Indeksy, optymalizator
Lab 6-7

Imię i nazwisko:
Jacek Budny, Mateusz Kleszcz

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
```

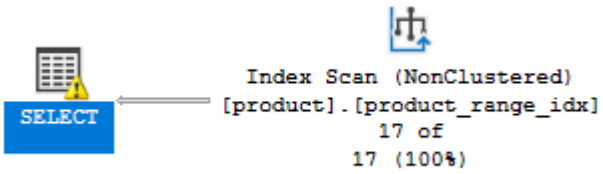
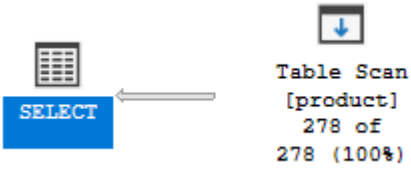
Sprawdź, czy indeks jest użyty w zapytaniu:

```
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:
> Plan dla pierwszego zapytania:
> 
>
>
> Koszt: 0.0033
> Plan dla drugiego zapytania:
> 
>
>
> Koszt: 0.0127
> Wyniki działania zapytań są zgodne z oczekiwaniami, biorąc pod uwagę warunek przedziałowy, który został określony podczas tworzenia indeksu.
> W pierwszym zaytaniu indeks został wykorzystany ponieważ warunki w klauzuli `where` mieszczą się w przedziale dla którego został stworzony ten indeks.
> W drugim zapytaniu indeks nie został wykorzystany, gdyż warunek w tym zapytaniu nie mieści się w przedziale. Zamiast tego przeprowadzony został proces skanowania tabeli.
> Z porównania kosztu wynika, że wykorzystanie indeksu znacząco zmniejsza koszt zapytania.
>
> Indeksy z warunkiem są specjalnym rodzajem indeksów , które przechowują tylko część danych z tabeli, opierając się na określonym warunku filtrowania. Ich główną zaletą jest zmniejszenie rozmiaru indeksu, dzięki przechowywaniu tylko części danych z tabeli. Zwiększa to ich efektywność w przypadku gdy chcemy mieć szybszy dostęb tylko do części danych z tabeli.

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
```

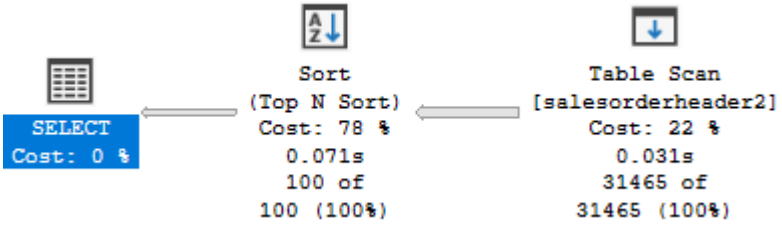
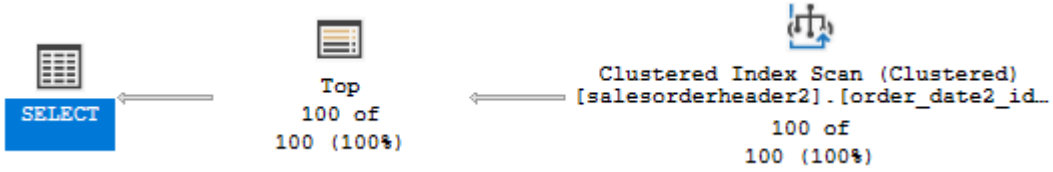
Wypisz sto pierwszych zamówień:

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

Stwórz indeks klastrujący według OrderDate:

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

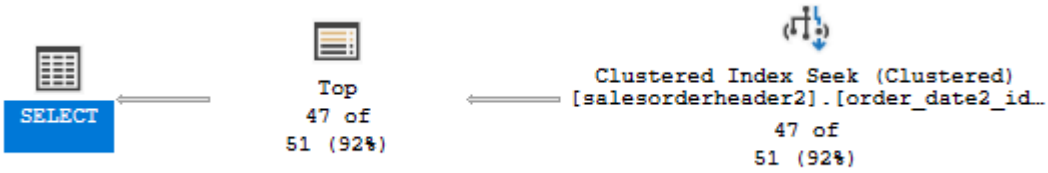
Wypisz ponownie sto pierwszych zamówień. Co się zmieniło?

