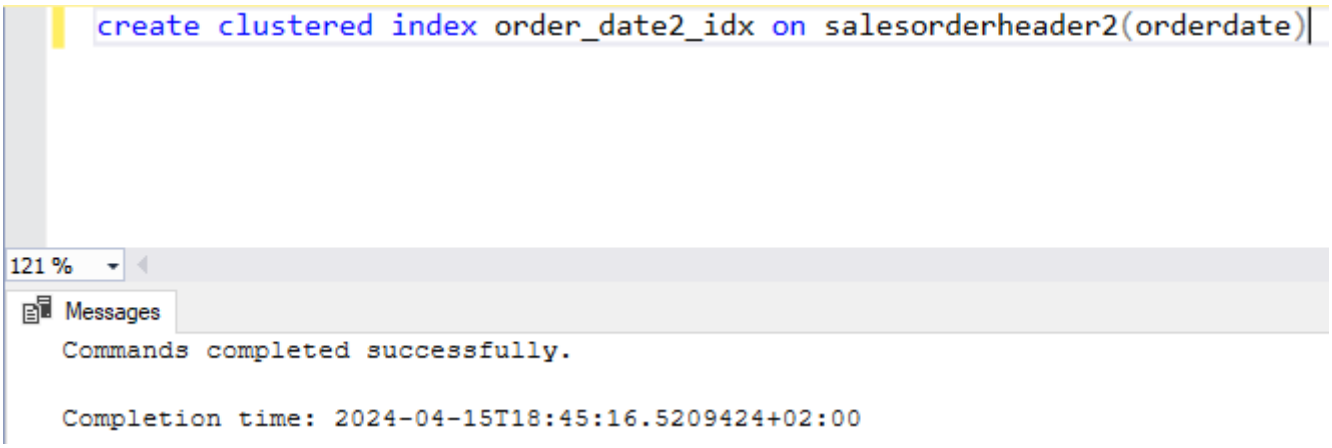
Results Messages

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerID	SalesPersonID	TerritoryID	BillToAddressID	ShipToAddressID	ShipMeth
64	43722	8	2011-06-06 00:00:00.000	2011-06-18 00:00:00.000	2011-06-13 00:00:00.000	5	1	SO43722	NULL	10-4030-013581	13581	NULL	10	23990	23990	1
65	43723	8	2011-06-06 00:00:00.000	2011-06-18 00:00:00.000	2011-06-13 00:00:00.000	5	1	SO43723	NULL	10-4030-014520	14520	NULL	1	12708	12708	1
66	43724	8	2011-06-06 00:00:00.000	2011-06-18 00:00:00.000	2011-06-13 00:00:00.000	5	1	SO43724	NULL	10-4030-016520	16520	NULL	9	25572	25572	1
67	43725	8	2011-06-07 00:00:00.000	2011-06-19 00:00:00.000	2011-06-14 00:00:00.000	5	1	SO43725	NULL	10-4030-013258	13258	NULL	8	23444	23444	1
68	43726	8	2011-06-07 00:00:00.000	2011-06-19 00:00:00.000	2011-06-14 00:00:00.000	5	1	SO43726	NULL	10-4030-014560	14560	NULL	4	25989	25989	1
69	43727	8	2011-06-07 00:00:00.000	2011-06-19 00:00:00.000	2011-06-14 00:00:00.000	5	1	SO43727	NULL	10-4030-016607	16607	NULL	9	15502	15502	1
70	43728	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43728	NULL	10-4030-027666	27666	NULL	4	20616	20616	1
71	43729	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43729	NULL	10-4030-011238	11238	NULL	10	12420	12420	1
72	43730	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43730	NULL	10-4030-025861	25861	NULL	4	23733	23733	1
73	43731	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43731	NULL	10-4030-016629	16629	NULL	9	14019	14019	1
74	43732	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43732	NULL	10-4030-011025	11025	NULL	9	21635	21635	1
75	43733	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43733	NULL	10-4030-027577	27577	NULL	1	24572	24572	1
76	43734	8	2011-06-09 00:00:00.000	2011-06-21 00:00:00.000	2011-06-16 00:00:00.000	5	1	SO43734	NULL	10-4030-027604	27604	NULL	1	23224	23224	1
77	43735	8	2011-06-09 00:00:00.000	2011-06-21 00:00:00.000	2011-06-16 00:00:00.000	5	1	SO43735	NULL	10-4030-016522	16522	NULL	9	25384	25384	1
78	43736	8	2011-06-09 00:00:00.000	2011-06-21 00:00:00.000	2011-06-16 00:00:00.000	5	1	SO43736	NULL	10-4030-011002	11002	NULL	9	20336	20336	1
79	43737	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43737	NULL	10-4030-013261	13261	NULL	8	29772	29772	1
80	43738	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43738	NULL	10-4030-011606	11606	NULL	7	26167	26167	1
81	43739	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43739	NULL	10-4030-013563	13563	NULL	10	29262	29262	1
82	43740	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43740	NULL	10-4030-016527	16527	NULL	9	28420	28420	1
83	43741	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43741	NULL	10-4030-027671	27671	NULL	1	13036	13036	1
84	43742	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43742	NULL	10-4030-013576	13576	NULL	10	20966	20966	1
85	43743	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43743	NULL	10-4030-011007	11007	NULL	9	12909	12909	1
86	43744	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43744	NULL	10-4030-016631	16631	NULL	9	12423	12423	1
87	43745	8	2011-06-12 00:00:00.000	2011-06-24 00:00:00.000	2011-06-19 00:00:00.000	5	1	SO43745	NULL	10-4030-016514	16514	NULL	9	25018	25018	1
88	43746	8	2011-06-12 00:00:00.000	2011-06-24 00:00:00.000	2011-06-19 00:00:00.000	5	1	SO43746	NULL	10-4030-016616	16616	NULL	9	28214	28214	1
89	43747	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43747	NULL	10-4030-027623	27623	NULL	4	27241	27241	1
90	43748	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43748	NULL	10-4030-027625	27625	NULL	1	16663	16663	1
91	43749	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43749	NULL	10-4030-027636	27636	NULL	1	17878	17878	1
92	43750	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43750	NULL	10-4030-011591	11591	NULL	7	26694	26694	1
93	43751	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43751	NULL	10-4030-011592	11592	NULL	7	17421	17421	1
94	43752	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43752	NULL	10-4030-013265	13265	NULL	8	25970	25970	1
95	43753	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43753	NULL	10-4030-016482	16482	NULL	9	19434	19434	1
96	43754	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43754	NULL	10-4030-027617	27617	NULL	1	18700	18700	1
97	43755	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43755	NULL	10-4030-027670	27670	NULL	4	21785	21785	1
98	43756	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43756	NULL	10-4030-019941	19941	NULL	10	12565	12565	1
99	43757	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43757	NULL	10-4030-011017	11017	NULL	9	11541	11541	1
100	43758	8	2011-06-15 00:00:00.000	2011-06-27 00:00:00.000	2011-06-22 00:00:00.000	5	1	SO43758	NULL	10-4030-027646	27646	NULL	4	14400	14400	1



Stwórz indeks klastrujący według OrderDate:

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



Wypisz ponownie sto pierwszych zamówień. Co się zmieniło?

```
select top 100 * from salesorderheader2
order by orderdate
```

121 %

Results Messages

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerID	SalesPersonID	TerritoryID	BillToAddressID	ShipToAddressID	ShipMeth
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	279	5	985	985	5
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	279	5	921	921	5
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	282	6	517	517	5
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	282	6	482	482	5
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	276	4	1073	1073	5
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	280	1	876	876	5
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	283	1	849	849	5
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	276	4	1074	1074	5
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	277	3	629	629	5
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	282	6	529	529	5
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	283	1	895	895	5
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	275	3	810	810	5
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	283	1	855	855	5
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	282	6	464	464	5
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	275	2	821	821	5
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	282	6	458	458	5
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	10-4020-000670	29827	277	3	631	631	5
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	SO43676	PO11861165059	10-4020-000017	29811	275	5	755	755	5
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	SO43677	PO11049174786	10-4020-000679	29824	278	6	556	556	5
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	SO43678	PO10817150168	10-4020-000203	29889	281	4	1021	1021	5
21	43679	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43679	PO10527142759	10-4020-000480	29761	278	6	525	525	5
22	43680	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43680	PO10730130087	10-4020-000491	29489	281	4	1069	1069	5
23	43681	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43681	PO1189177803	10-4020-000423	29661	279	5	955	955	5
24	43682	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43682	PO1566124200	10-4020-000486	29759	275	2	808	808	5
25	43683	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43683	PO2552113807	10-4020-000506	29497	283	1	889	889	5
26	43684	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43684	PO3393188842	10-4020-000549	29912	279	5	971	971	5
27	43685	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43685	PO4176124783	10-4020-000458	30084	279	5	958	958	5
28	43686	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43686	PO5075125561	10-4020-000344	29606	283	1	871	871	5
29	43687	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43687	PO4959110829	10-4020-000269	29610	275	2	783	783	5
30	43688	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43688	PO5365136389	10-4020-000161	29718	275	2	771	771	5
31	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	29646	277	4	578	578	5
32	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	29533	275	3	801	801	5
33	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	29570	277	4	592	592	5
34	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	29992	281	4	1024	1024	5
35	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	29491	275	5	807	807	5
36	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	29549	279	5	943	943	5
37	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	29958	279	5	911	911	5

```
select top 100 * from salesorderheader2
order by orderdate
```

121 %

Results Messages

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerID	SalesPersonID	TerritoryID	BillToAddressID	ShipToAddressID	ShipMeth
64	43722	8	2011-06-06 00:00:00.000	2011-06-18 00:00:00.000	2011-06-13 00:00:00.000	5	1	SO43722	NULL	10-4030-013581	13581	NULL	10	23990	23990	1
65	43723	8	2011-06-06 00:00:00.000	2011-06-18 00:00:00.000	2011-06-13 00:00:00.000	5	1	SO43723	NULL	10-4030-014520	14520	NULL	1	12708	12708	1
66	43724	8	2011-06-06 00:00:00.000	2011-06-18 00:00:00.000	2011-06-13 00:00:00.000	5	1	SO43724	NULL	10-4030-016520	16520	NULL	9	25572	25572	1
67	43725	8	2011-06-07 00:00:00.000	2011-06-19 00:00:00.000	2011-06-14 00:00:00.000	5	1	SO43725	NULL	10-4030-013258	13258	NULL	8	23444	23444	1
68	43726	8	2011-06-07 00:00:00.000	2011-06-19 00:00:00.000	2011-06-14 00:00:00.000	5	1	SO43726	NULL	10-4030-014560	14560	NULL	4	25989	25989	1
69	43727	8	2011-06-07 00:00:00.000	2011-06-19 00:00:00.000	2011-06-14 00:00:00.000	5	1	SO43727	NULL	10-4030-016607	16607	NULL	9	15502	15502	1
70	43728	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43728	NULL	10-4030-027666	27666	NULL	4	20616	20616	1
71	43729	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43729	NULL	10-4030-011238	11238	NULL	10	12420	12420	1
72	43730	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43730	NULL	10-4030-025861	25861	NULL	4	23733	23733	1
73	43731	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43731	NULL	10-4030-016629	16629	NULL	9	14019	14019	1
74	43732	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43732	NULL	10-4030-011025	11025	NULL	9	21635	21635	1
75	43733	8	2011-06-08 00:00:00.000	2011-06-20 00:00:00.000	2011-06-15 00:00:00.000	5	1	SO43733	NULL	10-4030-027577	27577	NULL	1	24572	24572	1
76	43734	8	2011-06-09 00:00:00.000	2011-06-21 00:00:00.000	2011-06-16 00:00:00.000	5	1	SO43734	NULL	10-4030-027604	27604	NULL	1	23224	23224	1
77	43735	8	2011-06-09 00:00:00.000	2011-06-21 00:00:00.000	2011-06-16 00:00:00.000	5	1	SO43735	NULL	10-4030-016522	16522	NULL	9	25384	25384	1
78	43736	8	2011-06-09 00:00:00.000	2011-06-21 00:00:00.000	2011-06-16 00:00:00.000	5	1	SO43736	NULL	10-4030-011002	11002	NULL	9	20336	20336	1
79	43737	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43737	NULL	10-4030-013261	13261	NULL	8	29772	29772	1
80	43738	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43738	NULL	10-4030-011606	11606	NULL	7	26167	26167	1
81	43739	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43739	NULL	10-4030-013563	13563	NULL	10	29262	29262	1
82	43740	8	2011-06-10 00:00:00.000	2011-06-22 00:00:00.000	2011-06-17 00:00:00.000	5	1	SO43740	NULL	10-4030-016527	16527	NULL	9	28420	28420	1
83	43741	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43741	NULL	10-4030-027671	27671	NULL	1	13036	13036	1
84	43742	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43742	NULL	10-4030-013576	13576	NULL	10	20966	20966	1
85	43743	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43743	NULL	10-4030-011007	11007	NULL	9	12909	12909	1
86	43744	8	2011-06-11 00:00:00.000	2011-06-23 00:00:00.000	2011-06-18 00:00:00.000	5	1	SO43744	NULL	10-4030-016631	16631	NULL	9	12423	12423	1
87	43745	8	2011-06-12 00:00:00.000	2011-06-24 00:00:00.000	2011-06-19 00:00:00.000	5	1	SO43745	NULL	10-4030-016514	16514	NULL	9	25018	25018	1
88	43746	8	2011-06-12 00:00:00.000	2011-06-24 00:00:00.000	2011-06-19 00:00:00.000	5	1	SO43746	NULL	10-4030-016616	16616	NULL	9	28214	28214	1
89	43747	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43747	NULL	10-4030-027623	27623	NULL	4	27241	27241	1
90	43748	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43748	NULL	10-4030-027625	27625	NULL	1	16663	16663	1
91	43749	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43749	NULL	10-4030-027636	27636	NULL	1	17878	17878	1
92	43750	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43750	NULL	10-4030-011591	11591	NULL	7	26694	26694	1
93	43751	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43751	NULL	10-4030-011592	11592	NULL	7	17421	17421	1
94	43752	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43752	NULL	10-4030-013265	13265	NULL	8	25970	25970	1
95	43753	8	2011-06-13 00:00:00.000	2011-06-25 00:00:00.000	2011-06-20 00:00:00.000	5	1	SO43753	NULL	10-4030-016482	16482	NULL	9	19434	19434	1
96	43754	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43754	NULL	10-4030-027617	27617	NULL	1	18700	18700	1
97	43755	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43755	NULL	10-4030-027670	27670	NULL	4	21785	21785	1
98	43756	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43756	NULL	10-4030-019941	19941	NULL	10	12565	12565	1
99	43757	8	2011-06-14 00:00:00.000	2011-06-26 00:00:00.000	2011-06-21 00:00:00.000	5	1	SO43757	NULL	10-4030-011017	11017	NULL	9	11541	11541	1
100	43758	8	2011-06-15 00:00:00.000	2011-06-27 00:00:00.000	2011-06-22 00:00:00.000	5	1	SO43758	NULL	10-4030-027646	27646	NULL	4	14400	14400	1

```
select top 100 * from salesorderheader2
order by orderdate
```

121 %

Results Messages Live Query Statistics Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select top 100 * from salesorderheader2 order by orderdate
```

SELECT
Cost: 0 %

Top
Cost: 0 %
0.000s

Clustered Index Scan (Clustered)
[salesorderheader2].[order_date2_id…
Cost: 100 %
0.000s

> Wyniki:

W wynikach nie zmieniło się nic.

W analizie zapytań, możemy zauważyć, że w pierwszym przypadku, gdzie nie mamy indeksu jest realizowana operacja sortowania, która jest bardzo kosztowa i stanowi większą część kosztu zapytania. W drugim przypadku, dzięki zastosowaniu indeksu klastrującego na kolumnę, po której sortujemy w naszym zapytaniu pozbywamy się konieczności sortowania, czyli najbardziej kosztownej operacji, a więc koszt zapytania się zmniejsza.

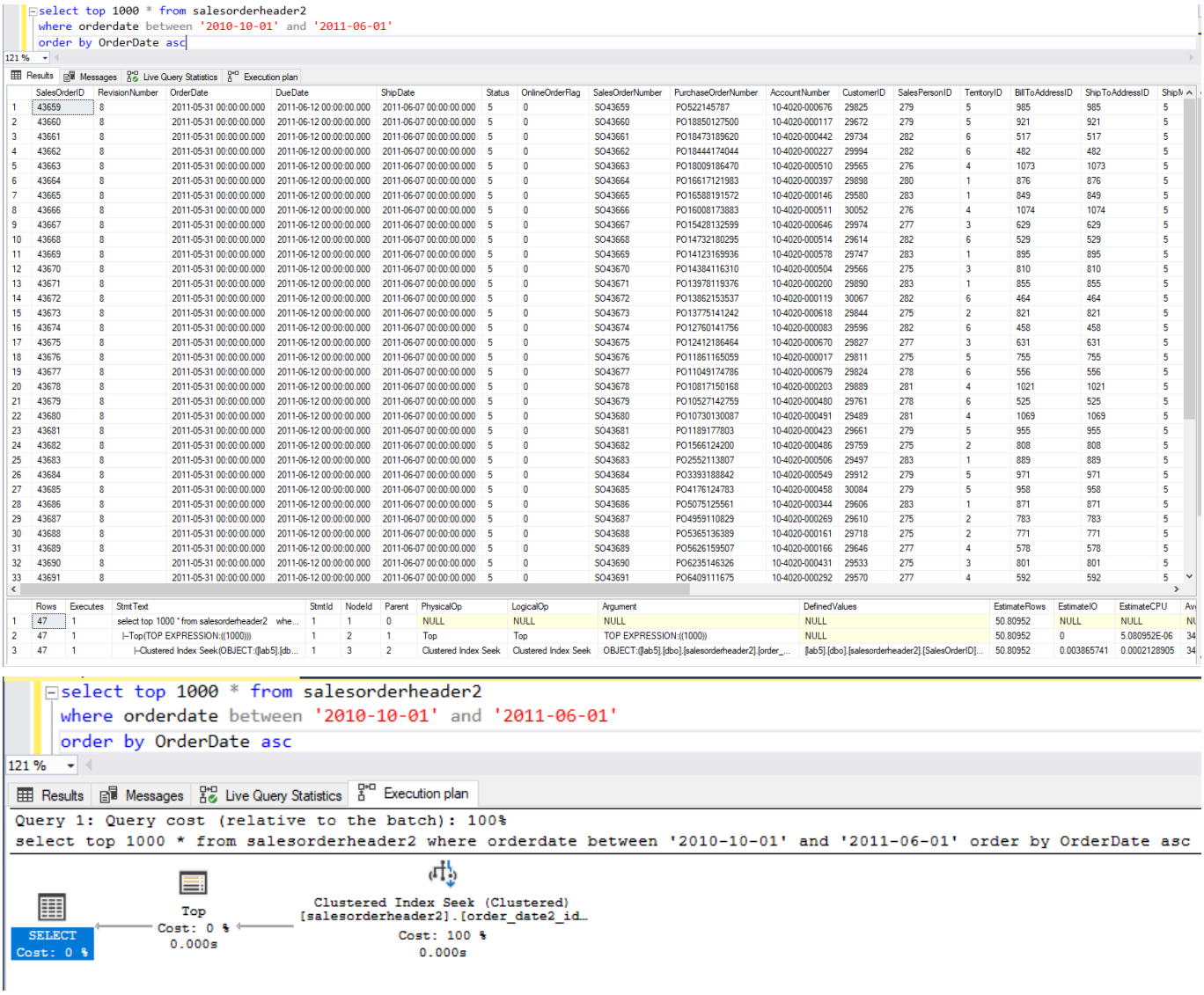
Sprawdź zapytanie:

```
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?

