Indeksy, optymalizator
Lab 5

Imię i nazwisko:

Wojciech Jasiński, Błażej Nowicki, Przemysław Węglik

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

Uruchom Microsoft SQL Managment Studio.

Stwórz swoją bazę danych o nazwie XYZ.

```
create database lab5
go

use lab5
go
```

Dokumentacja/Literatura

Obowiązkowo:

- https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes
- https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide
- https://www.simple-talk.com/sql/performance/14-sql-server-indexing-questions-you-were-too-shy-to-ask/

Materiały rozszerzające:

- https://www.sqlshack.com/sql-server-query-execution-plans-examples-select-statement/

Zadanie 1 - Indeksy klastrowane I nieklastrowane

Skopiuj tabelę `Customer` do swojej bazy danych:

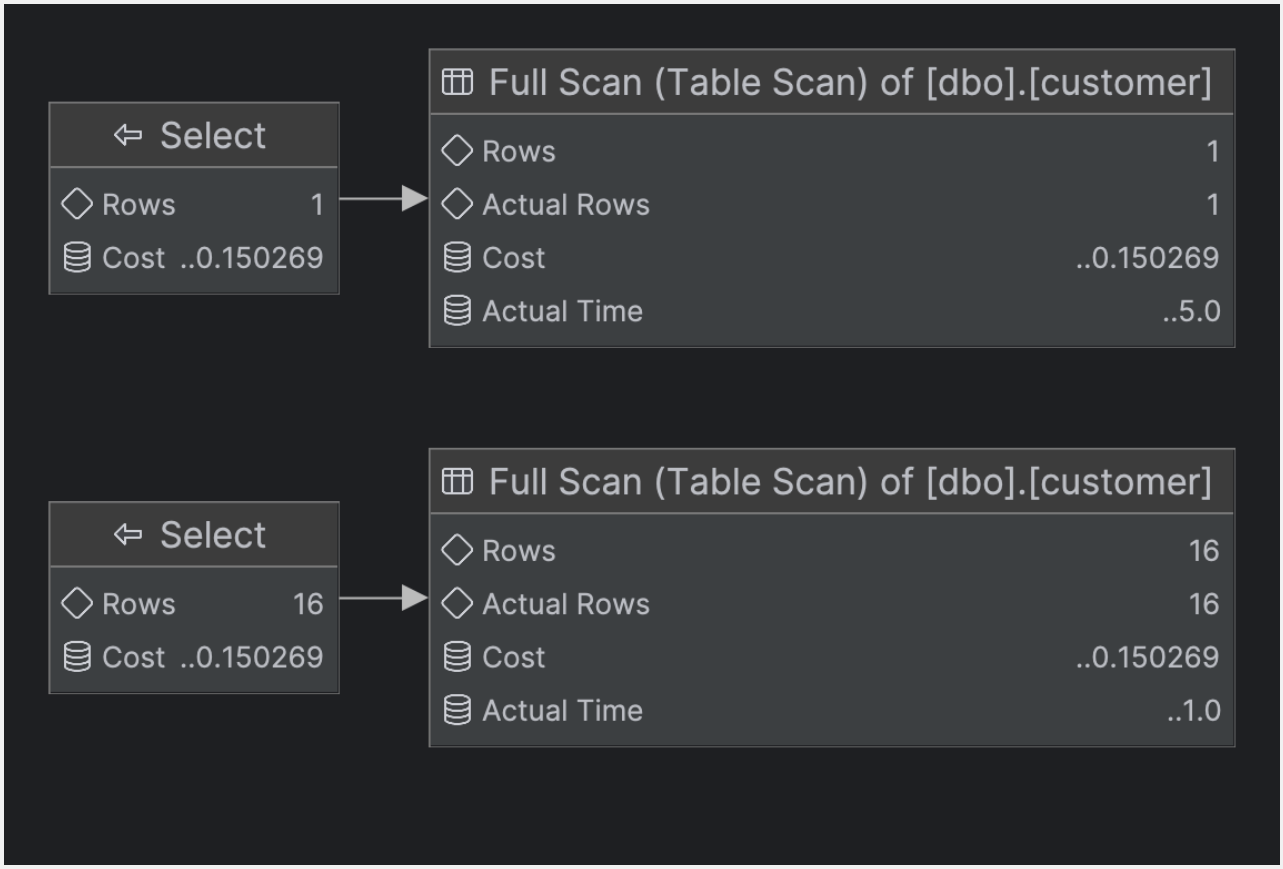
```
select * into customer from adventureworks2017.sales.customer
```

Wykonaj analizy zapytań:

```
select * from customer where storeid = 594

select * from customer where storeid between 594 and 610
```

Zanotuj czas zapytania oraz jego koszt koszt:

Wyniki:



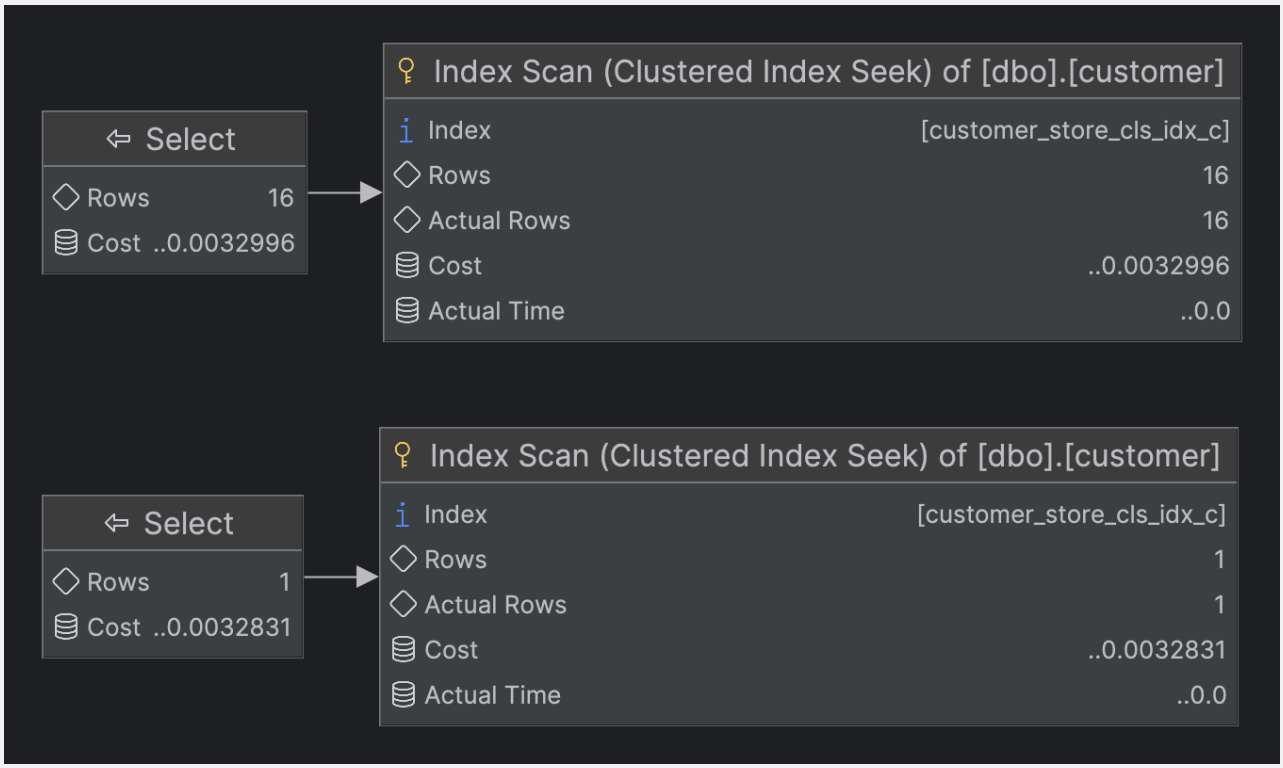
no index	= 594	between 594 and 610
time	5	1
cost	0.150269	0.150269

Czasy faktycznego wykonania są pomijalnie małe i tak.