> Wyniki:
> Plan dla pierwszego zapytania:
>
> 
>
>
> Koszt: 2.799
> Plan dla drugiego zapytania:
>
> 
>
>
> Koszt: 0.023

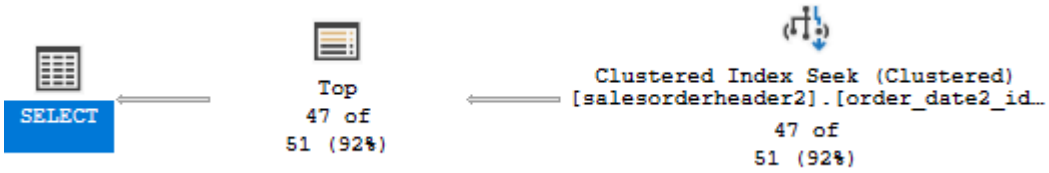
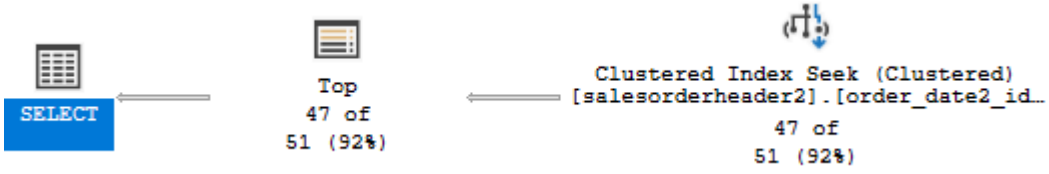
> Zamiast pełnego skanowania tabeli, silnik bazy danych skorzystał z indeksu klastrującego, co znacznie zmniejsza koszt operacji. Koszt wykonania zapytania znacząco się zmniejszył z 2.799 do 0.023.

Sprawdź zapytanie:

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

> Plan zapytania:
>
> 
>
>
> Koszt: 0.0041
> Koszt wykonania zapytania również zmniejszył się, z 0.023 do 0.0041.

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

> Plan dla zapytania z sortowaniem rosnąco:
>
> 
>
>
> Koszt: 0.0041
> Plan dla zapytania z sortowaniem malejąco:
>
> 
>
>
> Koszt: 0.0041
> Indeks klastrujący jest nadal używany, co eliminuje potrzebę dodatkowego sortowania. Koszt operacji pozostaje niski, niezależnie od kierunku sortowania.
> Indeks klastrujący na kolumnie `orderdate` umożliwia szybkie zlokalizowanie odpowiednich

> rekordów zgodnie z kryteriami wyszukiwania oraz automatyczne posortowanie wyników według tej samej kolumny. Dodatkowo, nie jest wykonywane dodatkowe sortowanie w przypadku zapytań, które wymagają sortowania wyników.

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

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
```

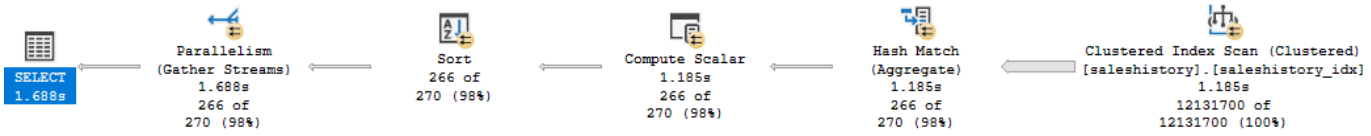
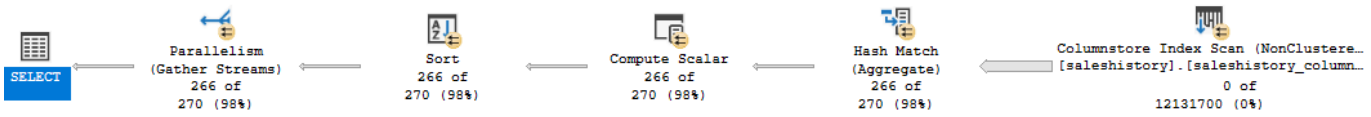
Sprawdź jak zachowa się zapytanie, które używa obecny indeks:

```
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.

> Plan dla zapytania z indeksem klastrującym :
>
> 
>
>
> Koszt: 262.3
>
> Plan dla zapytania z indeksem ColumnStore:
>
> 
>
>
> Koszt: 3.60
>
> Dla pierwszego zapytania wykorzystany został indeks klastrujący. Dla każdego wiersza została przeprowadzona operacja odczytu zawartości indeksu/
>
> Dla drugiego zapytanie wykorzystany został indeks ColumnStore. Operacja odczytu indeksu

> została przeprowadzona tylko dwa razy. Koszt zapytania srastycznie zmalał.