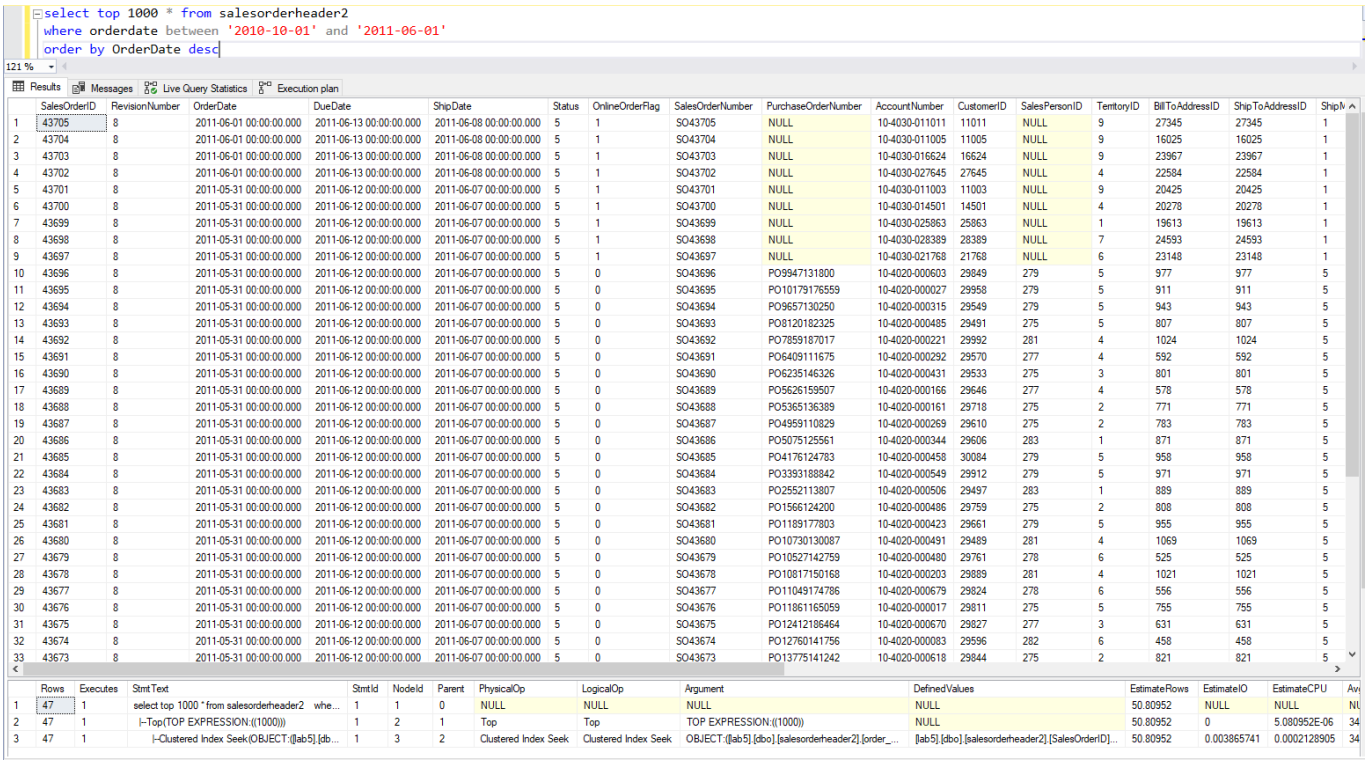
> Wyniki:

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



Widzimy, że indeks został wykorzystany, na co wskazuje operacja `Clustered Index Seek`. Nie jest wykonywane dodatkowe sortowanie.

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



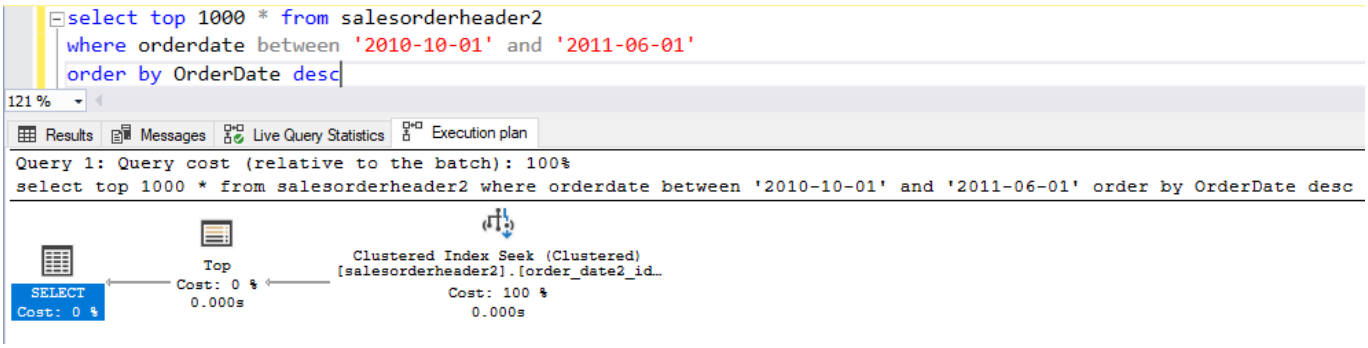
```
select top 1000 * from salesorderheader2
where orderdate between '2010-10-01' and '2011-06-01'
order by OrderDate desc
```

121 %

Results | Messages | Live Query Statistics | Execution plan

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerID	SalesPersonID	TerritoryID	BillToAddressID	ShipToAddressID	ShipM
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	11011	NULL	9	27345	27345	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	11005	NULL	9	16025	16025	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	16624	NULL	9	23967	23967	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	27645	NULL	4	22584	22584	1
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	11003	NULL	9	20425	20425	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	14501	NULL	4	20278	20278	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	25863	NULL	1	19613	19613	1
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	28389	NULL	7	24593	24593	1
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	21768	NULL	6	23148	23148	1
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	29849	279	5	977	977	5
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	29958	279	5	911	911	5
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	29549	279	5	943	943	5
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	29491	275	5	807	807	5
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	29992	281	4	1024	1024	5
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	29570	277	4	592	592	5
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	29533	275	3	801	801	5
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	29646	277	4	578	578	5
18	43688	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43688	PO5365136389	10-4020-000161	29718	275	2	771	771	5
19	43687	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43687	PO4959110829	10-4020-000269	29610	275	2	783	783	5
20	43686	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43686	PO5075125561	10-4020-000344	29606	283	1	871	871	5
21	43685	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43685	PO4176124783	10-4020-000458	30084	279	5	958	958	5
22	43684	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43684	PO3393188842	10-4020-000549	29912	279	5	971	971	5
23	43683	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43683	PO2552113807	10-4020-000506	29497	283	1	889	889	5
24	43682	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43682	PO1566124200	10-4020-000486	29759	275	2	808	808	5
25	43681	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43681	PO1189177803	10-4020-000423	29661	279	5	955	955	5
26	43680	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43680	PO10730130087	10-4020-000491	29489	281	4	1069	1069	5
27	43679	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43679	PO10527142759	10-4020-000480	29761	278	6	525	525	5
28	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	SO43678	PO10817150168	10-4020-000203	29889	281	4	1021	1021	5
29	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	SO43677	PO11049174786	10-4020-000679	29824	278	6	556	556	5
30	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	SO43676	PO11861165059	10-4020-000017	29811	275	5	755	755	5
31	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	10-4020-000670	29827	277	3	631	631	5
32	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	282	6	458	458	5
33	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	275	2	821	821	5

	Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	DefinedValues	EstimateRows	EstimateIO	EstimateCPU	Av
1	47	1	select top 1000 * from salesorderheader2 whe...	1	1	0	NULL	NULL	NULL	NULL	50.80952	NULL	NULL	NU
2	47	1	|--Top(TOP EXPRESSION:((1000)))	1	2	1	Top	Top	TOP EXPRESSION:((1000))	NULL	50.80952	0	5.080952E-06	34
3	47	1	|--Clustered Index Seek(OBJECT:([lab5].[db...	1	3	2	Clustered Index Seek	Clustered Index Seek	OBJECT:([lab5].[dbo].[salesorderheader2].[order_...	[lab5].[dbo].[salesorderheader2].[SalesOrderID]...	50.80952	0.003865741	0.0002128905	34



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

121 %

Results | Messages | Live Query Statistics | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
select top 1000 * from salesorderheader2 where orderdate between '2010-10-01' and '2011-06-01' order by OrderDate desc
```

SELECT
Cost: 0 %

Top
Cost: 0 %
0.000s

Clustered Index Seek (Clustered)
[salesorderheader2].[order_date2_id…
Cost: 100 %
0.000s

Widzimy, że indeks został wykorzystany, na co wskazuje operacja `Clustered Index Seek`. Nie jest wykonywane dodatkowe sortowanie.

Indeks działa w obu przypadkach dzięki czemu w obu przypadkach unikamy konieczności sortowania.

Zadanie 3 – indeksy column store

Celem zadania jest poznanie indeksów typu column store![]
(file:////Users/rm/Library/Group%20Containers/UBF8T346G9.Office/TemporaryItems/msohtmlclip/clip_image001.jpg)

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)
```

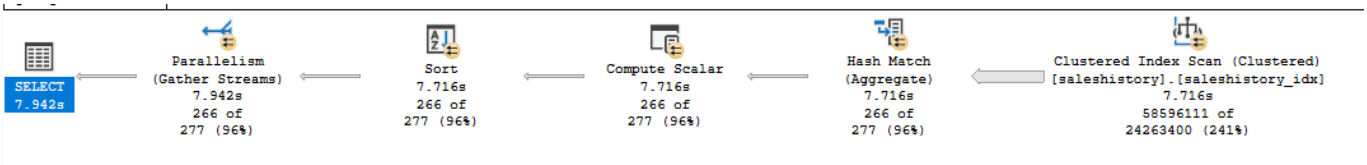
Sprawdź różnicę pomiędzy przetwarzaniem w zależności od indeksów. Porównaj plany i opisz różnicę.

> Wyniki:

Zapytania zostały wykonane na tabeli z 52 mln wierszy

Clustered index

	productid	(No column name)	(No column name)	(No column name)	(No column name)
1	707	45992795,8434	30,8865	3026478	2
2	708	44224316,073	30,4495	3154956	2
3	709	513590,805	5,656	534681	5
4	710	121136,40	5,70	43470	2
5	711	45318854,7378	30,365	3256869	2
6	712	12526101,2793	7,6682	4014213	2
7	713	10358277,93	49,99	207207	1
8	714	21638124,4485	36,7811	1756188	2
9	715	27561511,6896	34,901	3183936	4
10	716	19314950,5248	37,165	1439340	2
11	717	85713673,9158	814,0413	234255	2



SELECT

Estimated operator progress: 100%

Actual Number of Rows for All Executions	266
Cached plan size	56 KB
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	360,019
Estimated Number of Rows Per Execution	277
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
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Batch
Storage	RowStore
Actual Number of Rows for All Executions	58596111
Estimated I/O Cost	688,24
Estimated Operator Cost	698,983 (99%)
Estimated CPU Cost	10,7426
Estimated Subtree Cost	698,983
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	21 B
Ordered	False
Node ID	4

Object

[lab6].[dbo].[saleshistory].[saleshistory_idx]

Output List

[lab6].[dbo].[saleshistory].orderqty; [lab6].[dbo].[saleshistory].productid; [lab6].[dbo].[saleshistory].unitprice

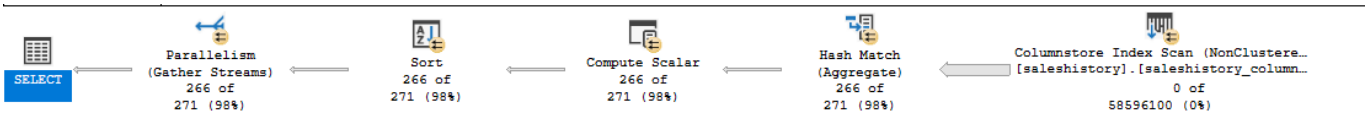
Czas wykonania około: 8s

Columnstore index

Zakładnie column store indexa trwało około: 41 sek

Czas wykonania: 0s

	productid	(No column name)	(No column name)	(No column name)	(No column name)
1	707	45992795,8434	30,8865	3026478	2
2	708	44224316,073	30,4495	3154956	2
3	709	513590,805	5,656	534681	5
4	710	121136,40	5,70	43470	2
5	711	45318854,7378	30,365	3256869	2
6	712	12526101,2793	7,6682	4014213	2
7	713	10358277,93	49,99	207207	1
8	714	21638124,4485	36,7811	1756188	2
9	715	27561511,6896	34,901	3183936	4
10	716	19314950,5248	37,165	1439340	2
11	717	85713673,9158	814,0413	234255	2



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
```

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

Porównanie

Jak widać columnstore mimo dość długiego procesu zakładania indeksu, drastycznie zwiększa prędkość odczytu. Estimated Subtree Cost zwykłego indeksu to 360, columnstora to 6.

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
- 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:

Eksperyment 1 - Indeks warunkowy (Filtered Index)

Rozpoczynamy od stworzenia tabeli `Products`, która będzie przechowywać informacje o produktach, w tym nazwę, kategorię, cenę i ilość w magazynie. Następnie wypełniamy tę tabelę 20,000 rekordami.