Dodaj indeks:

```
create index customer_store_cls_idx on customer(storeid)
```

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

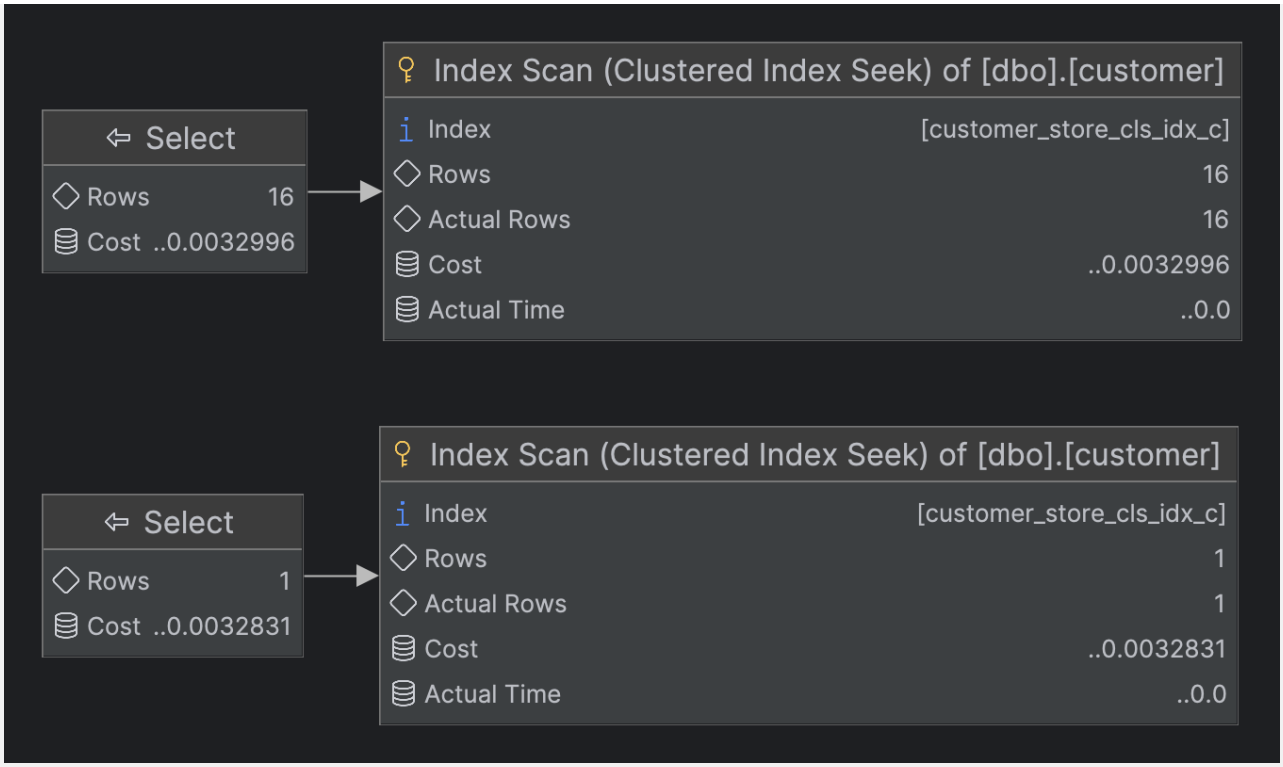
> Wyniki:
>
> 
>
>
> | nonclustered | = 594 | between 594 and 610 |
> |---|---|---|
> | time | 1 | 0 |
> | cost | 0.00657038 | 0.05088 |
>
> W obu przypadkach czas wykonania jest pomijalnie mały. Indeksowanie dużo poprawiło koszt wykonania dla filtrowania po pojedynczej wartości, dla zakresu mamy ok rząd wielkości większy koszt. Jeśli chcemy robić większe zapytania, można optymalizować dalej.

Dodaj indeks klastrowany:

```
create clustered index customer_store_cls_idx on customer(storeid)
```

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

> Wyniki:



clustered	= 594	between 594 and 610
time	0	0
cost	0.0032831	0.0032996

Przy clustered index roznica miedzy dwoma zapytaniami jest bardzo mala. Indeks klastrowany bardzo dobrze sobie radzi.

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

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

Skopiuj tabelę `Person` do swojej bazy danych:

```
select businessentityid
      ,persontype
      ,namestyle
      ,title
      ,firstname
      ,middlename
      ,lastname
      ,suffix
      ,emailpromotion
      ,rowguid
      ,modifieddate
into person
from adventureworks2017.person.person
```

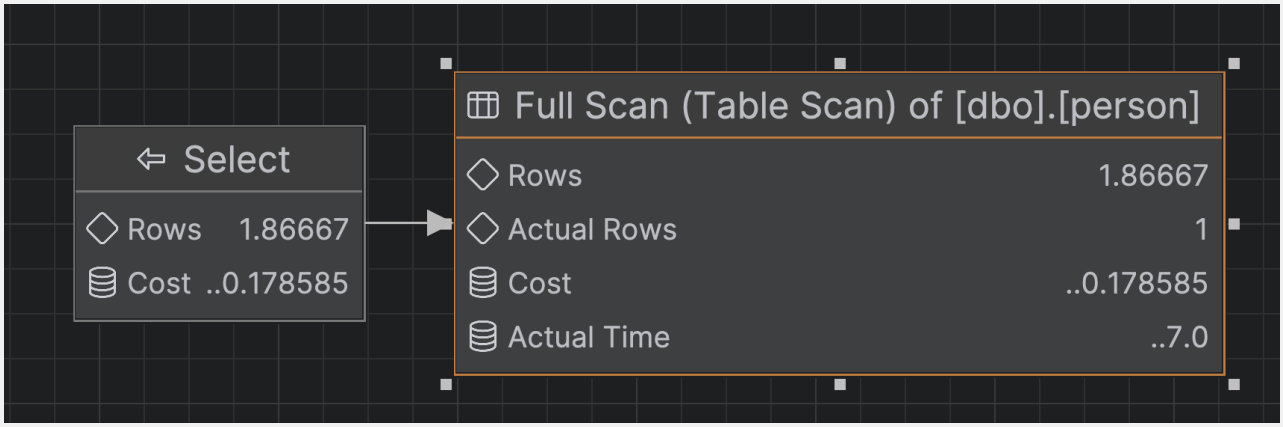
Wykonaj analizę planu dla trzech zapytań:

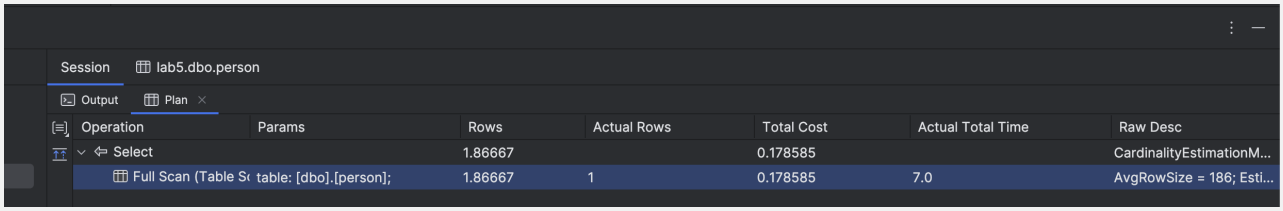
```
select * from [person] where lastname = 'Agbonile'

select * from [person] where lastname = 'Agbonile' and firstname =
'Osarumwense'

select * from [person] where firstname = 'Osarumwense'
```

Co można o nich powiedzieć?

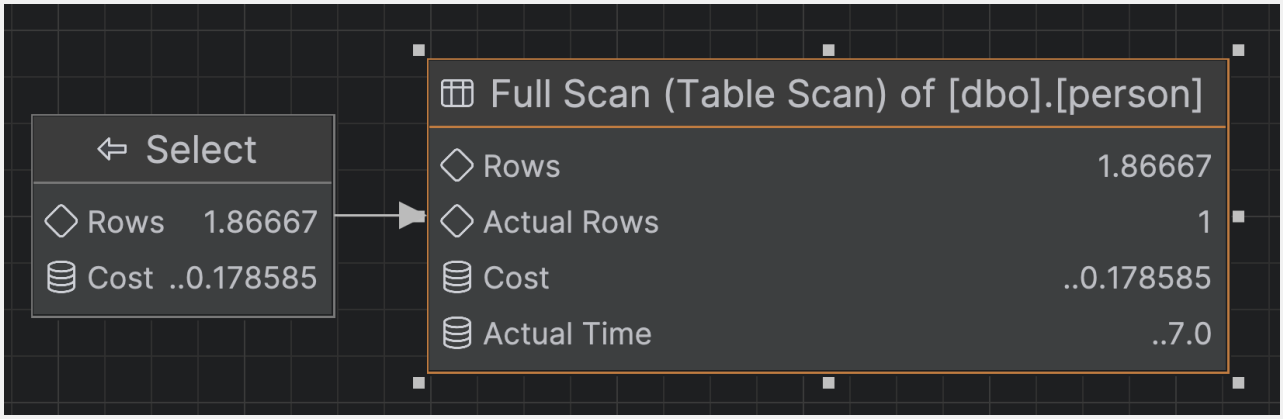
Wyniki:

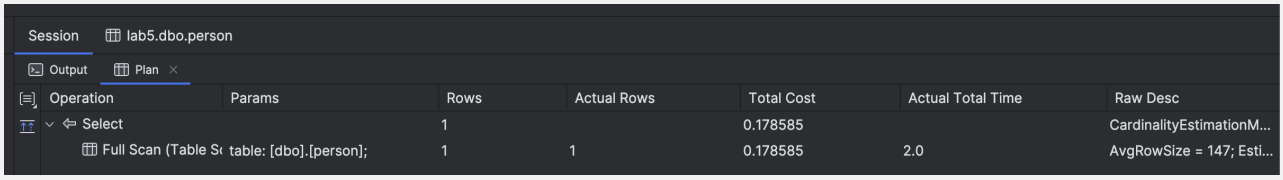
Zapytanie 1



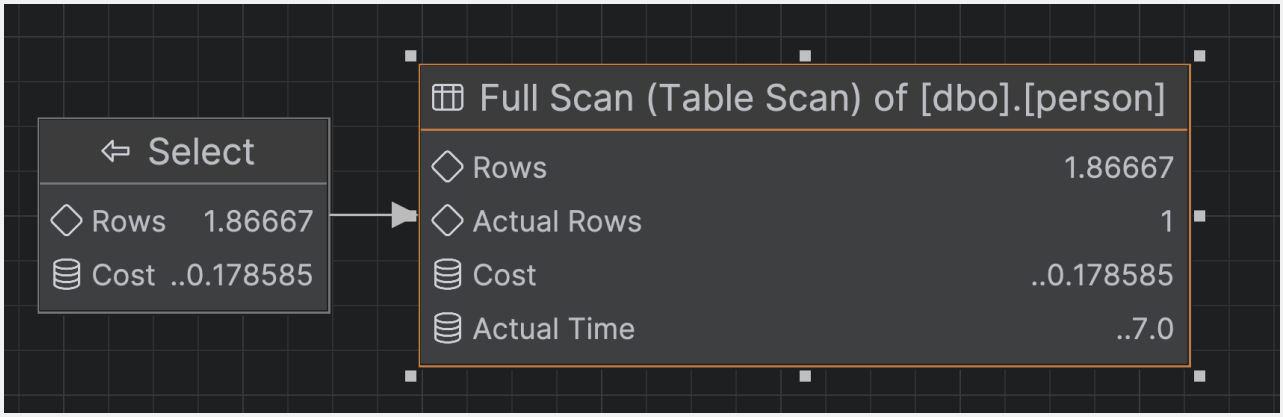


Zapytanie 2





Zapytanie 3



Session lab5.dbo.person

Output Plan

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Raw Desc
Select		13.5714		0.178585		CardinalityEstimationM...
Full Scan (Table S	table: [dbo].[person];	13.5714	1	0.178585	4.0	AvgRowSize = 186; Esti...

Plany zapytań są identyczne - mają ten sam koszt. Plany zapytań to po prostu full table scan.

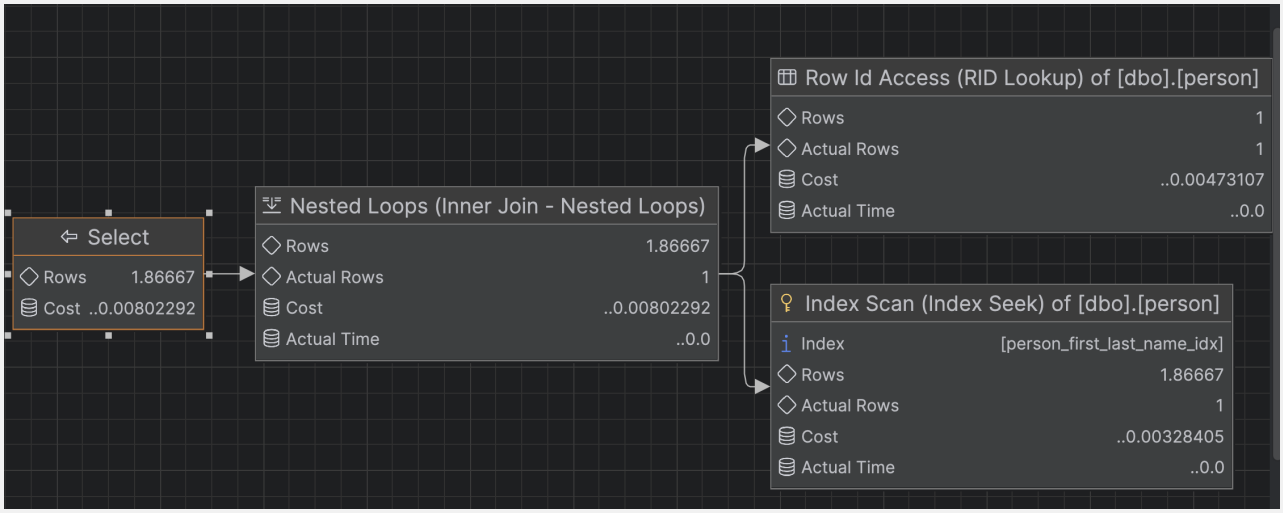
Przygotuj indeks obejmujący te zapytania:

```
create index person_first_last_name_idx
on person(lastname, firstname)
```

Sprawdź plan zapytania. Co się zmieniło?

Wyniki:

Zapytanie 1

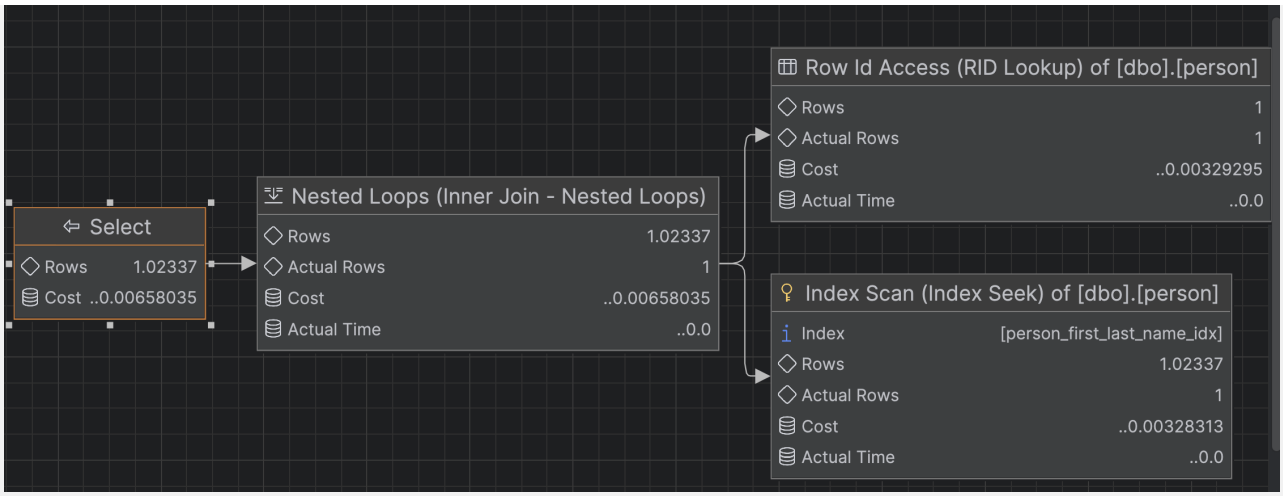


Session lab5.dbo.person

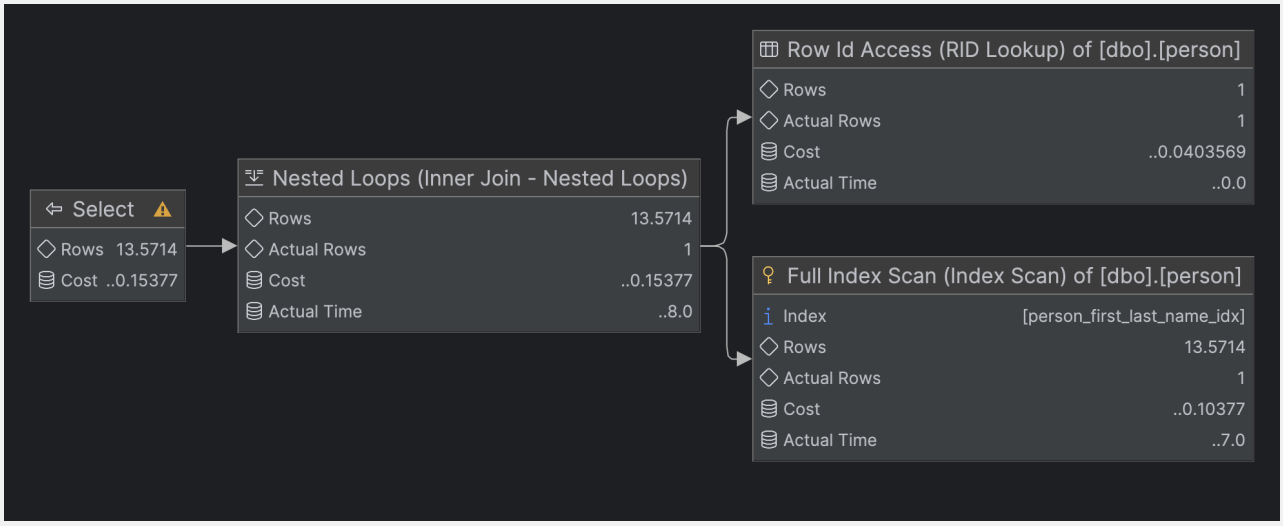
Output Plan

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Raw Desc
Select		1.86667		0.00802292		CardinalityEstimat...
Nested Loops (I		1.86667	1	0.00802292	0.0	AvgRowSize = 18...
Index Scan (I	table: [dbo].[person]; index: [person_first_last_name_idx];	1.86667	1	0.00328405	0.0	AvgRowSize = 82...
Row Id Acce	table: [dbo].[person];	1	1	0.00473107	0.0	AvgRowSize = 12...