> Różnica polega głównie na tym, że indeks klastrujący jest bardziej efektywny w przypadku pojedynczych operacji odczytu na konkretnych wierszach, podczas gdy indeks ColumnStore jest bardziej efektywny w przypadku agregacji danych na dużą skalę.

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)

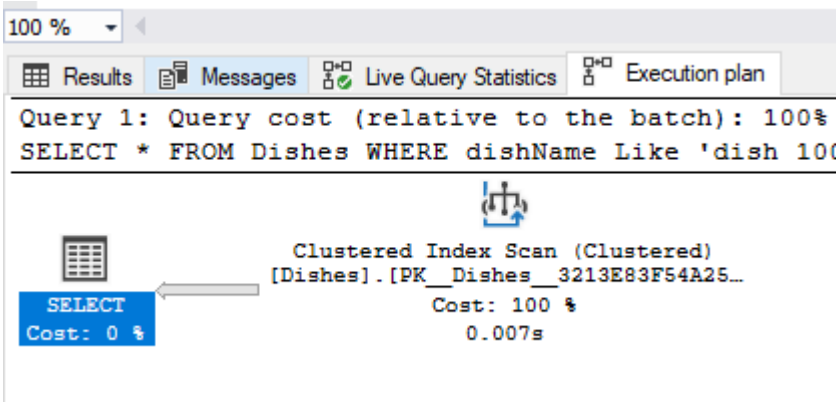
```
CREATE TABLE Dishes (
    id INT PRIMARY KEY IDENTITY(1,1),
    dishName VARCHAR(255),
    dishType VARCHAR(50),
        price INT,
    rating DECIMAL(2,1),
        vegan BIT
);
GO

declare @i int
set @i = 1
while @i <= 10000
begin
    insert into Dishes(dishName, dishType, price, rating, vegan)
    values(
                concat('dish ', @i),
                CASE FLOOR(RAND() * 4)
                        WHEN 0 THEN 'starter'
                        WHEN 1 THEN 'main dish'
                        WHEN 2 THEN 'dessert'
                        ELSE 'drink'
                END,
                ROUND(RAND() * 20 + 100, 2),
                ROUND(RAND() * 5 + 1, 1),
                ROUND(RAND() * 2, 0)
        );
    set @i = @i + 1;
end;
GO
```

> Tabela reprezentuje Menu restauracji. Każde danie to zestaw unikalnych cech - nazwy, typu (przystawka, danie główne, deser, napój), ceny (wartość z zakresu 20 - 120, z dokładnością do 2 miejsc po przecinku), oceny w skali 1 - 5 z dokładnością do jednego miejsca po przecinku, oraz czy danie jest wegańskie (1 - danie wegańskie, 0 - nie). W celu przetestowania zapytań zostało wygenerowane w sposób losowy 10000 rekordów.

> Testy zaczniemy od sprawdzenia jak wygląda przykładowe zapytanie bez dodania przez nas dodatkowych indeksów

```
SELECT * FROM Dishes WHERE dishName Like 'dish 1000'
```



> Koszt 0.0676153

> Możemy zobaczyć, że podczas generowania danych SZBD sam wygenerował klastrowy indeks. Możemy go zobaczyć za pomocą poniższej komendy.

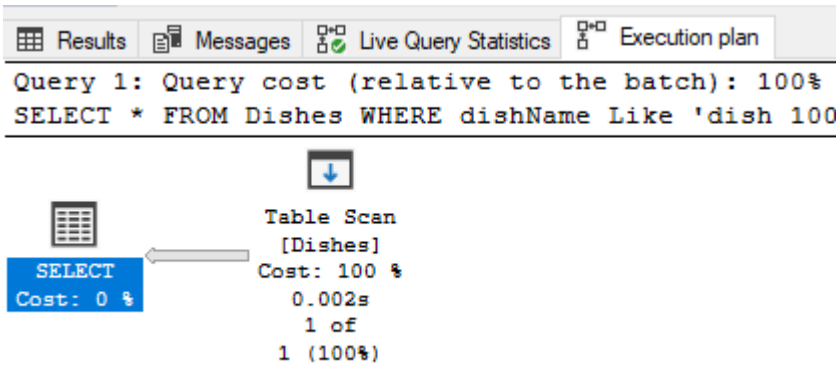
```
select * from sys.indexes
where object_id = (select object_id from sys.objects where name = 'dishes')
```

	object_id	name	index_id	type	type_desc
1	1239675464	PK__Dishes__3213E83FBD80821A	1	1	CLUSTERED

> Aby indeks nie wpływał na nasze testy usunęliśmy go. Trzeba było użyć poniższego polecenia, ze względu na nałożone na nim constrainty uniemożliwiające usunięcie za pomocą DROP INDEX.