```
CREATE TABLE Products (
    ProductID INT IDENTITY(1,1),
    ProductName NVARCHAR(255),
    Category NVARCHAR(50),
    Price DECIMAL(10,2),
    StockQuantity INT
);

DECLARE @i INT = 1;
WHILE @i <= 20000
BEGIN
    INSERT INTO Products (ProductName, Category, Price, StockQuantity)
    VALUES (
        CONCAT('Product', @i),
        CASE
            WHEN @i % 3 = 0 THEN 'Electronics'
            WHEN @i % 3 = 1 THEN 'Clothing'
```

```
            ELSE 'Books'
        END,
        RAND() * 1000,
        RAND() * 100
    );
    SET @i = @i + 1;
END;
```

```
select * from products;
```

121 %

Results Messages

	ProductID	ProductName	Category	Price	StockQuantity
1	1	Product1	Clothing	12.47	59
2	2	Product2	Books	378.83	30
3	3	Product3	Electronics	630.13	56
4	4	Product4	Clothing	914.93	14
5	5	Product5	Books	180.87	93
6	6	Product6	Electronics	704.04	37
7	7	Product7	Clothing	94.82	16
8	8	Product8	Books	478.85	7
9	9	Product9	Electronics	382.01	89
10	10	Product10	Clothing	733.91	26
11	11	Product11	Books	754.50	93
12	12	Product12	Electronics	982.00	24
13	13	Product13	Clothing	127.91	74
14	14	Product14	Books	189.84	72
15	15	Product15	Electronics	997.23	34
16	16	Product16	Clothing	726.42	52
17	17	Product17	Books	284.66	10
18	18	Product18	Electronics	474.10	97
19	19	Product19	Clothing	666.16	77
20	20	Product20	Books	121.50	3
21	21	Product21	Electronics	461.20	47
22	22	Product22	Clothing	725.53	83
23	23	Product23	Books	828.59	32
24	24	Product24	Electronics	734.88	69
25	25	Product25	Clothing	340.48	89
26	26	Product26	Books	686.69	38
27	27	Product27	Electronics	357.42	29
28	28	Product28	Clothing	717.24	51
29	29	Product29	Books	989.27	23
30	30	Product30	Electronics	880.18	62
31	31	Product31	Clothing	757.59	60
32	32	Product32	Books	746.00	81
33	33	Product33	Electronics	985.95	59
34	34	Product34	Clothing	245.02	30
35	35	Product35	Books	764.00	31
36	36	Product36	Electronics	990.58	82
37	37	Product37	Clothing	763.07	87
38	38	Product38	Books	372.11	87
39	39	Product39	Electronics	891.76	26
40	40	Product40	Clothing	564.90	80

Chcemy wybrać produkty z kategorii "Electronics".

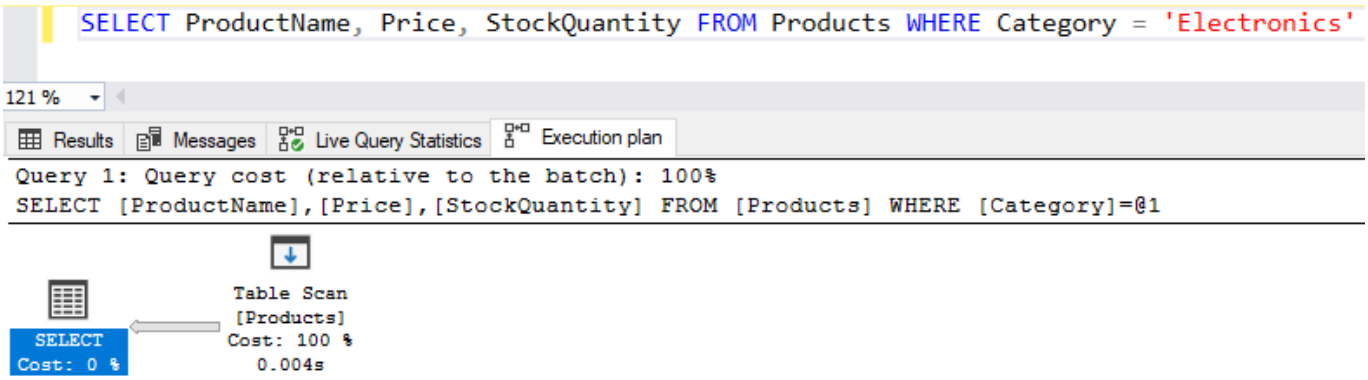
```
SELECT ProductName, Price, StockQuantity FROM Products WHERE Category =
'Electronics'
```

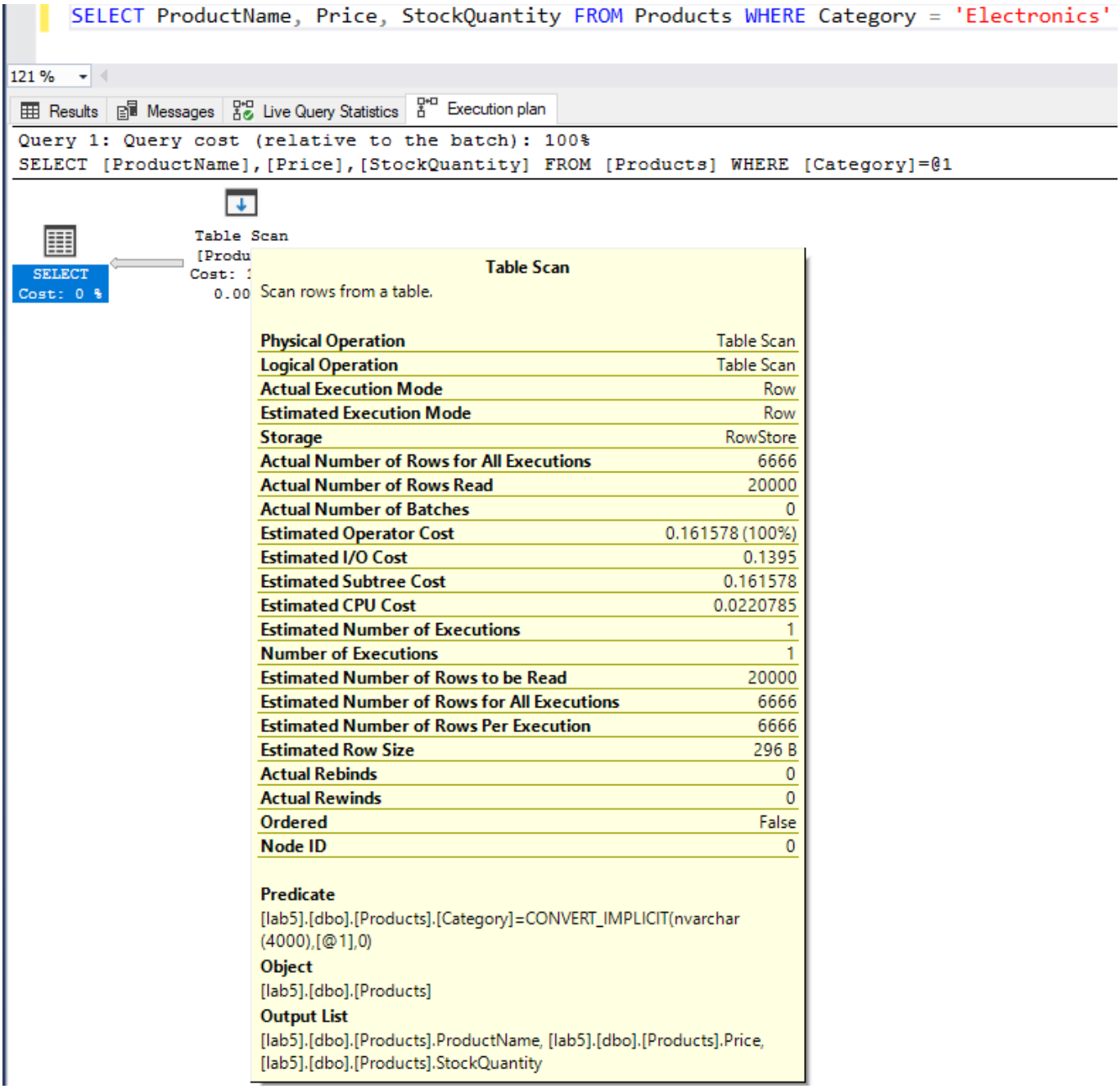
Tworzymy indeks warunkowy na kolumnie Category dla kategorii "Electronics" i dodajemy pozostałe kolumny jako INCLUDE.

```
CREATE NONCLUSTERED INDEX IX_Filtered_Category_Electronics
ON Products (Category)
INCLUDE(ProductName, Price, StockQuantity)
WHERE Category = 'Electronics';
```

> Wyniki

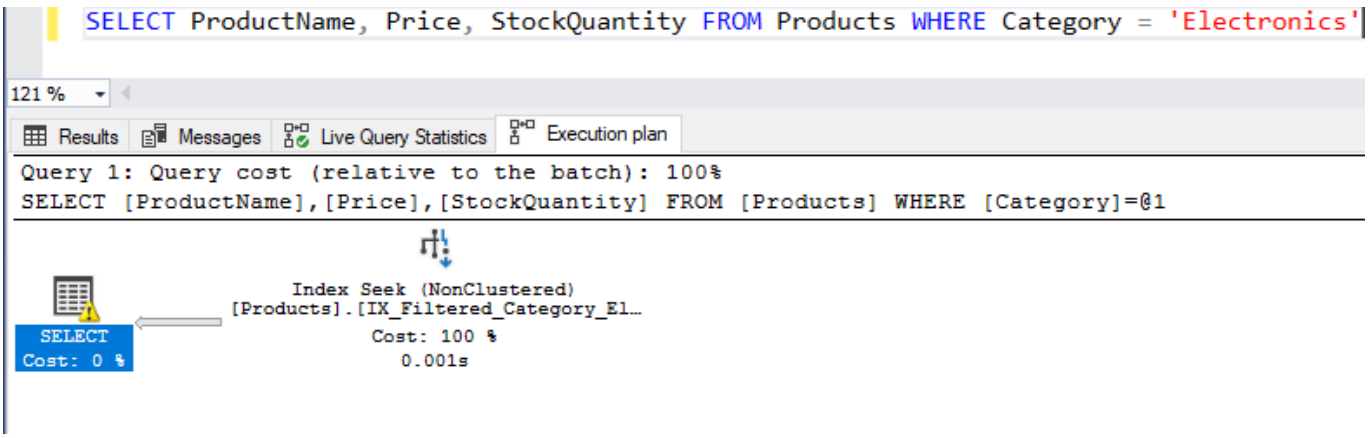
Bez indeksu

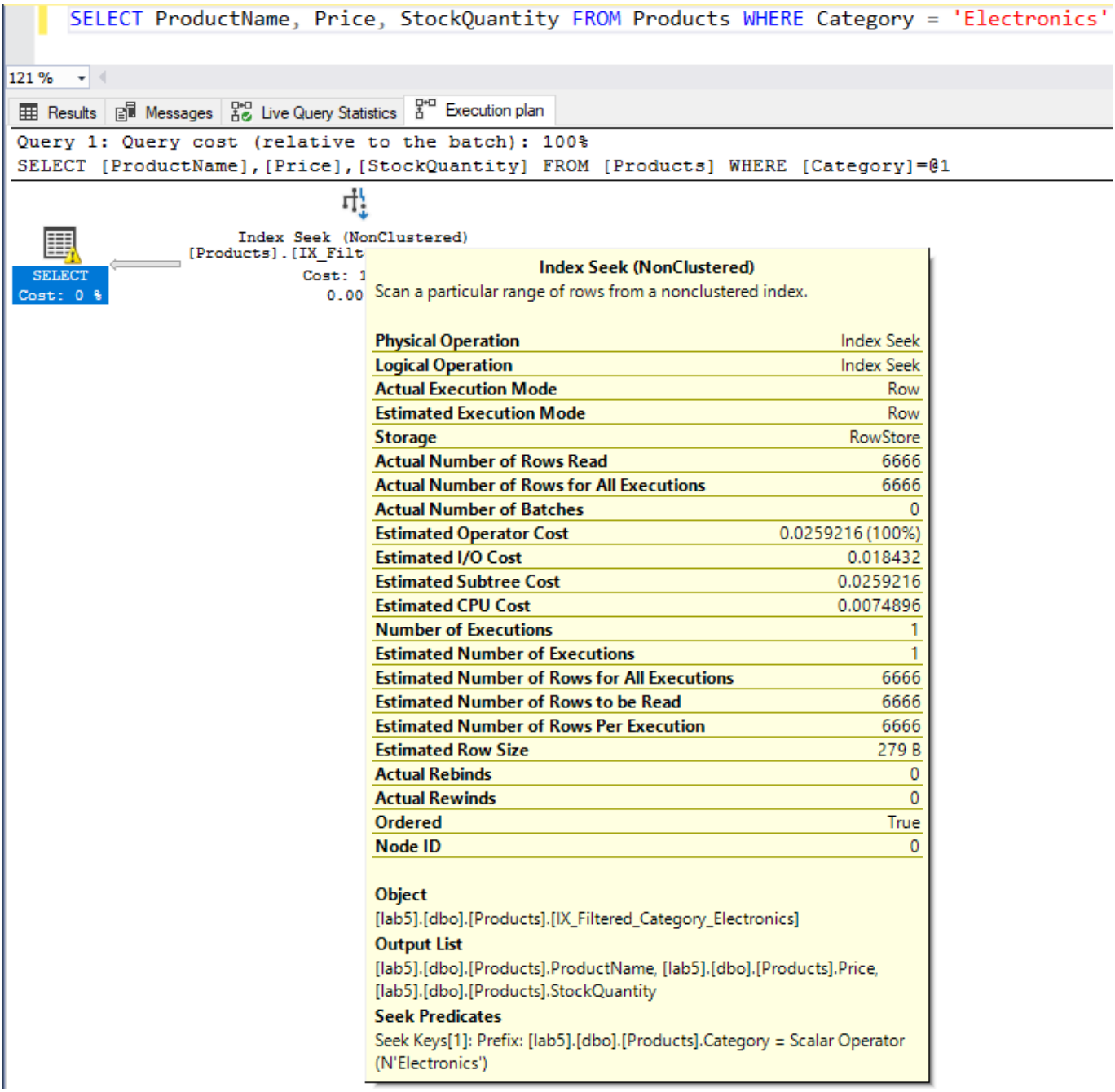




Zapytanie wykonuje pełne skanowanie tabeli, aby znaleźć odpowiednie rekordy.

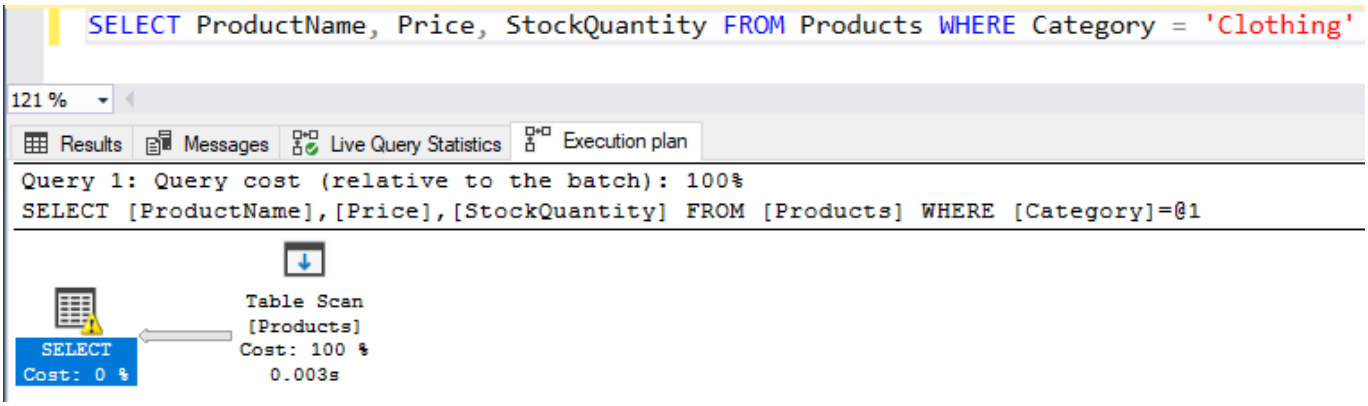
Z indeksem w jego zasięgu

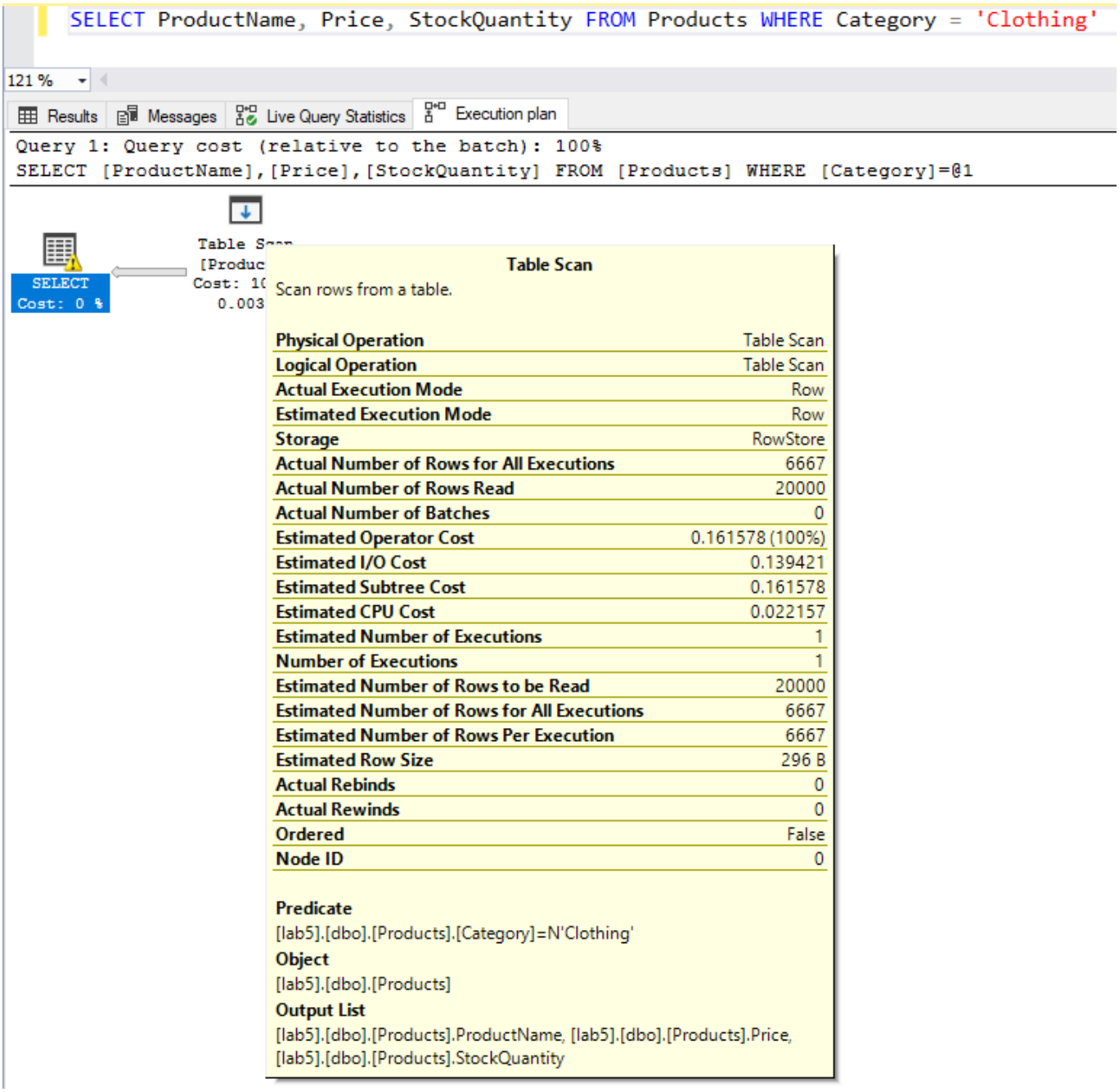




Dzięki indeksowi, koszt zapytania jest znacznie niższy, a liczba operacji wejścia/wyjścia jest mniejsza.

Z indeksem poza jego zasięgiem

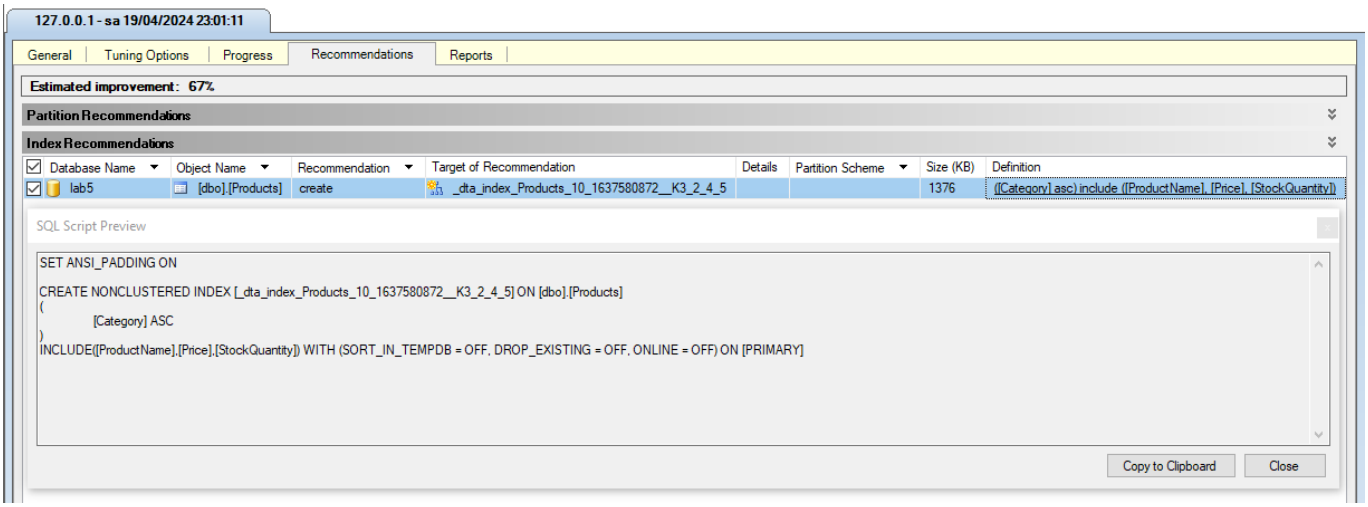




W tym przypadku, gdy warunek filtru nie jest spełniony, wykonuje się pełne skanowanie tabeli.

Rekomendacja Database Engine Tuning Advisor

Narzędzie sugeruje utworzenie identycznego indeksu z dodatkową rekomendacją uporządkowania kategorii rosnąco.



Wnioski

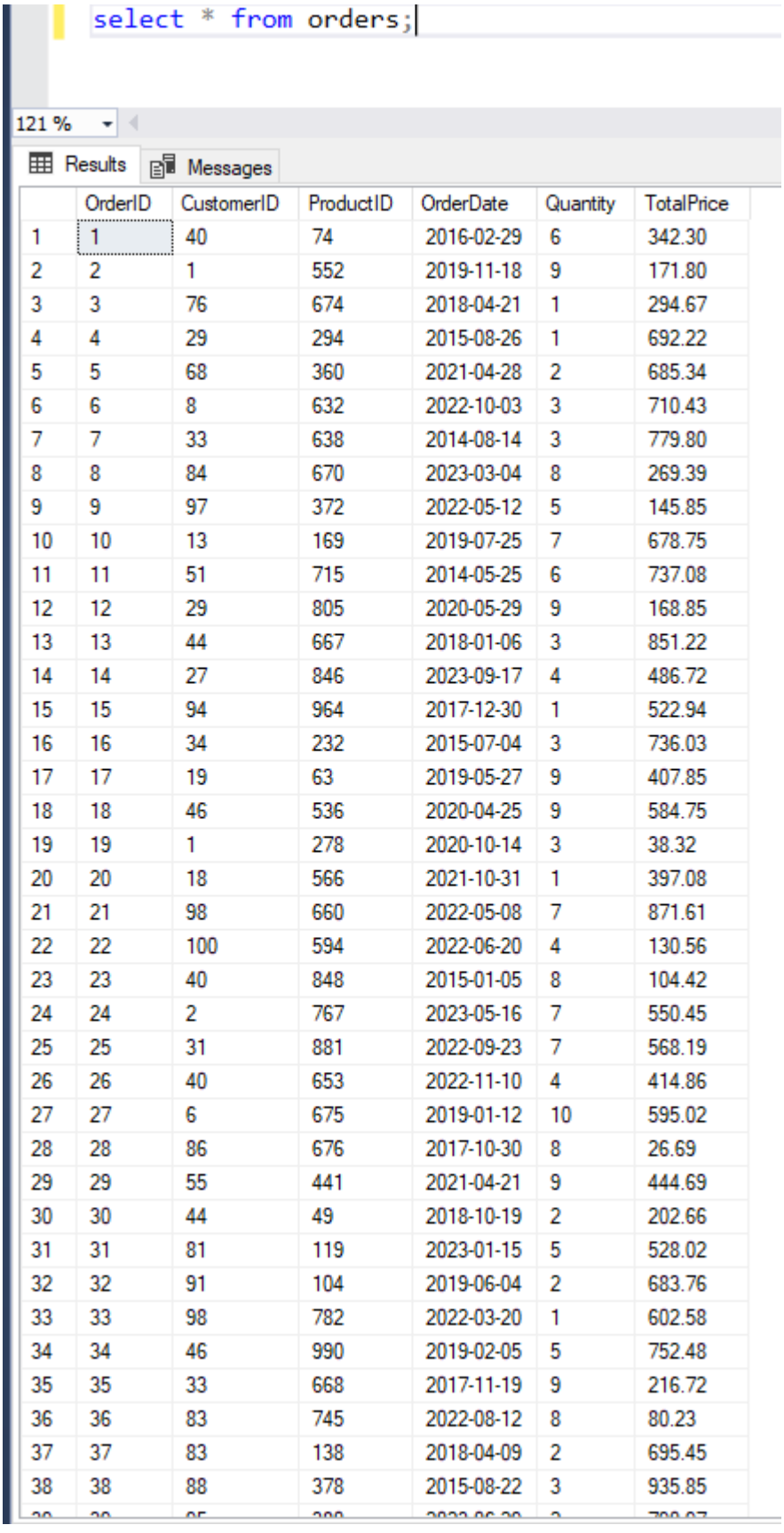
Indeks warunkowy jest skuteczny w przypadku, gdy zapytanie pokrywa się z jego warunkami. Jednakże, gdy warunki nie są spełnione, może dojść do pełnego skanowania tabeli, co jest kosztowne. Stosowanie tego rodzaju indeksów ma sens, gdy często wyszukujemy dane z konkretnie określonej kategorii.

Eksperyment 2 - Indeks kolumnowy (Column Index)

Tworzymy tabelę `Orders`, która będzie przechowywać informacje o zamówieniach, w tym identyfikator klienta, produktu, datę zamówienia, ilość i łączną cenę. Następnie wypełniamy tę tabelę 50,000 rekordami.