Zapytanie 2



Zapytanie 3



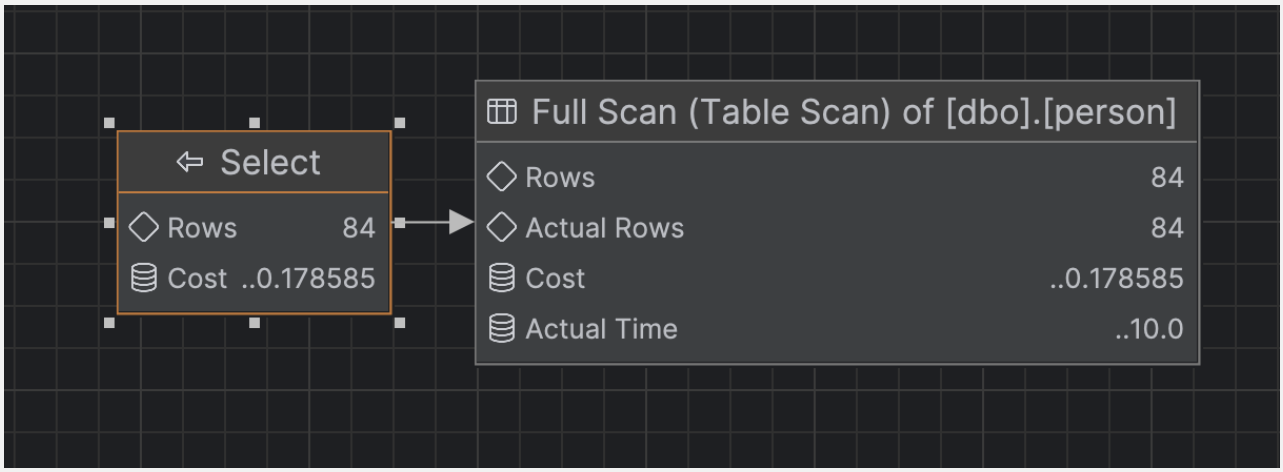
Koszty zapytań 1 i 2 poszły znacząco w dół. Koszt zapytania 3, gdzie filtrujemy po imieniu jest cały czas wysoki - tak określiliśmy kolejnosć kolumn po których indeksujemy, że indeks w tym przypadku dużo nie pomaga. Warto zauważyć, że indeks `person(lastname, firstname)` jest najbardziej efektywny dla filtrowania po lastname i firstname naraz, co wykonujemy w zapytaniu 2. Koszt jest najmniejszy spośród tych zapytań.

Przeprowadź ponownie analizę zapytań tym razem dla parametrów: `FirstName = 'Angela' LastName = 'Price'`. (Trzy zapytania, różna kombinacja parametrów).

Czym różni się ten plan od zapytania o `'Osarumwense Agbonile'` . Dlaczego tak jest?

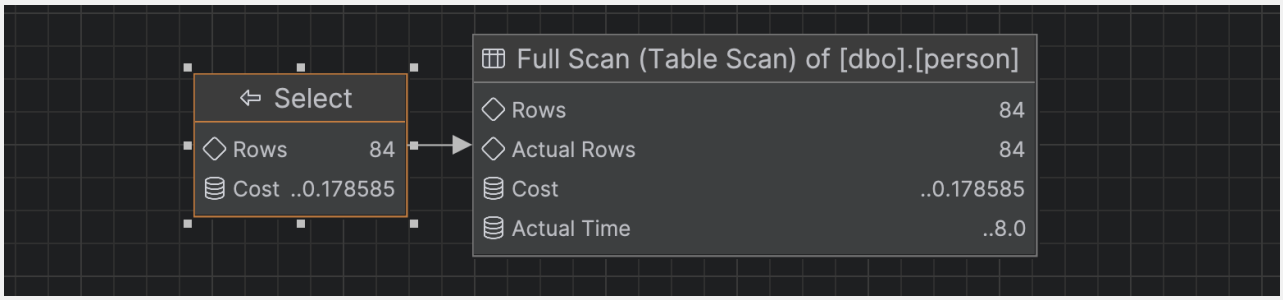
Wyniki:

Wcześniej mieliśmy tylko jeden wynik dla trzech róznych zapytań. Teraz mamy 50 wyników dla imienia Angela, 84 dla nazwiska Price i jedną osobę na przecieciu tych zbiorów.