```
ALTER TABLE dishes DROP CONSTRAINT PK__Dishes__3213E83FBD80821A
```

> Teraz możemy zauważyć, że indeks nie został użyty. Co ciekawe nie wpłynęło to na całkowity koszt zapytania.



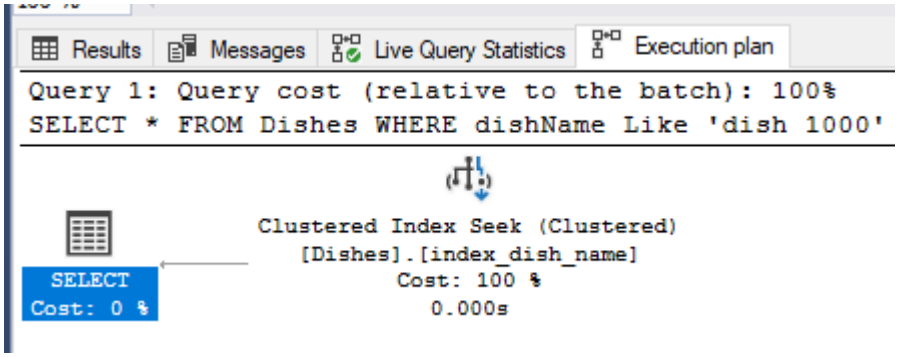
> Koszt 0.0676153

> Zacznijmy od dodania indeksu klastrowanego. Z racji tego, że możemy nałożyć tylko jeden taki indeks na całą tabelę, najwięcej sensu będzie miało nałożenie go na pole reprezentujące nazwę

> dania, z racji tego że potencjalni użytkownicy najczęściej będą wyszukiwać potrawę właśnie po nazwie.

```
CREATE CLUSTERED INDEX index_dish_name ON dishes(dishName)
```

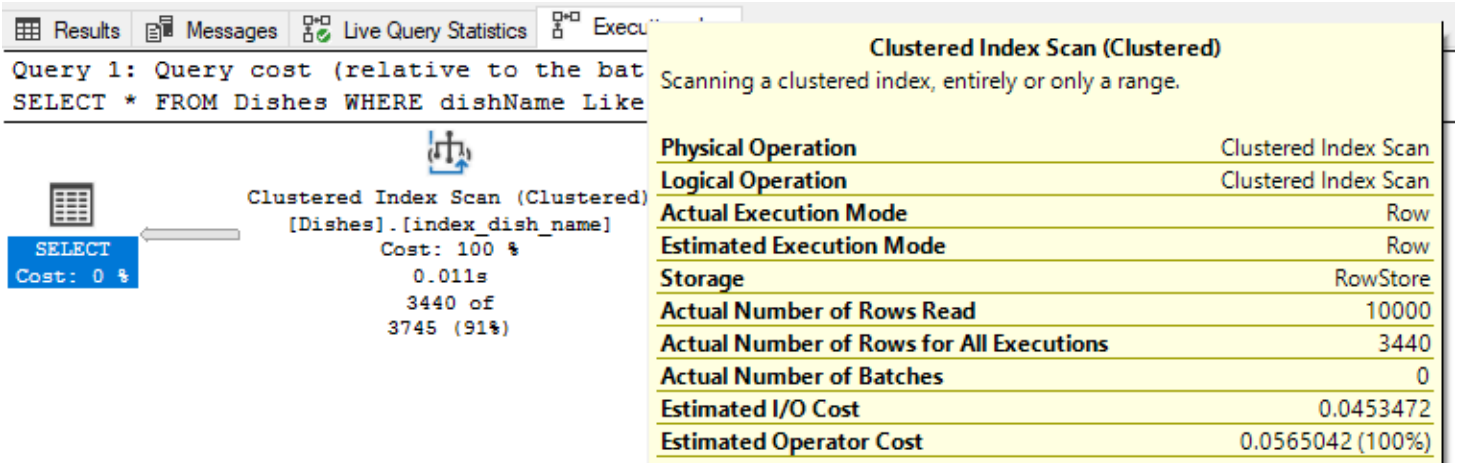
> Spróbujmy ponownie przetestować zapytanie, które wyszukiwało danie dish 1000.