```
CREATE TABLE Orders (
    OrderID INT IDENTITY(1,1),
    CustomerID INT,
    ProductID INT,
    OrderDate DATE,
    Quantity INT,
    TotalPrice DECIMAL(10,2)
);

DECLARE @i INT = 1;
WHILE @i <= 50000
BEGIN
    INSERT INTO Orders (CustomerID, ProductID, OrderDate, Quantity, TotalPrice)
    VALUES (
        FLOOR(RAND()*(100-1+1))+1,
        FLOOR(RAND()*(1000-1+1))+1,
        DATEADD(DAY, -RAND()*(365*10), GETDATE()),
        FLOOR(RAND()*(10-1+1))+1,
        RAND() * 1000
    );
    SET @i = @i + 1;
END;
```



```
select * from orders;
```

121 %

Results Messages

	OrderID	CustomerID	ProductID	OrderDate	Quantity	TotalPrice
1	1	40	74	2016-02-29	6	342.30
2	2	1	552	2019-11-18	9	171.80
3	3	76	674	2018-04-21	1	294.67
4	4	29	294	2015-08-26	1	692.22
5	5	68	360	2021-04-28	2	685.34
6	6	8	632	2022-10-03	3	710.43
7	7	33	638	2014-08-14	3	779.80
8	8	84	670	2023-03-04	8	269.39
9	9	97	372	2022-05-12	5	145.85
10	10	13	169	2019-07-25	7	678.75
11	11	51	715	2014-05-25	6	737.08
12	12	29	805	2020-05-29	9	168.85
13	13	44	667	2018-01-06	3	851.22
14	14	27	846	2023-09-17	4	486.72
15	15	94	964	2017-12-30	1	522.94
16	16	34	232	2015-07-04	3	736.03
17	17	19	63	2019-05-27	9	407.85
18	18	46	536	2020-04-25	9	584.75
19	19	1	278	2020-10-14	3	38.32
20	20	18	566	2021-10-31	1	397.08
21	21	98	660	2022-05-08	7	871.61
22	22	100	594	2022-06-20	4	130.56
23	23	40	848	2015-01-05	8	104.42
24	24	2	767	2023-05-16	7	550.45
25	25	31	881	2022-09-23	7	568.19
26	26	40	653	2022-11-10	4	414.86
27	27	6	675	2019-01-12	10	595.02
28	28	86	676	2017-10-30	8	26.69
29	29	55	441	2021-04-21	9	444.69
30	30	44	49	2018-10-19	2	202.66
31	31	81	119	2023-01-15	5	528.02
32	32	91	104	2019-06-04	2	683.76
33	33	98	782	2022-03-20	1	602.58
34	34	46	990	2019-02-05	5	752.48
35	35	33	668	2017-11-19	9	216.72
36	36	83	745	2022-08-12	8	80.23
37	37	83	138	2018-04-09	2	695.45
38	38	88	378	2015-08-22	3	935.85

Chcemy zsumować wartość zamówień dla określonego produktu w określonym przedziale czasowym.

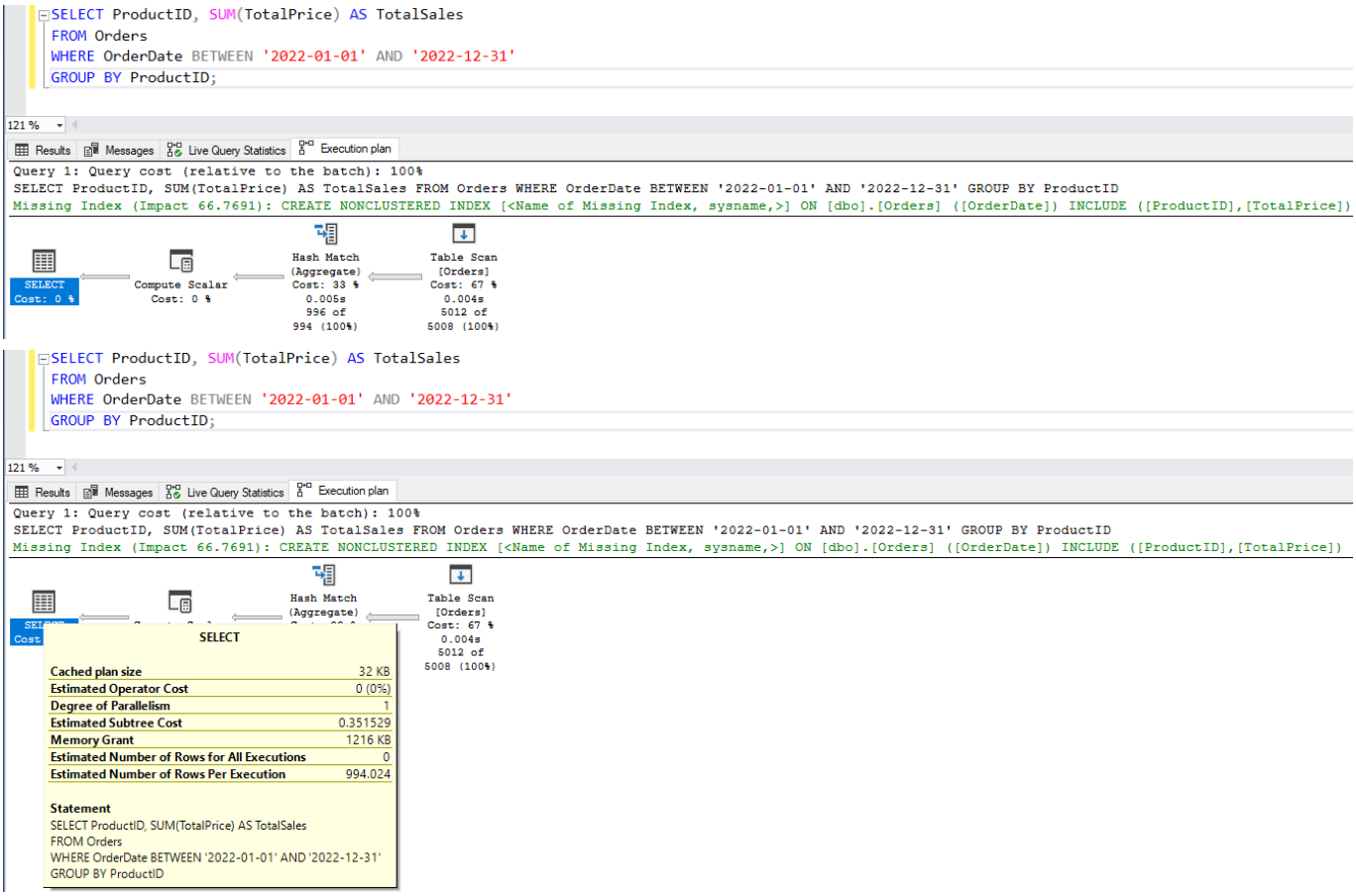
```sql
SELECT ProductID, SUM(TotalPrice) AS TotalSales
FROM Orders
WHERE OrderDate BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY ProductID;
```

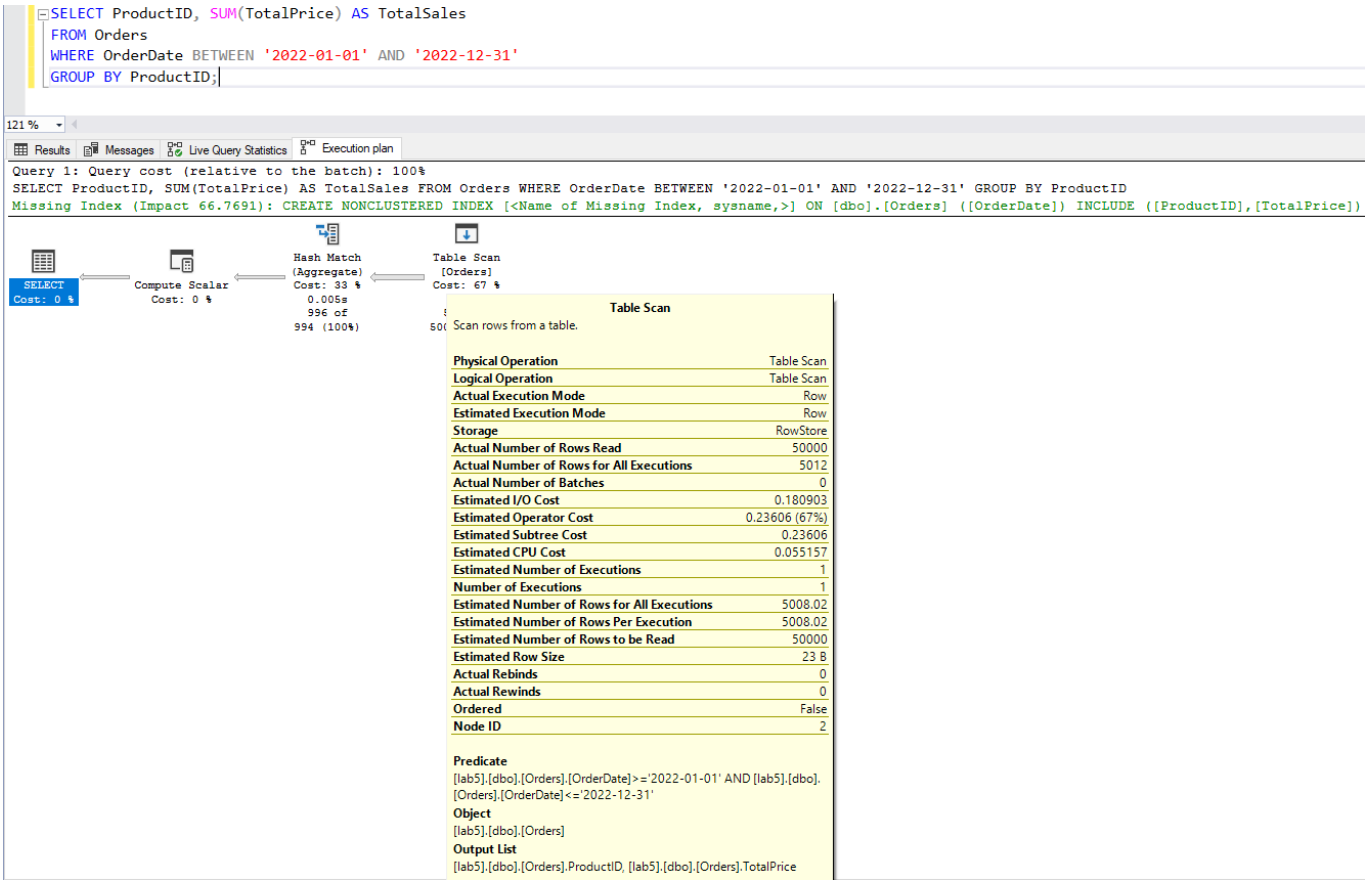
Tworzymy indeks kolumnowy na kolumnie TotalPrice.