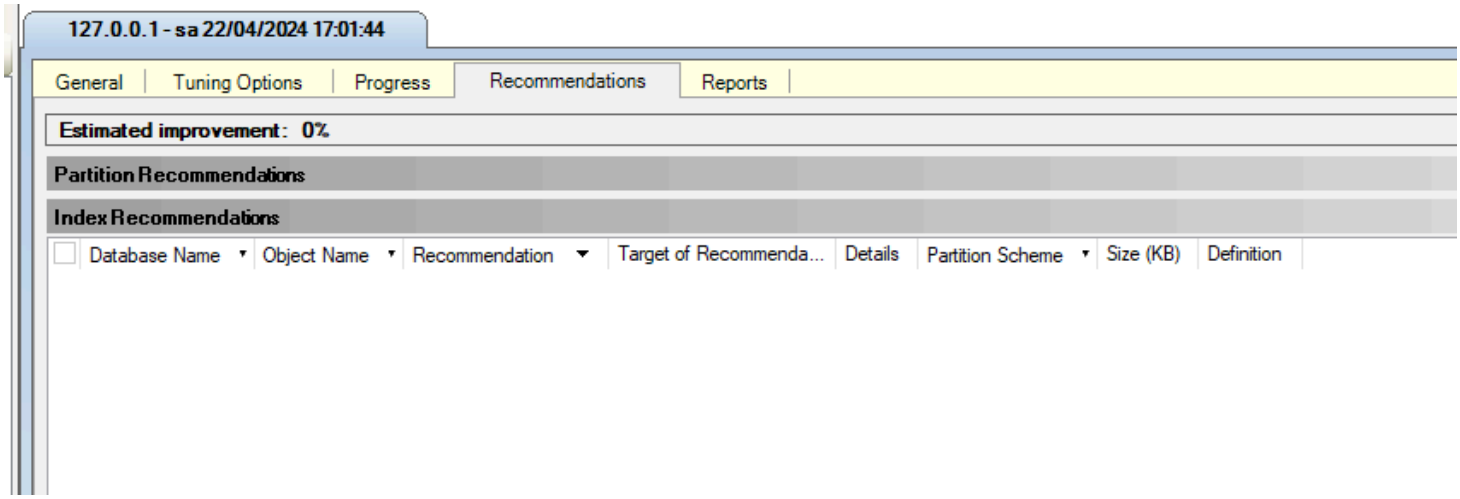
> Koszt 0.0032831

> Możemy zauważyć aż 20-krotne zmniejszenie kosztu zapytania. Spróbujmy jednak rozszerzyć zapytanie i wyszukać wszystkie dania w którego nazwie pojawia się 1. Takich rekordów jest 3440.

```
SELECT * FROM Dishes WHERE dishName Like '%1%'
```



> Możemy zauważyć, że pomimo zastosowania indeksu klastrowego podczas wykonywania tego zapytania i tak zostały odczytane wszystkie kolumny.



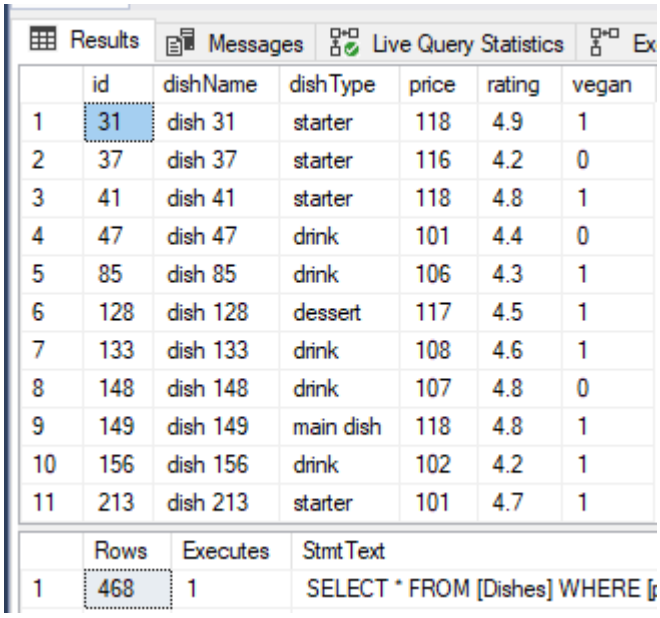
> DETA nie zwróciła żadnych rekomendacji.

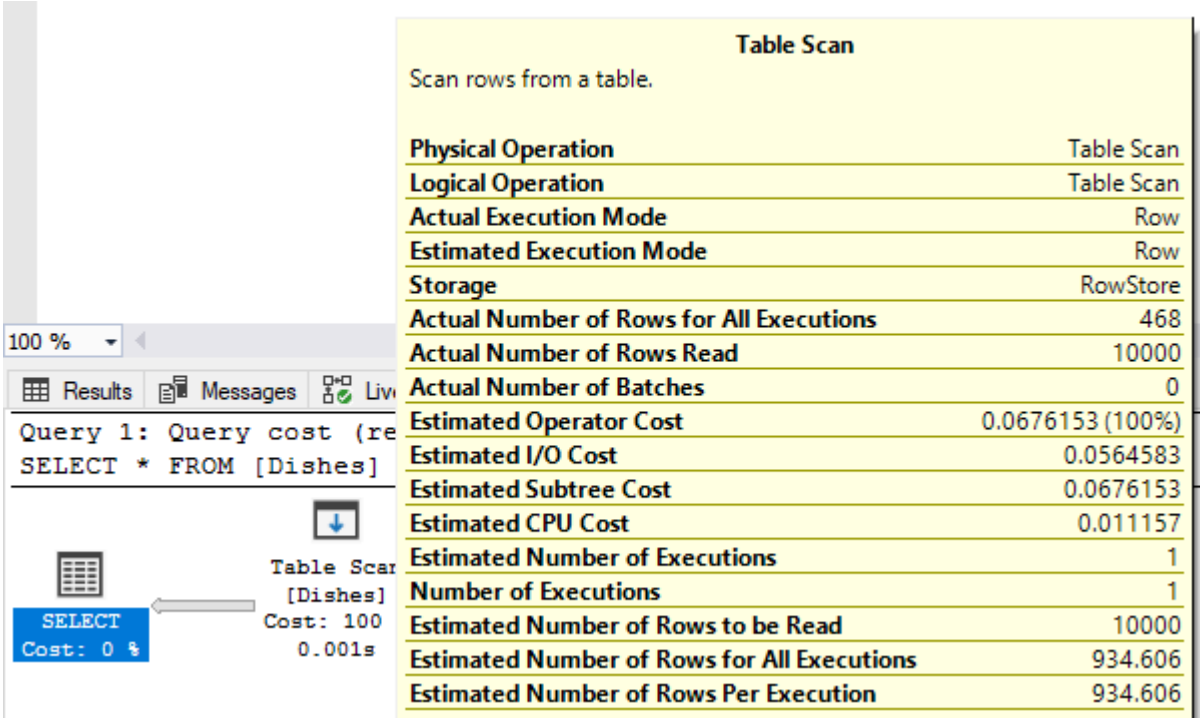
> Kolejnym dodanym przez nas indeksem będzie indeks nieklastrowy nałożony na kilka kolumn. Zdecydowaliśmy się na nałożenie go na kolumny price oraz rating. Była to według nas najbardziej prawdopodobna kombinacja, po której użytkownicy mogliby filtrować potrawy w naszej restauracji.

```
CREATE NONCLUSTERED INDEX index_dish_name ON dishes(price, rating)
```

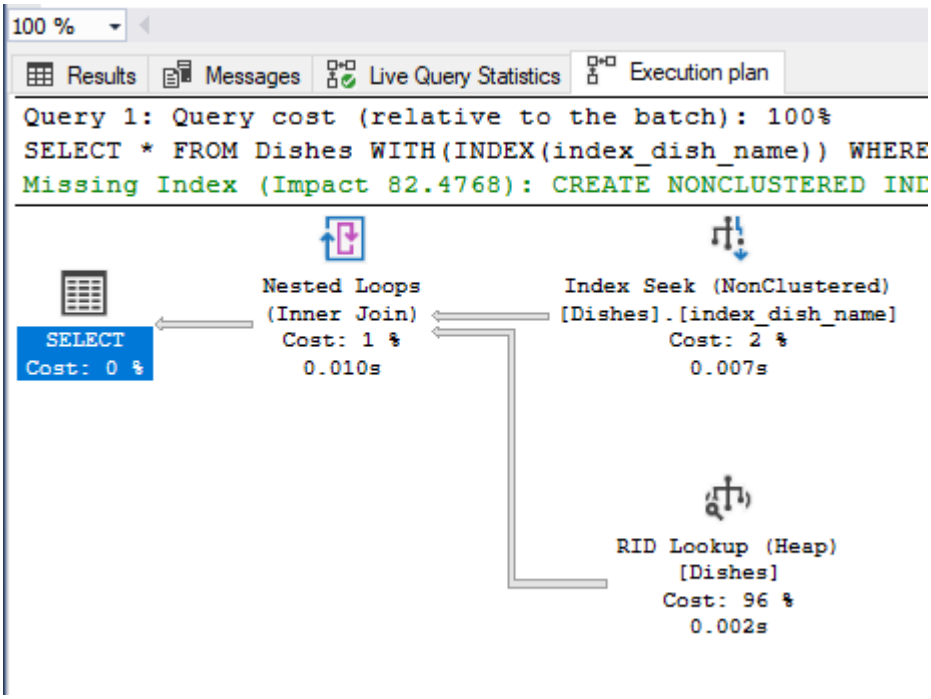
> Na początku możemy przetestować indeks za pomocą zapytania, które znajdzie wszystkie potrawy o cenie większej niż 100 i ocenach większych niż 4. Liczba zwracanych rekordów to 468.