```
CREATE NONCLUSTERED COLUMNSTORE INDEX IX_Column_TotalPrice
ON Orders (TotalPrice);
```

> Wyniki

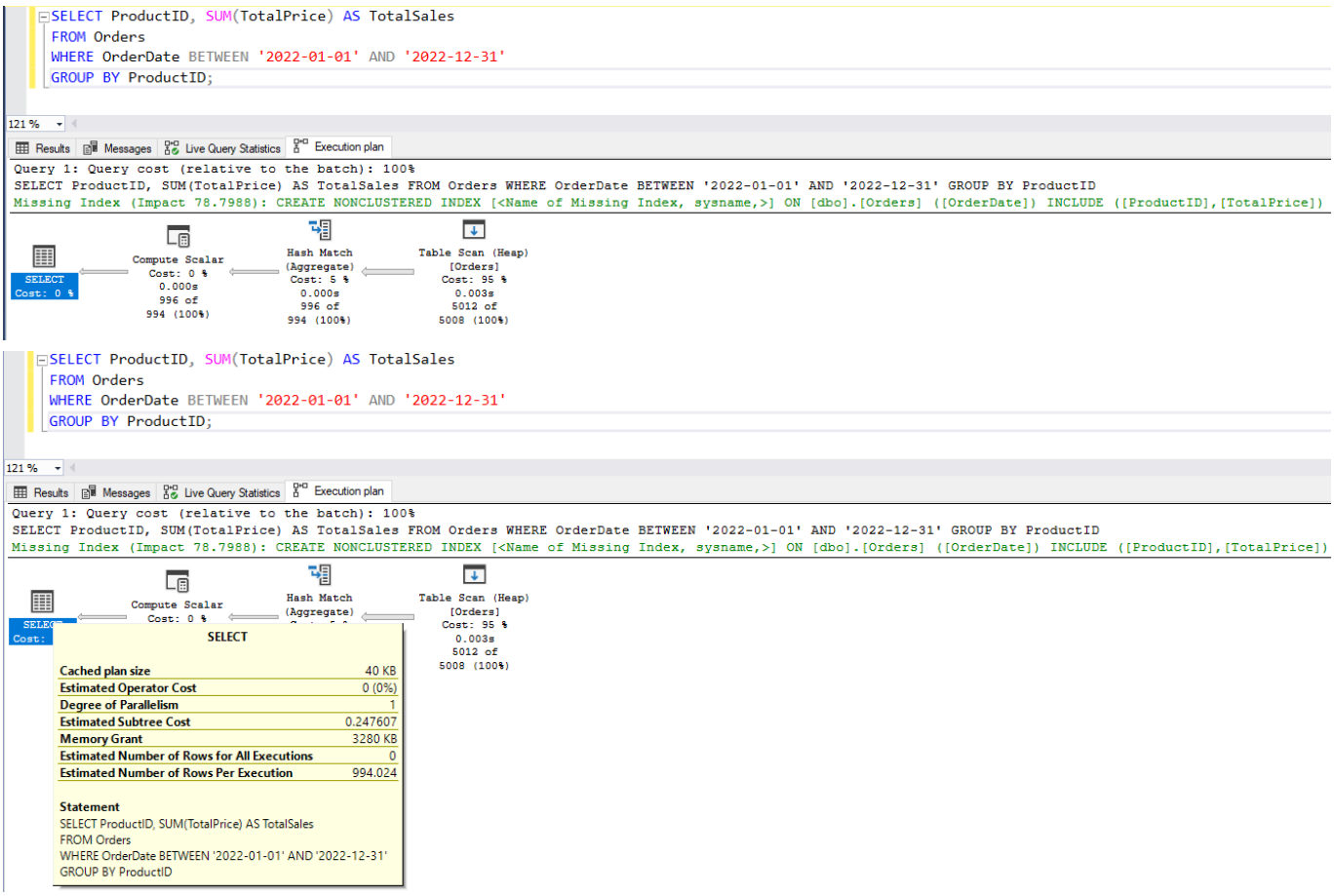
Bez indeksu

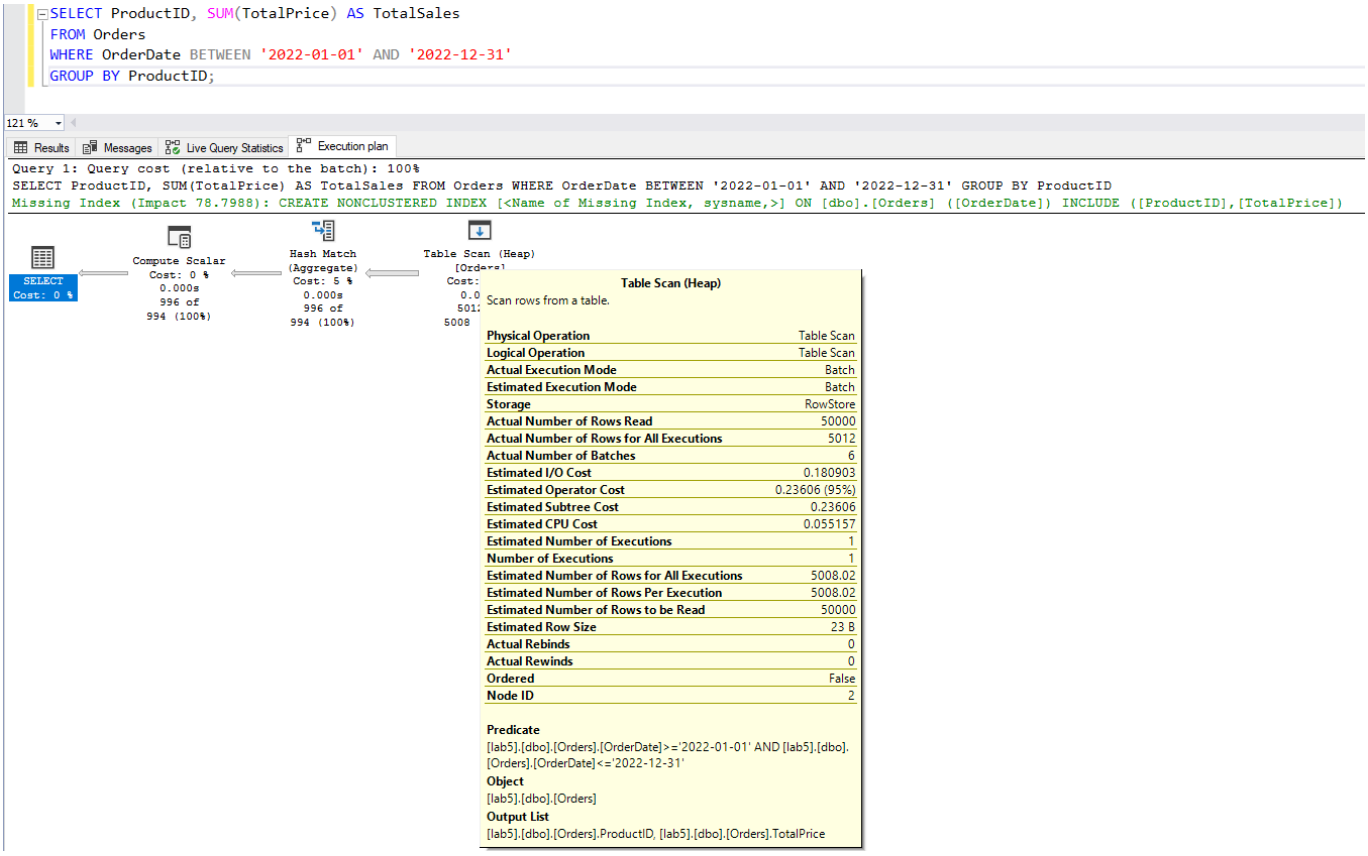




Zapytanie wykonuje pełne skanowanie tabeli, aby obliczyć sumę wartości zamówień dla określonego przedziału czasowego.

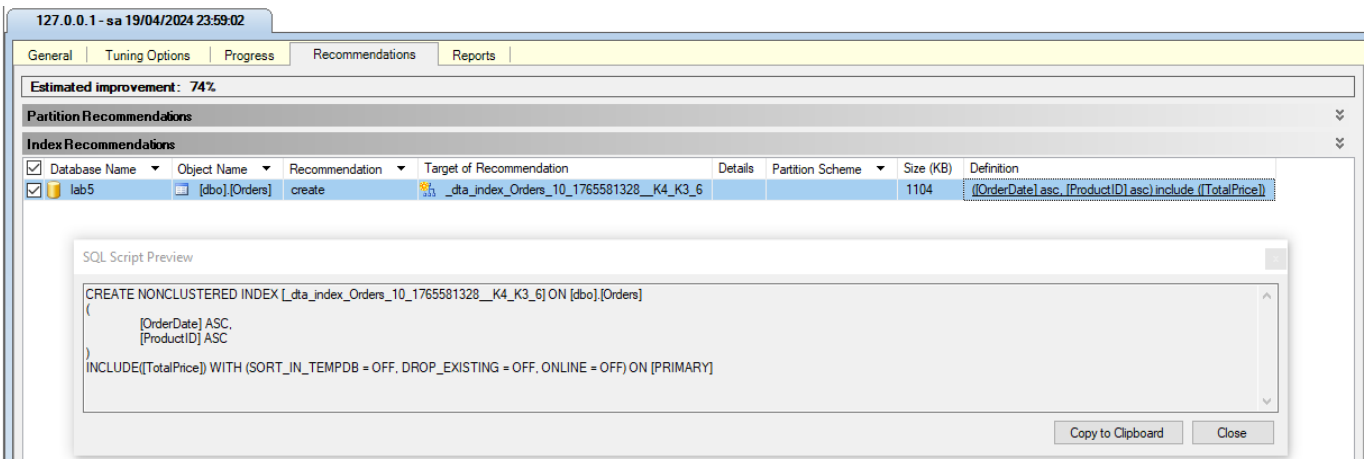
Z indeksem





Dzięki indeksowi kolumnowemu, koszt zapytania jest znacznie niższy, ponieważ SZBD może szybko uzyskać dostęp do wartości TotalPrice, co umożliwia szybsze obliczenie sumy. Bez wykorzystania indeksu operacja agregująca dane stanowi aż 33% kosztu zapytania, gdzie po wykorzystaniu indeksu ta wartość spada do 5% a jej koszt i czas wykonania są znacznie niższe i bliskie zeru.

Rekomendacja Database Engine Tuning Advisor



Narzędzie Database Engine Tuning Advisor sugeruje utworzenie indeksu, który ma kolumnę `TotalPrice` 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

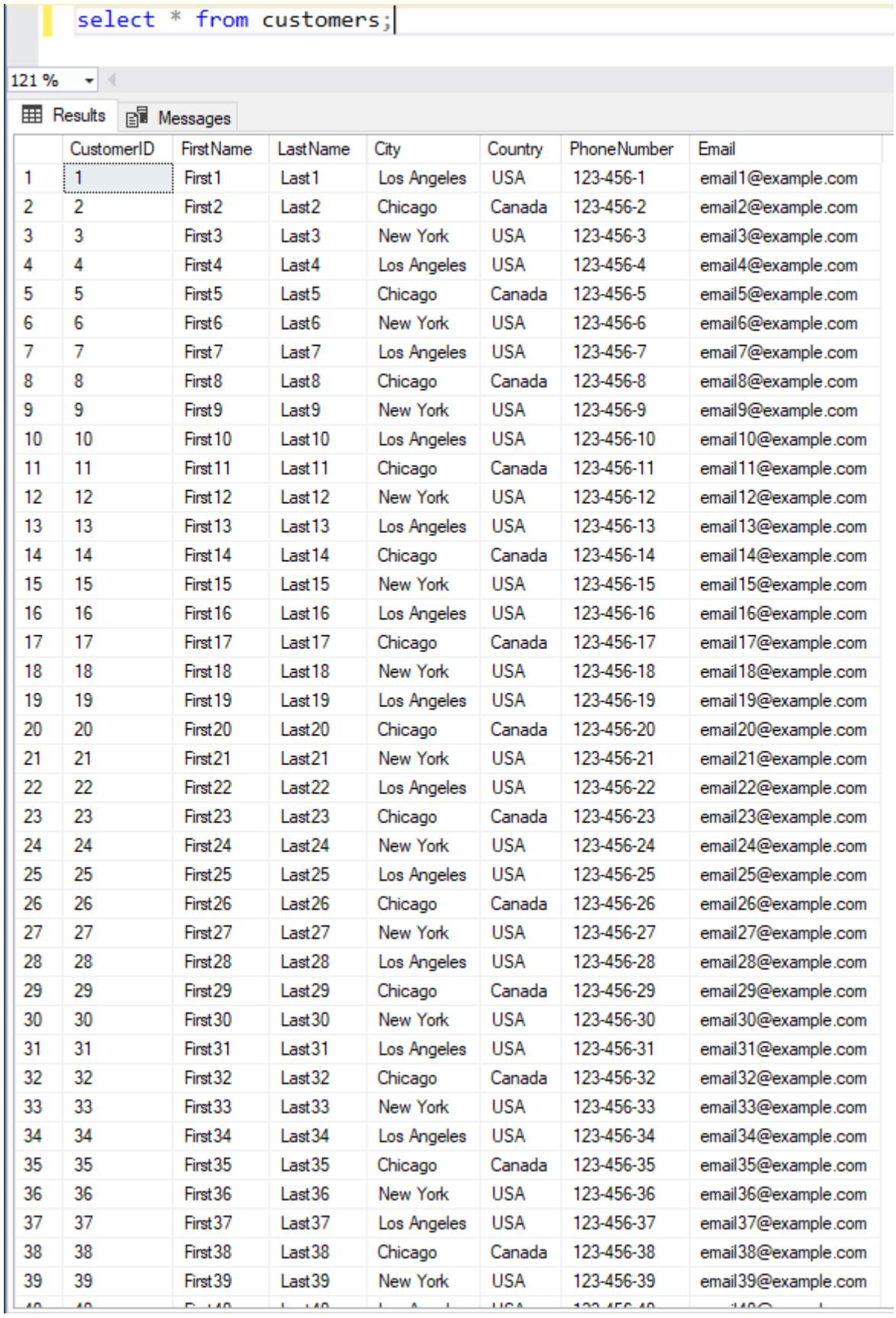
Indeks kolumnowy doskonale nadaje się do zapytań analitycznych, które wymagają szybkiego dostępu do dużej ilości danych i obliczeń agregujących. Może znacznie poprawić wydajność zapytań, szczególnie tych, w których istotna jest suma kolumny.

Eksperyment 3 - Indeksy wykorzystujące kilka atrybutów, indeksy include

Tworzymy tabelę `Customers`, która będzie przechowywać informacje o klientach, takie jak imię, nazwisko, miasto, kraj, numer telefonu i adres e-mail. Następnie wypełniamy tę tabelę 50,000 rekordami.