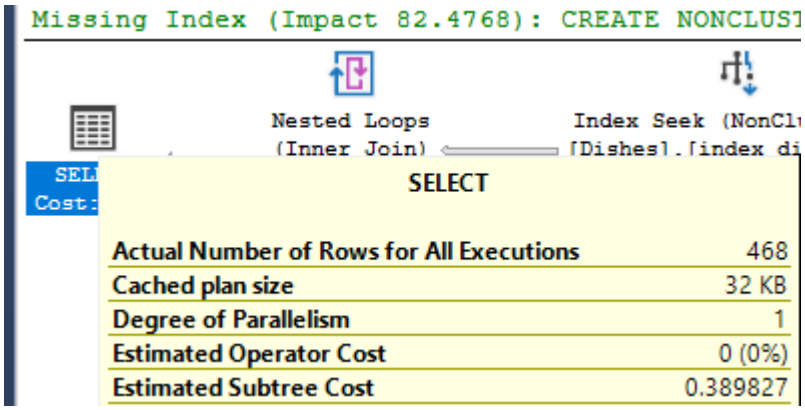
```
SELECT * FROM Dishes WHERE price > 100 AND rating > 4
```





> Możemy zobaczyć, że stworzony przez nas indeks nie został zastosowany. Wynika to prawdopodobnie ze struktury zapytania, gdzie nasz warunek jest zbyt ogólny aby zastosowanie indeksu miało sens. Spróbowaliśmy wymusić użycie tego indeksu w tym zapytaniu.

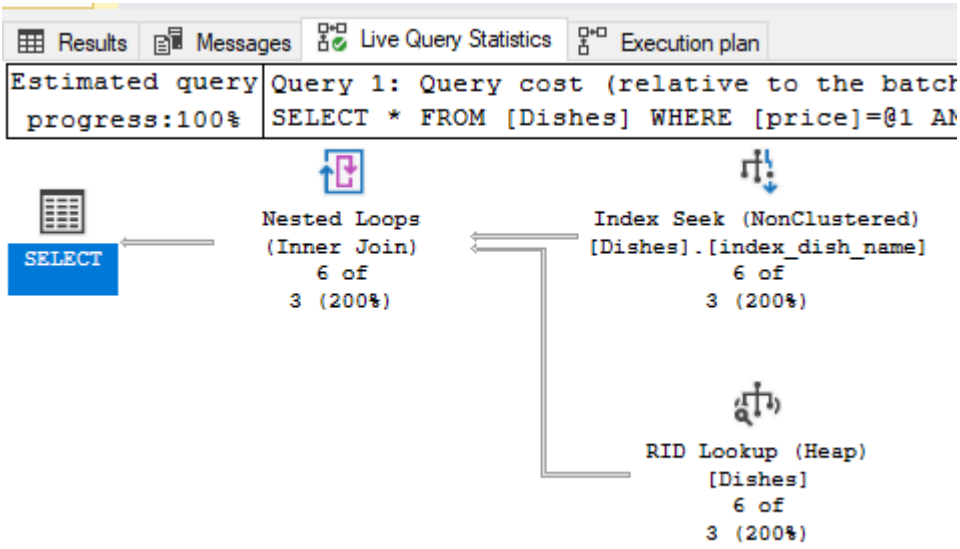




> Jak widać całkowity koszt zapytania znacząco wzrósł w przypadku wymuszenia użyciu indeksu. Spróbujmy zmienić zadanie i zastosować warunek równości a nie większości w naszym zapytaniu. Sprawia to, że liczba zwracanych rekordów to tylko 6.

Results | Messages | Live Query Statistics | Exec

	id	dishName	dishType	price	rating	vegan
1	1419	dish 1419	drink	100	4.0	0
2	1979	dish 1979	starter	100	4.0	0
3	2816	dish 2816	starter	100	4.0	0
4	3582	dish 3582	main dish	100	4.0	1
5	4396	dish 4396	starter	100	4.0	1
6	605	dish 605	drink	100	4.0	0



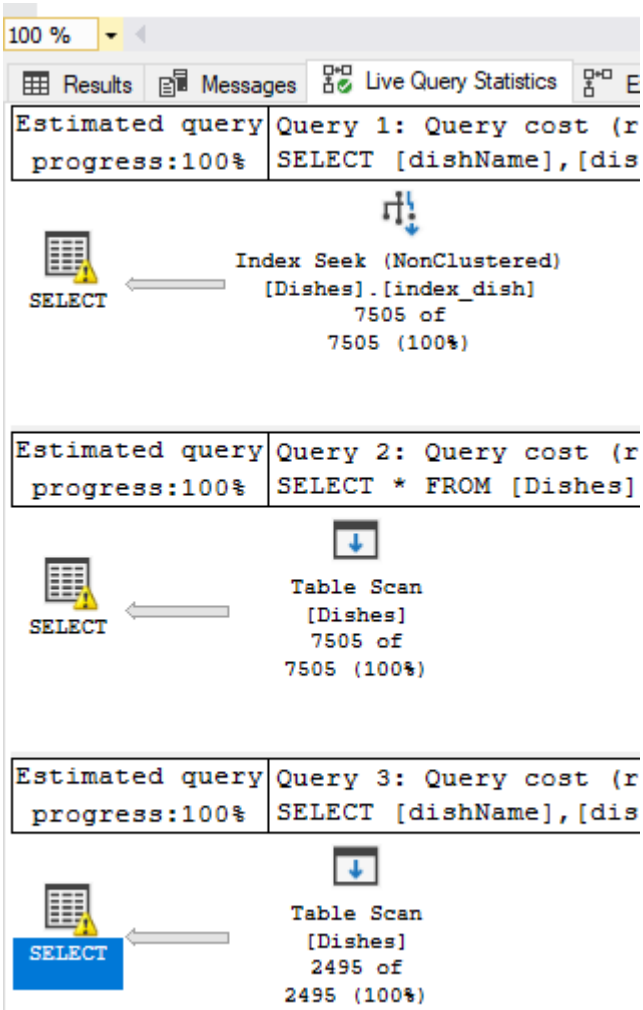
> Indeks tym razem został zastowany. DETA ponownie nie zwróciło żadnych rekomendacji, które mogłyby zoptymalizować powyższe zapytania.

> Spróbujmy dodać nieklastrowany indeks filtrowany. Będziemy chcieli znaleźć wszystkie wegańskie dania. Dodatkowo dodamy INCLUDE, aby zwracać tylko nazwę i typ w przypadku wyszukiwania wegańskiego dania.

```
CREATE NONCLUSTERED INDEX index_dish ON dishes(vegan) INCLUDE(dishName, dishType) WHERE vegan =
```

> Przetestowaliśmy 3 możliwe zapytania - w zapytaniu pierwszym spodziewamy się użycia indeksu, podczas gdy w pozostałych dwóch nie

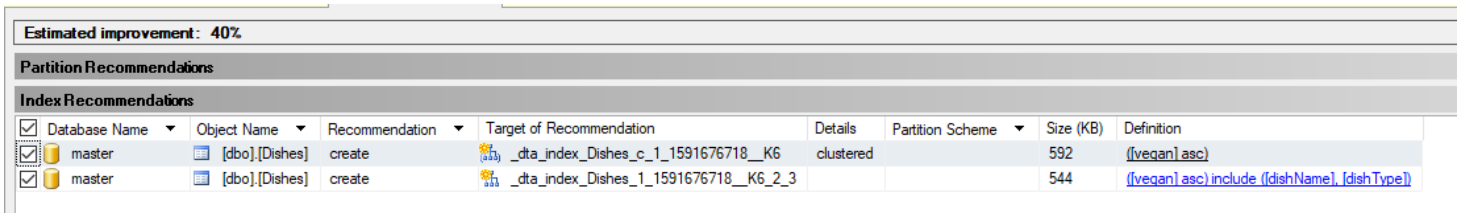
```
SELECT dishName, dishType FROM Dishes WHERE vegan = 1
SELECT * FROM Dishes WHERE vegan = 1
SELECT dishName, dishType FROM Dishes WHERE vegan = 0
```



> Koszt pierwszego zapytania wyniósł około 2.5 razy mniej niż zapytania drugiego.
>
> 0.0296982
>
> 0.0676153

> DETA tym razem zwróciło dwie rekomendacje. Po pierwsze zaproponowało dodanie dodatkowego indeksu klastrowanego dla wartości określającej czy danie jest wegańskie czy nie.

> Nie wydaje nam się jednak, żeby wartość ta była aż tak często używana, żeby wprowadzać dla niej indeks klastrowy. Druga rekomendacja polega na usunięciu filtru z indeksu.



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