```
CREATE TABLE Customers (
    CustomerID INT IDENTITY(1,1),
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50),
    City NVARCHAR(100),
    Country NVARCHAR(100),
    PhoneNumber NVARCHAR(20),
    Email NVARCHAR(100)
);

DECLARE @i INT = 1;
WHILE @i <= 50000
BEGIN
    INSERT INTO Customers (FirstName, LastName, City, Country, PhoneNumber, Email)
    VALUES (
        CONCAT('First', @i),
        CONCAT('Last', @i),
        CASE
            WHEN @i % 3 = 0 THEN 'New York'
            WHEN @i % 3 = 1 THEN 'Los Angeles'
            ELSE 'Chicago'
        END,
        CASE
            WHEN @i % 3 = 0 THEN 'USA'
            WHEN @i % 3 = 1 THEN 'USA'
            ELSE 'Canada'
        END,
        CONCAT('123-456-', @i),
        CONCAT('email', @i, '@example.com')
    );
    SET @i = @i + 1;
END;
```



```
select * from customers;
```

121 %

Results Messages

	CustomerID	FirstName	LastName	City	Country	PhoneNumber	Email
1	1	First1	Last1	Los Angeles	USA	123-456-1	email1@example.com
2	2	First2	Last2	Chicago	Canada	123-456-2	email2@example.com
3	3	First3	Last3	New York	USA	123-456-3	email3@example.com
4	4	First4	Last4	Los Angeles	USA	123-456-4	email4@example.com
5	5	First5	Last5	Chicago	Canada	123-456-5	email5@example.com
6	6	First6	Last6	New York	USA	123-456-6	email6@example.com
7	7	First7	Last7	Los Angeles	USA	123-456-7	email7@example.com
8	8	First8	Last8	Chicago	Canada	123-456-8	email8@example.com
9	9	First9	Last9	New York	USA	123-456-9	email9@example.com
10	10	First10	Last10	Los Angeles	USA	123-456-10	email10@example.com
11	11	First11	Last11	Chicago	Canada	123-456-11	email11@example.com
12	12	First12	Last12	New York	USA	123-456-12	email12@example.com
13	13	First13	Last13	Los Angeles	USA	123-456-13	email13@example.com
14	14	First14	Last14	Chicago	Canada	123-456-14	email14@example.com
15	15	First15	Last15	New York	USA	123-456-15	email15@example.com
16	16	First16	Last16	Los Angeles	USA	123-456-16	email16@example.com
17	17	First17	Last17	Chicago	Canada	123-456-17	email17@example.com
18	18	First18	Last18	New York	USA	123-456-18	email18@example.com
19	19	First19	Last19	Los Angeles	USA	123-456-19	email19@example.com
20	20	First20	Last20	Chicago	Canada	123-456-20	email20@example.com
21	21	First21	Last21	New York	USA	123-456-21	email21@example.com
22	22	First22	Last22	Los Angeles	USA	123-456-22	email22@example.com
23	23	First23	Last23	Chicago	Canada	123-456-23	email23@example.com
24	24	First24	Last24	New York	USA	123-456-24	email24@example.com
25	25	First25	Last25	Los Angeles	USA	123-456-25	email25@example.com
26	26	First26	Last26	Chicago	Canada	123-456-26	email26@example.com
27	27	First27	Last27	New York	USA	123-456-27	email27@example.com
28	28	First28	Last28	Los Angeles	USA	123-456-28	email28@example.com
29	29	First29	Last29	Chicago	Canada	123-456-29	email29@example.com
30	30	First30	Last30	New York	USA	123-456-30	email30@example.com
31	31	First31	Last31	Los Angeles	USA	123-456-31	email31@example.com
32	32	First32	Last32	Chicago	Canada	123-456-32	email32@example.com
33	33	First33	Last33	New York	USA	123-456-33	email33@example.com
34	34	First34	Last34	Los Angeles	USA	123-456-34	email34@example.com
35	35	First35	Last35	Chicago	Canada	123-456-35	email35@example.com
36	36	First36	Last36	New York	USA	123-456-36	email36@example.com
37	37	First37	Last37	Los Angeles	USA	123-456-37	email37@example.com
38	38	First38	Last38	Chicago	Canada	123-456-38	email38@example.com
39	39	First39	Last39	New York	USA	123-456-39	email39@example.com

Chcemy wyszukać klientów z danego miasta i kraju.

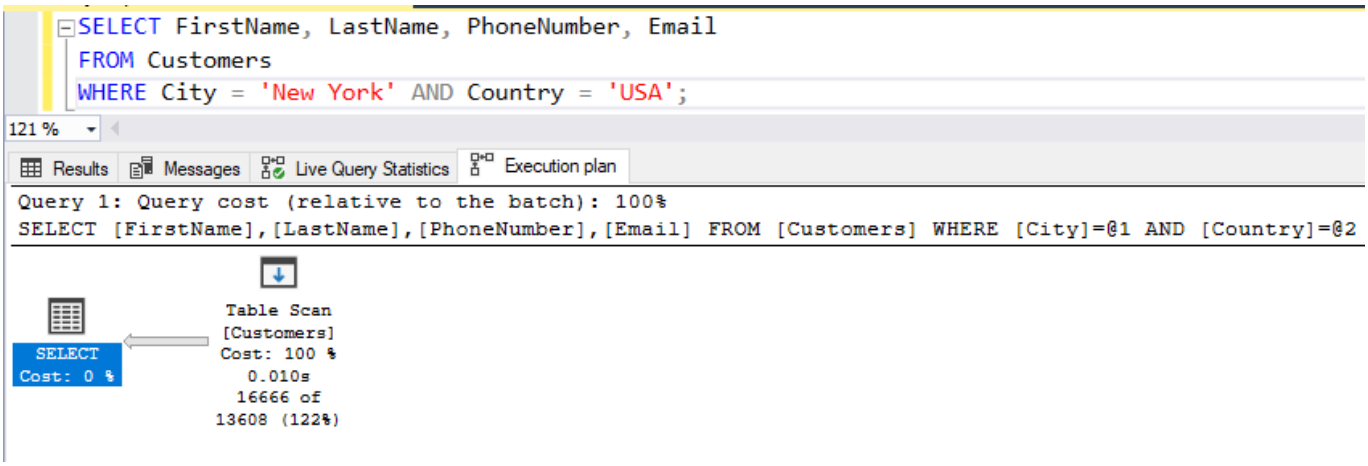
```sql
SELECT FirstName, LastName, PhoneNumber, Email
FROM Customers
WHERE City = 'New York' AND Country = 'USA';
```

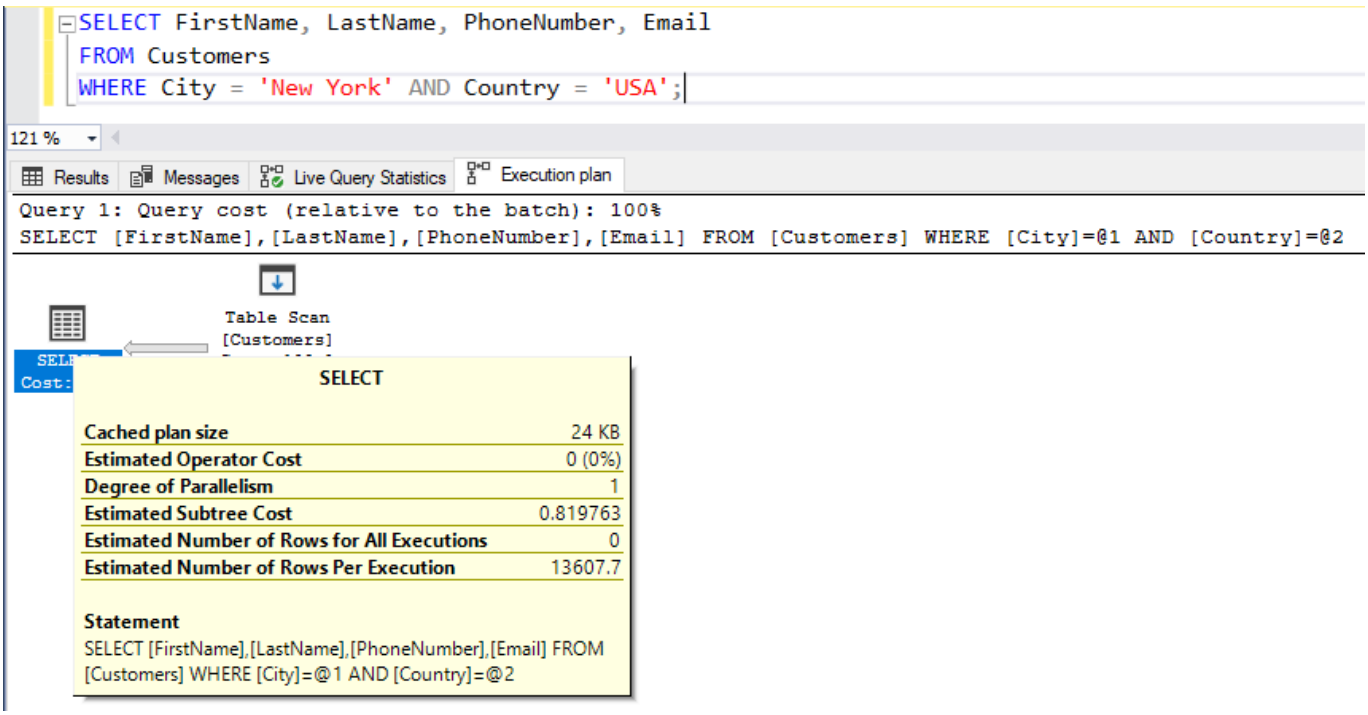
Tworzymy indeks na kolumnach City i Country oraz include na FirstName, LastName, PhoneNumber, Email.

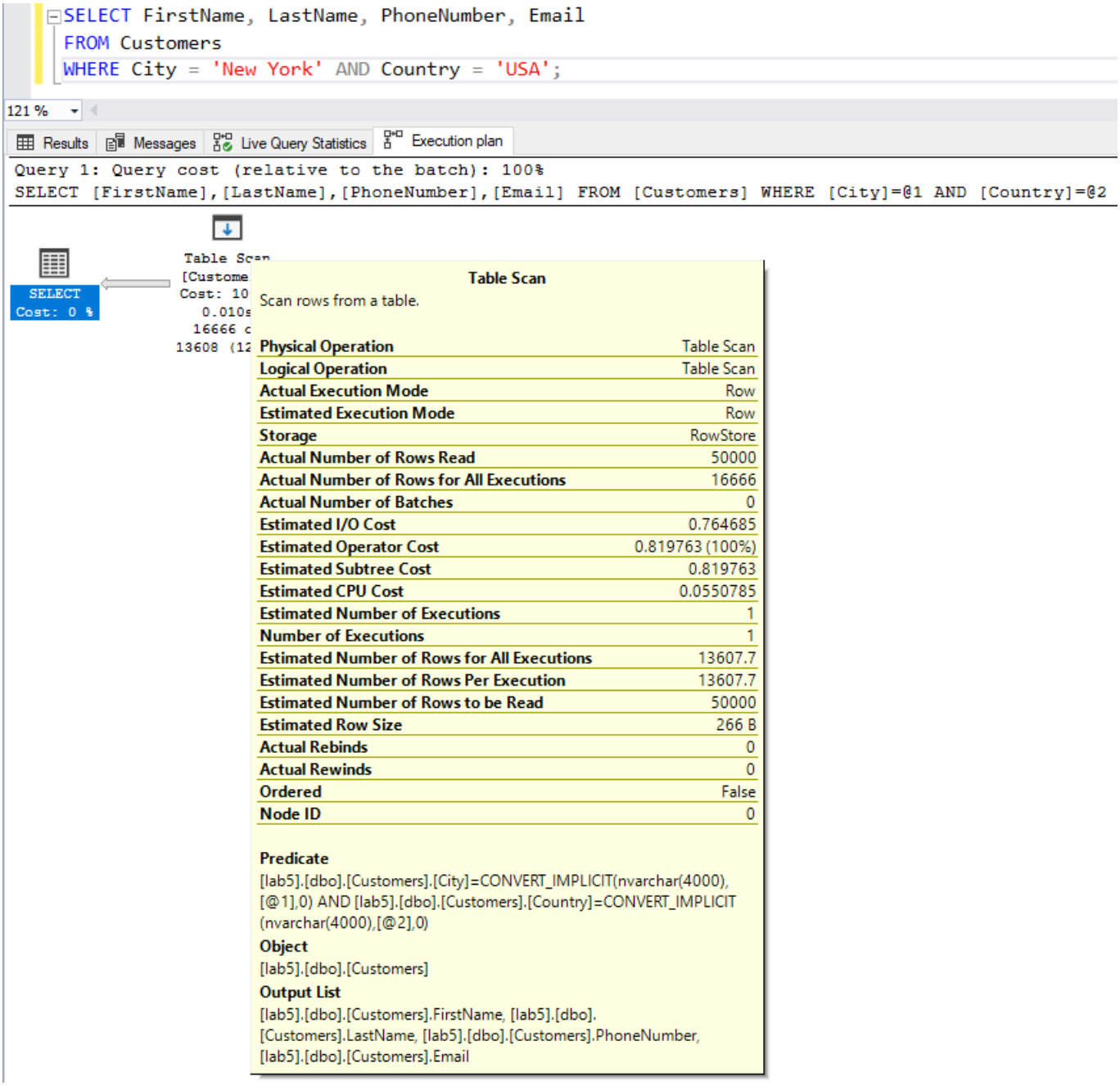
```
CREATE NONCLUSTERED INDEX IX_City_Country_Include
ON Customers (City, Country)
INCLUDE (FirstName, LastName, PhoneNumber, Email);
```

> Wyniki

Bez indeksu

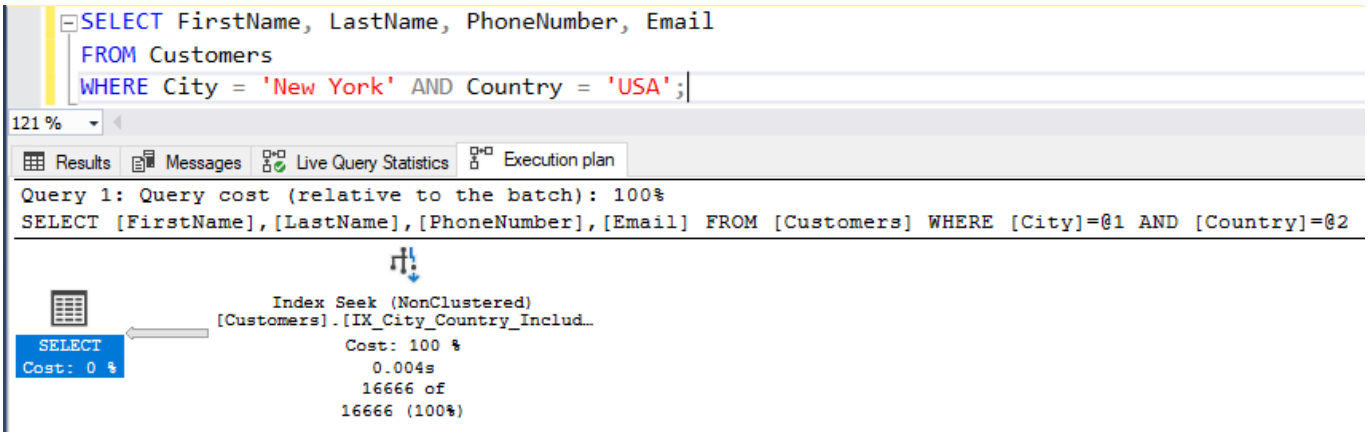


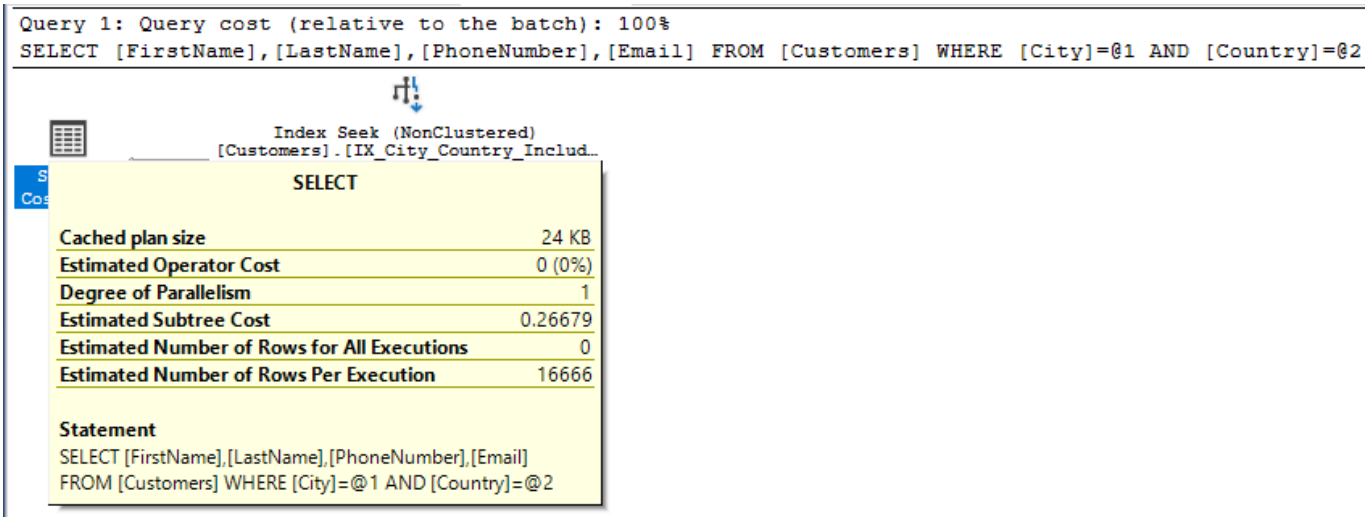




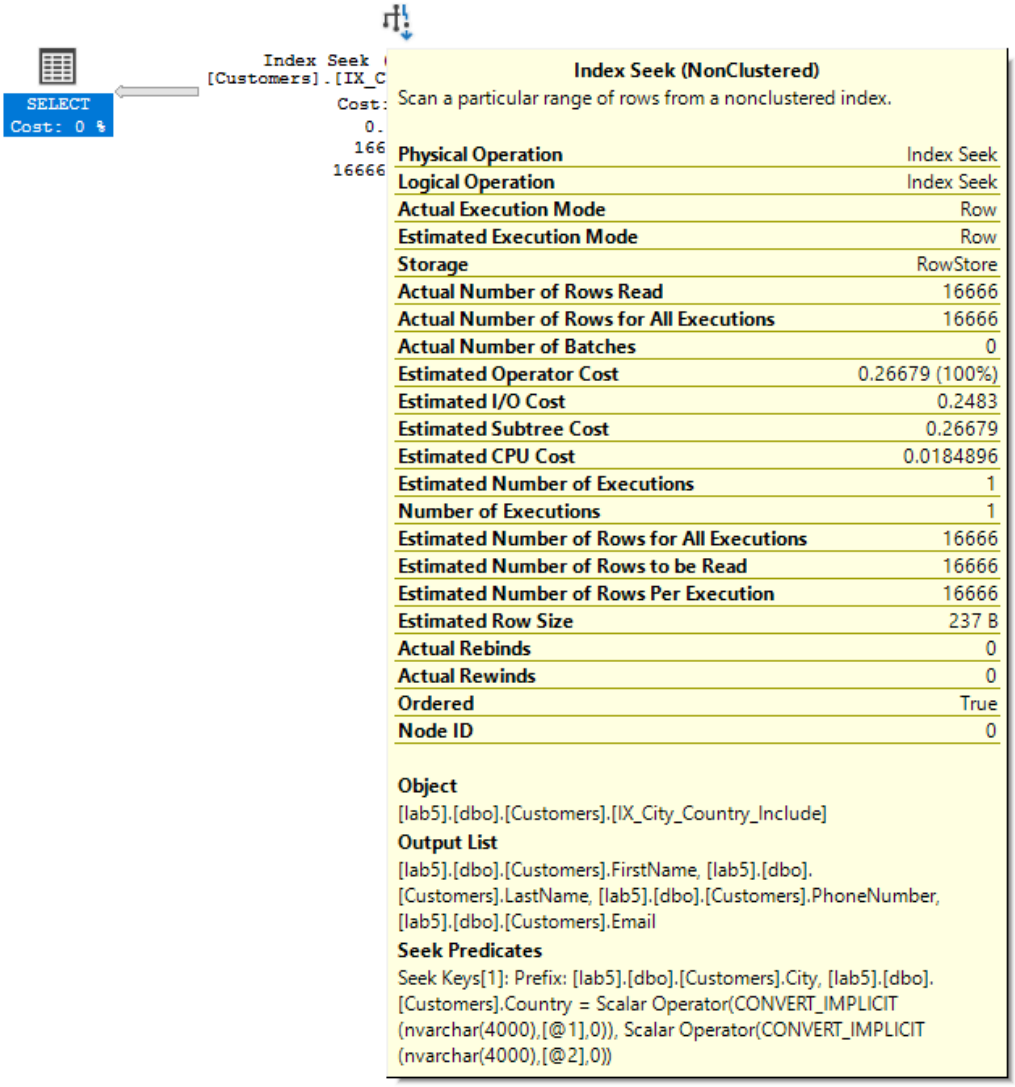
Zapytanie wykonuje pełne skanowanie tabeli, aby znaleźć klientów spełniających warunki zapytania.

Z indeksem


SELECT FirstName, LastName, PhoneNumber, Email
FROM Customers
WHERE City = 'New York' AND Country = 'USA';
121 %
Results
Messages
Live Query Statistics
Execution plan
Query 1: Query cost (relative to the batch): 100%
SELECT [FirstName],[LastName],[PhoneNumber],[Email] FROM [Customers] WHERE [City]=@1 AND [Country]=@2
SELECT
Cost: 0 %
Index Seek (NonClustered)
[Customers].[IX_City_Country_Includ...
Cost: 100 %
0.004s
16666 of
16666 (100%)


Query 1: Query cost (relative to the batch): 100%
SELECT [FirstName],[LastName],[PhoneNumber],[Email] FROM [Customers] WHERE [City]=@1 AND [Country]=@2
Index Seek (NonClustered)
[Customers].[IX_City_Country_Includ...
SELECT
Cached plan size 24 KB
Estimated Operator Cost 0 (0%)
Degree of Parallelism 1
Estimated Subtree Cost 0.26679
Estimated Number of Rows for All Executions 0
Estimated Number of Rows Per Execution 16666
Statement
SELECT [FirstName],[LastName],[PhoneNumber],[Email]
FROM [Customers] WHERE [City]=@1 AND [Country]=@2

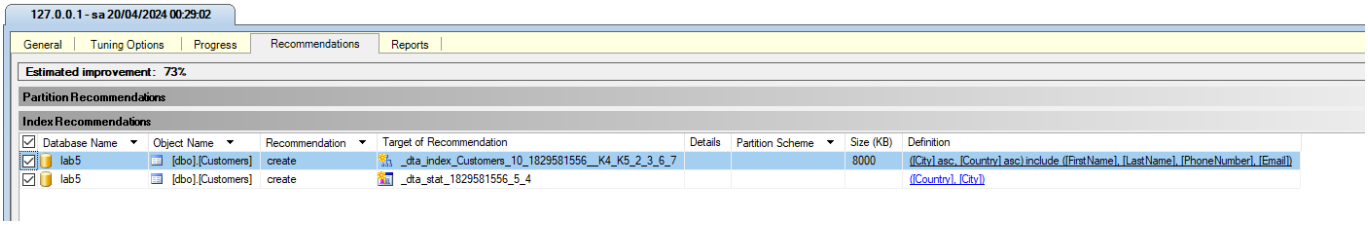
```
Query 1: Query cost (relative to the batch): 100%
SELECT [FirstName],[LastName],[PhoneNumber],[Email] FROM [Customers] WHERE [City]=@1 AND [Country]=@2
```



Dzięki indeksowi, koszt zapytania jest znacznie niższy, ponieważ SZBD może szybko zlokalizować klientów z danego miasta i kraju za pomocą indeksu, a następnie uzyskać pozostałe dane z include.

Rekomendacja Database Engine Tuning Advisor

Narzędzie sugeruje utworzenie indeksu, który został przez nas zdefiniowany, bez dodatkowych rekomendacji.



Wnioski

Indeksy wykorzystujące kilka atrybutów, wraz z include, są przydatne w przypadkach, gdy zapytania obejmują wiele kolumn w warunkach wyszukiwania oraz dodatkowych kolumnach do wyświetlenia. Dzięki nim zapytania stają się bardziej wydajne, ponieważ SZBD może szybciej dostępować się do danych spełniających kryteria zapytania.

Eksperyment 4 - Indeksownie napisów

Opis i cel

W tym eksperymencie zbadamy wydajność operacji wyszukiwania tekstu na tabeli `testString`. Tabela ta zawiera kolumnę `content`, która przechowuje duże bloki tekstu XML. Celem jest porównanie czasu i kosztu wykonania zapytań tekstowych przed i po dodaniu indeksu na kolumnie `content`.

Tworzenie tabeli i generowanie danych

Najpierw tworzymy tabelę `testString`:

```
CREATE TABLE testString (
    content VARCHAR(500),
    metadata1 VARCHAR(20),
    metadata2 VARCHAR(20),
    cnt INT
);
```

Następnie generujemy dane do tabeli:

```
DECLARE @i INT = 1;
WHILE @i <= 100000
BEGIN
    INSERT INTO testString (content, metadata1, metadata2, cnt)
    VALUES (
        '<root><block>bottom' + CAST(RAND() AS VARCHAR(10)) + '</block><within
charge="habit"><modern>rice</modern><wore>14' + CAST(RAND() AS VARCHAR(10)) +
'44283974</wore><jet solve="tribe">-421801468.1904454</jet></within>
<some>1167830737.' + CAST(RAND() AS VARCHAR(10)) + '</some></root>',
        'meta1',
        'value' + CAST(@i AS VARCHAR(10)),
        @i
    );
    SET @i = @i + 1;
END;
```

Zapytania

Będziemy wykonywać trzy różne zapytania tekstowe:

Zapytanie 1

Wyszukiwanie bloku tekstu zawierającego konkretną frazę `<block>bottom 0</block>`:

```
select content from testString where content like '%<block>bottom 0</block>%';
```

Zapytanie 2

Wyszukiwanie bloku tekstu zaczynającego się od `<root><block>bottom 0`:

```
select content from testString where content like '<root><block>bottom 0%';
```

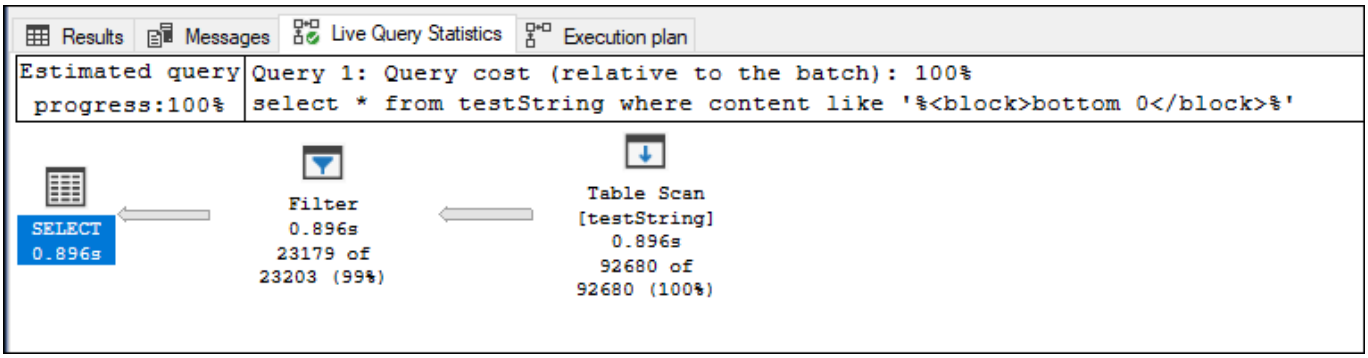
Zapytanie 3

Wyszukiwanie bloku tekstu kończącego się na `313</fruit></root>`:

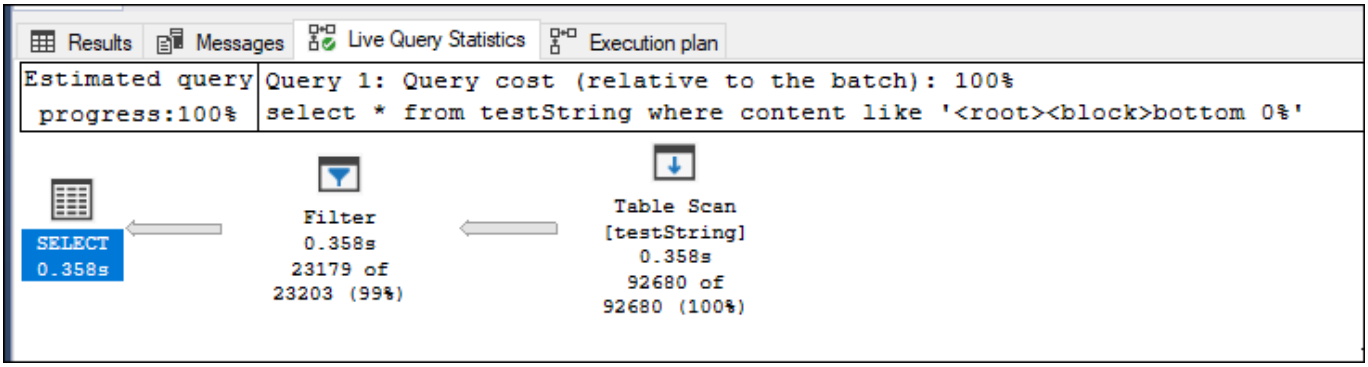
```
select content from testString where content like '%313</fruit></root>';
```

Wyniki bez indeksu

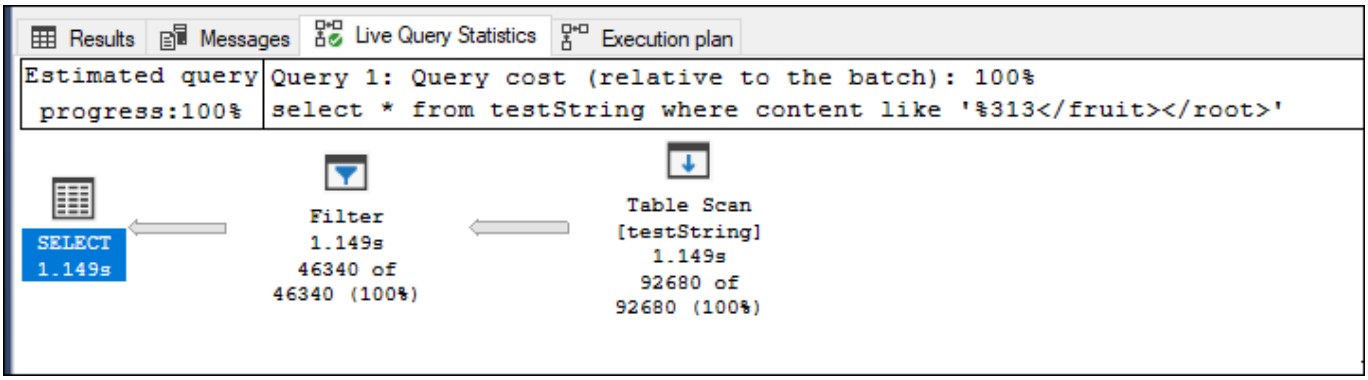
Zapytanie 1



Zapytanie 2



Zapytanie 3



Na tabeli nie ma indeksów, więc oczywiście trzeba przeskanować całą tabelę i ręcznie odfiltrować wyniki niepasujące do klauzuli `WHERE`. Można zauważeyć, że w zależności tego czy wyszukiwany jest prefix, infix czy sufix czasy wyszukiwania są zancząco różne. Jest to spodziewane, te 3 "fixy" mają różny stopień skomplikowania w znalezieniu. Prefix jest najprostszy - wystarczy sprawdzić początek napisu, sufix wymaga jeszcze przejścia do końca napisu (w zależności od implementacji może to być kosztowne lub nie), a infix wymaga przeszukania całego napisu.

Dodanie Indeksu na kolumnie `content`

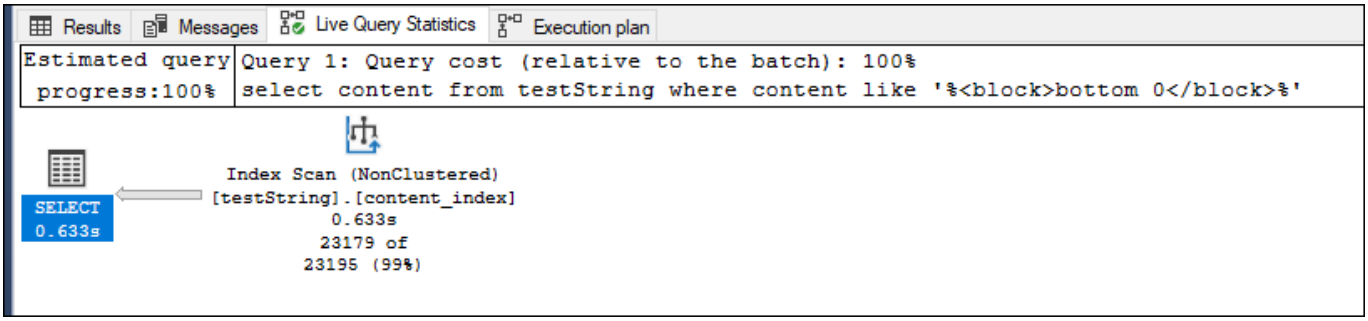
Tworzymy indeks *nonclustered* na kolumnie `content`:

```
create nonclustered index content_index on testString (content);
```

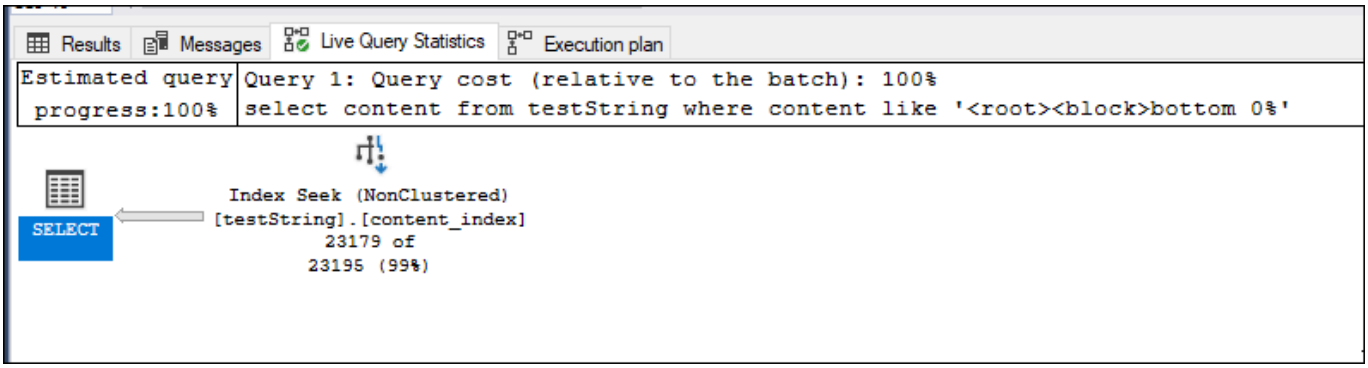
Pierwotnie kolumna `content` była typu `text`, ale okazało się MS SQL Server nie wspiera indeksowaniwa kolumn tego typu. Wspiera natomiast indeksowanie kolumn o typie `varchar`.

Wyniki zapytań z indeksem

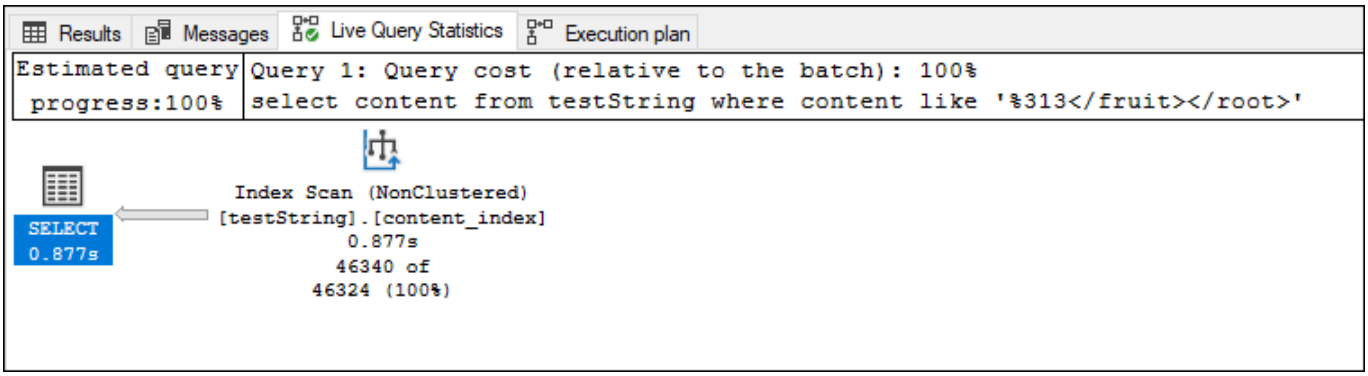
Zapytanie 1



Zapytanie 2



Zapytanie 3



Z indeksem wyszukiwanie infixu i sufixu używa `Index Scan` który jest bardzo podobny to `Table Scan`, ale używa danych zawartych w indeksie, a nie w tabeli. Zysk czasowy jest więc prawdopodobnie wynikiem ominięcia operacji I/O. Wyszukiwanie prefixu za to używa `Index Seek`, które jest efektywnym wykorzystaniem struktury drzewa. Widać tutaj, że MS SQL Server tworząc indeks `nonclustered` na polu `varchar` w żaden sposób nie optymalizuje go pod wyszukiwanie tekstowe (np. budując inny typ drzewa).

Wnioski

Dodanie indeksu na kolumnie `content` znacząco poprawia wydajność operacji wyszukiwania tekstu. Indeks umożliwia szybkie dostęp do danych, co prowadzi do znacznego skrócenia czasu wykonania zapytań i redukcji kosztów operacji. W przypadku tabel zawierających duże bloki tekstu, stosowanie indeksów może być kluczowe dla zapewnienia odpowiedniej wydajności operacji wyszukiwania tekstu.

Eksperyment 5 - Kompresja tabeli - porównanie różnych metod

Opis i cel

W tym eksperymencie przeprowadzimy analizę różnych metod kompresji danych na bazie danych z ćwiczenia 3, zawierającej 58 milionów rekordów. Założenie eksperymentu zakłada, że mamy dużą tabelę z danymi archiwalnymi, na której operacje manipulacyjne (insert, update, delete) nie będą wykonywane, a głównie będą wykonywane operacje odczytu. Celem jest zbadanie, która metoda kompresji danych jest najskuteczniejsza.

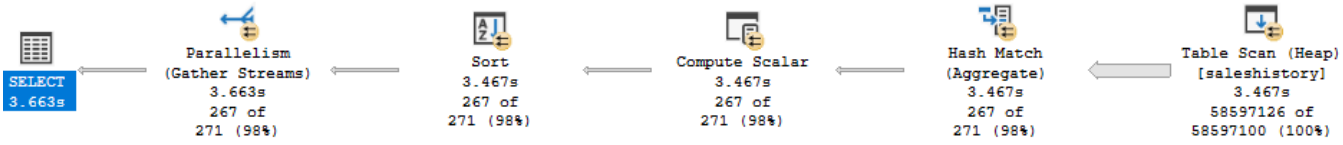
Bez kompresji

	name	rows	reserved	data	index_size	unused
1	saleshistory	58597126	6144704 KB	6144312 KB	128 KB	264 KB

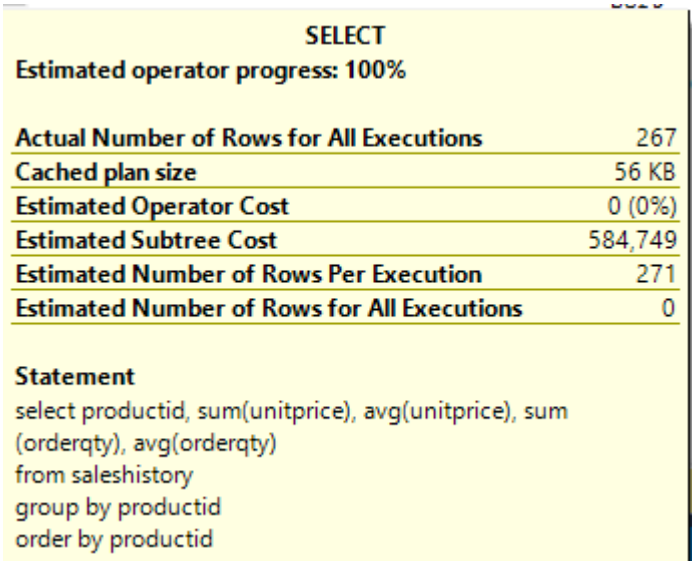
Początkowo tabela zajmuje 5.86 GB.

Spróbujmy wykonać zapytanie z zadania 3.

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



Czas wykonania zapytania wynosi 3.66s



SELECT

Estimated operator progress: 100%

Actual Number of Rows for All Executions	267
Cached plan size	56 KB
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	584,749
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
```

a jego koszt około 584.

Page compression

W pierszym kroku spróbujemy wykonać *Page Compression*. Kompresja ta składa się z trzech kroków:

- Row compression
- Prefix compression
- Dictionary compression

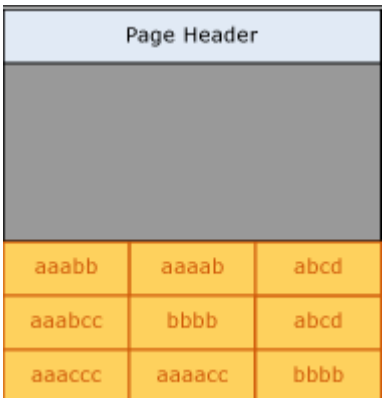
Row compression

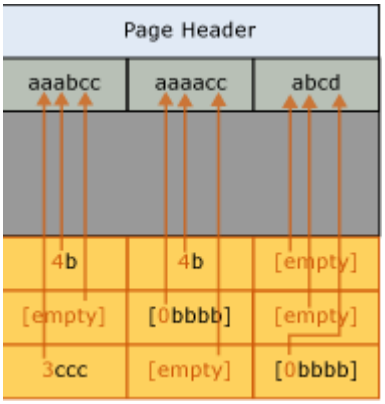
Row compression optymalizuje dane na trzy sposoby:

1. Elimunuje narzut związany z metadanymi. Są to informacje odnośnie kolumn, ich długości, offsetów.
2. Używa pól o zmiennej długości aby przechowywać wartości numeryczne oraz typy oparte o typy numeryczne
3. Powyższa metoda stosowana jest także dla stringów, np. poprzez pomijanie pustych znaków

Prefix compression

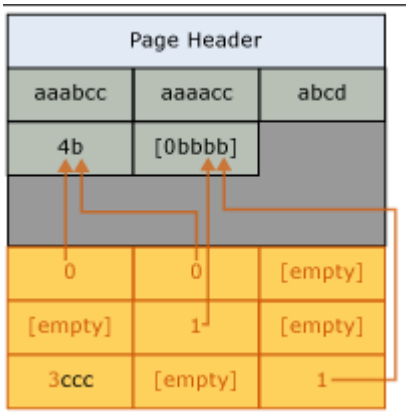
Prefix compression polega na wyznaczeniu dla każdej kolumny pewenego prefixu który powtarza się w jak największej ilości wierszy. Prefix taki jest przenoszony do nagłówka strony. Ilustracje z dokumentacji MsSql:





Dictionary compression

Dictionary Compression wykonane jest po prefix comppresion i polega na stworzeniu słównika powtarzających się wartości. W przeciwieństwie do prefix compression nie jest ona ograniczona do jednej kolumny. Ilustracja:



Wyniki

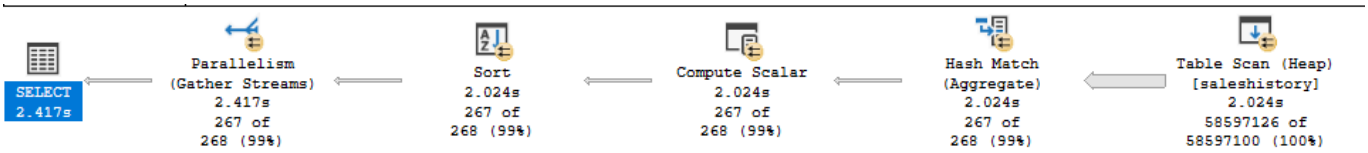
```
ALTER TABLE dbo.saleshistory REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE);
```

	name	rows	reserved	data	index_size	unused
1	saleshistory	58597126	765128 KB	764768 KB	8 KB	352 KB

Jak widać efekty tej kompresji są bardzo zadowolające. Tabela po kompresji zajmuje jedynie 0.729 GB.

Wydajność

Sprawdźmy ile czasu zajmie zapytanie z zadania 3:



Czas wykonania zapytania wynosi teraz 2.4s

SELECT

Estimated operator progress: 0%

Cached plan size	56 KB
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	584,749
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	271

Statement

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

a jego koszt pozostaje na poziomie około 584.

Jak widać w tym przypadku kompresja danych nie ma żadnego wypływu na szybkość tego zapytania.

Column store

Kompresja kolumnowa polega na zmianie sposobu przechowywania danych, gdzie kolumny są podzielone na segmenty, z których każdy jest niezależnie kompresowany. Tworzony jest również słownik kolumnowy dla unikalnych wartości w kolumnie, co pozwala na bardziej efektywne zarządzanie danymi powtarzającymi się.

Struktura kolumonowa jest szczególnie efektywna przy dużej ilości powtarzających się danych. Wyobraźmy sobie że mamy 10 wierszy z wartością *Joe*. W wierszowej reprezentacji dane przechowywane byłby w ten sposób:

```
1: Joe
2: Joe
3: Joe
.
.
.
```

W strukturze kolumnowej przechowywane są w następujący sposób:

```
Joe: 1, 2, 3, 4 ...
```

Wyniki

```
CREATE CLUSTERED COLUMNSTORE INDEX clustered_columnstore_idx ON dbo.saleshistory;
GO
```

	name	rows	reserved	data	index_size	unused
1	saleshistory	58597126	15176 KB	14768 KB	0 KB	408 KB

Jak widać zysk z tej kompresji jest ogromny. Tabela zajmuje teraz zaledwie 0.014 GB.

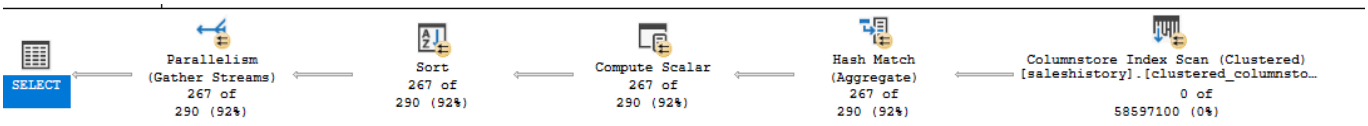
Wydajność

SELECT
Estimated operator progress: 100%

Actual Number of Rows for All Executions	267
Cached plan size	56 KB
Degree of Parallelism	12
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	6,21806
Memory Grant	39 MB
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	290

Statement

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



Koszt zapytania wynosi tylko 6. Zatem nie tylko zyskaliśmy ogromną kompresje danych, ale także uzyskaliśmy przyspieszenie dla tego konkretnego zapytania.

Column store archive

Kompresja kolumnowa archive wykorzystuje specjalny algorytm XPRESS firmy Microsoft, będący implementacją algorytmu *LZ77*.

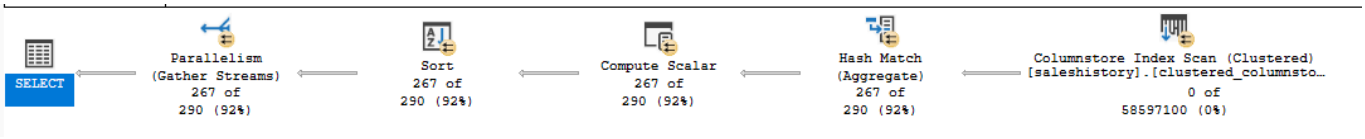
Wyniki

```
ALTER TABLE dbo.saleshistory REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION =  COLUMNSTORE_ARCHIVE);
```

	name	rows	reserved	data	index_size	unused
1	saleshistory	58597126	8392 KB	7880 KB	0 KB	512 KB

W ten sposób udało nam się zmiejszyć wielkość tabeli dwukrotnie, do 0.0075 GB.

Wydajność



SELECT

Estimated operator progress: 100%

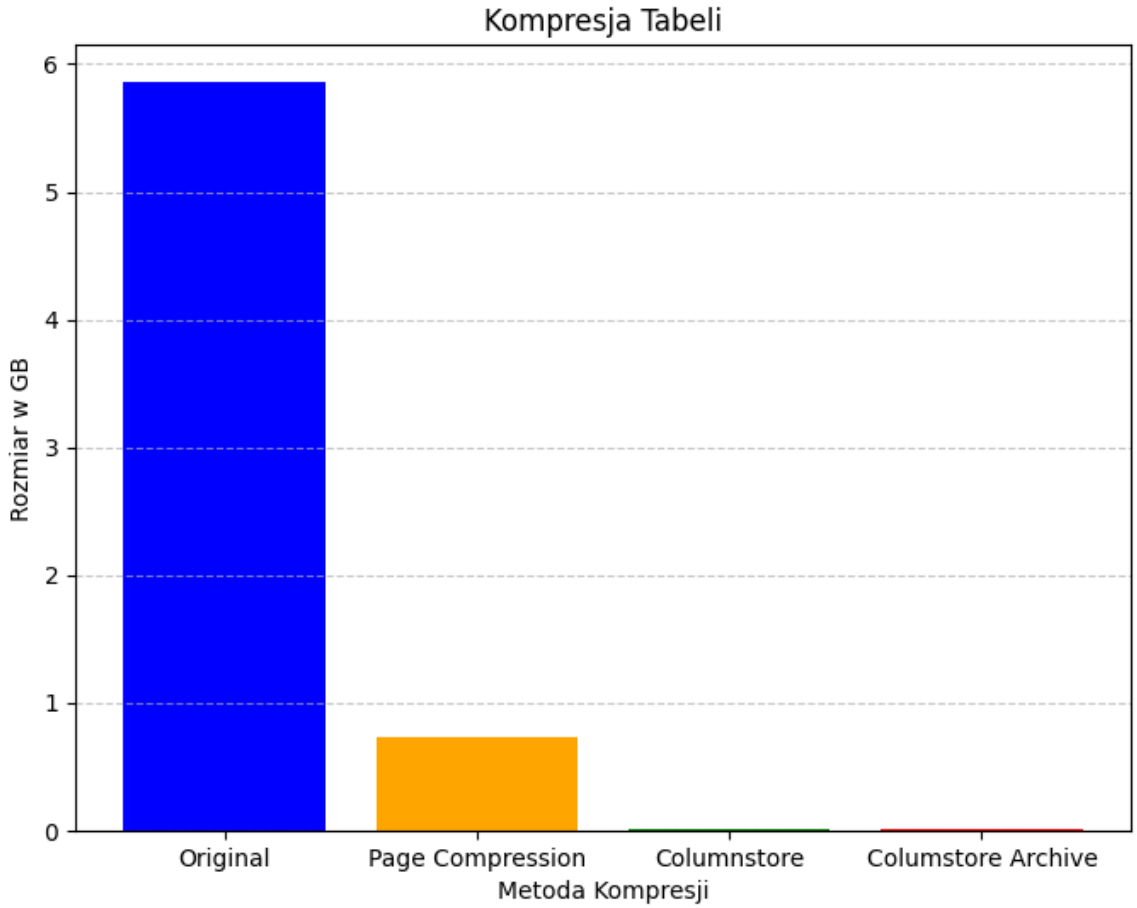
Actual Number of Rows for All Executions	267
Cached plan size	56 KB
Degree of Parallelism	12
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	6,18325
Memory Grant	39 MB
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	290

Statement

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

Plan wykonania jest taki sam jak przypadku `Column store`. Czas i koszt zapytania jest również na prawie identycznym poziomie, choć jest tutaj niewiele niższy.

Wnioski



Analiza wykazała, że nawet podstawowa *Page Compression* znacząco redukuje objętość danych, ale to *kompresja kolumnowa* daje najbardziej imponujące wyniki. Dodanie *kolumnowego indeksu* nie tylko drastycznie zmniejszyło objętość danych, ale także znacząco przyspieszyło wykonywanie zapytań. Warto zauważyć, że wykorzystanie *kolumnowego indeksu archive* jeszcze bardziej zmniejszyło wielkość danych oraz koszt wykonania zapytań, co czyni go idealnym wyborem dla archiwalnych danych. Metody te są szczególnie efektywne w przypadku dużych tabel, na których przeważają operacje odczytu.

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