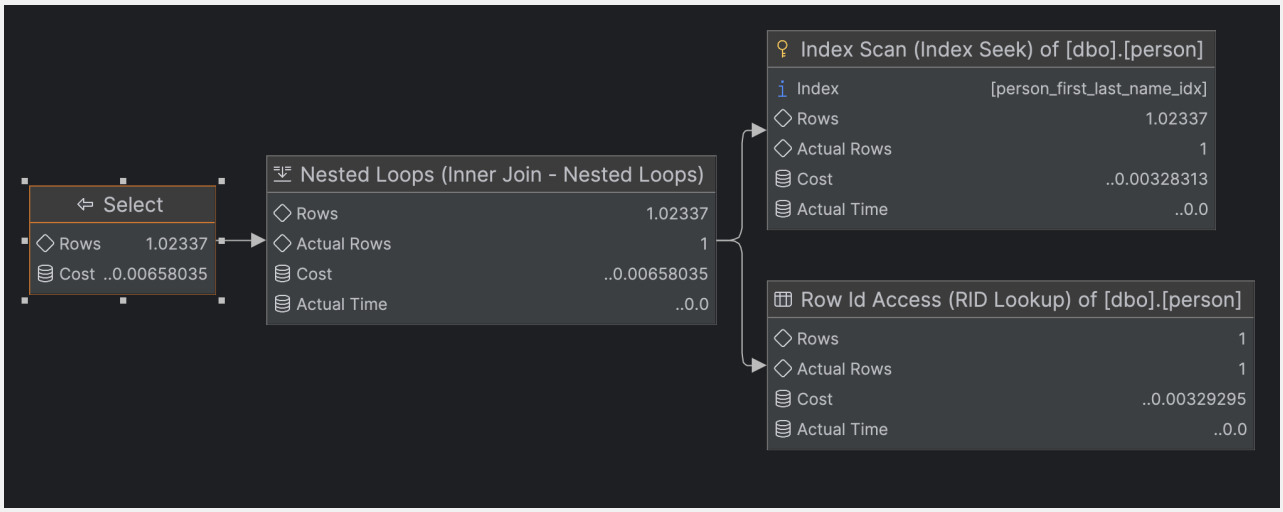


Wszystkie zapytania bez założonego indeksu mają identyczny koszt, bo robimy full table scan.

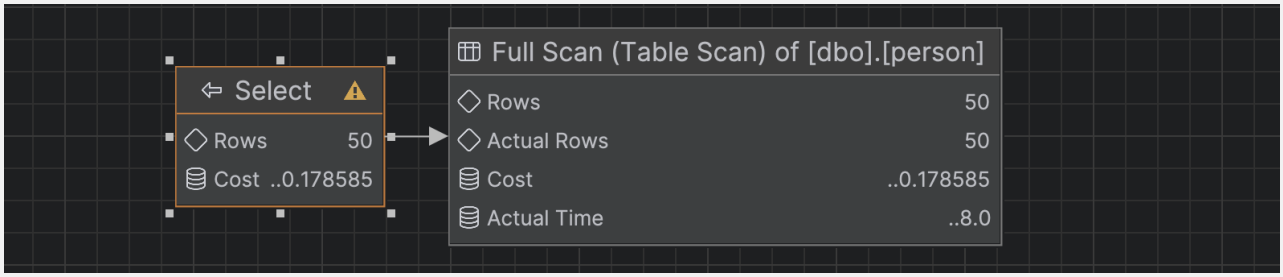
Zapytanie 1 - filtrowanie po nazwisku.



Zapytanie 2 - filtrowanie po nazwisku i imieniu.

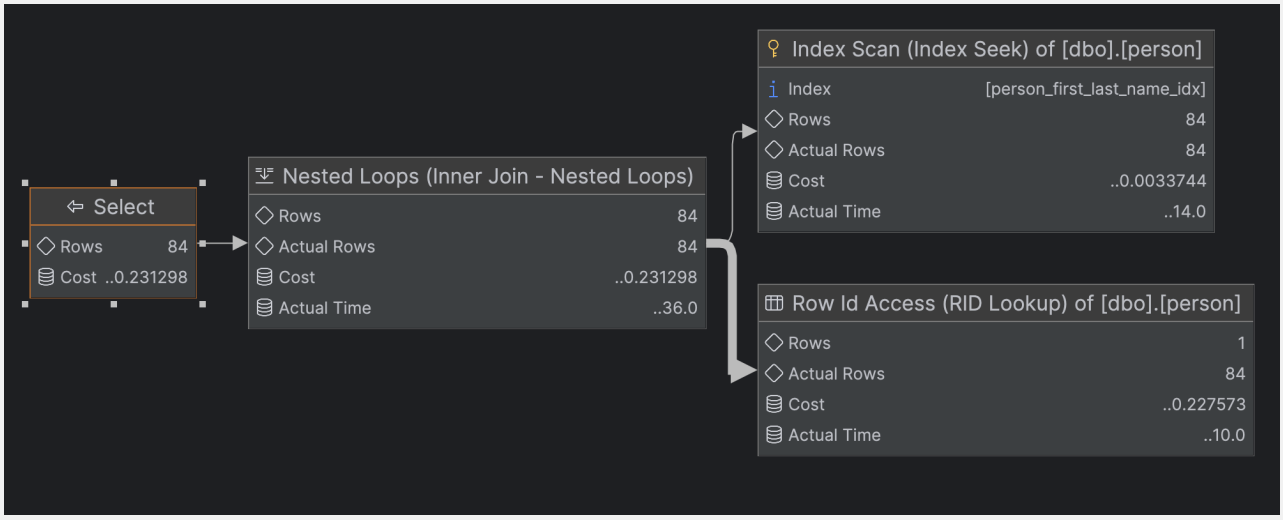


Zapytanie 3 - filtrowanie po imieniu.



Dla 2 i 3 jest zgodnie z oczekiwaniami. 2 jest bardzo wydajne, a dla 3 nie mamy założonego indeksuna imię.

> Dziwi wynik zapytania 1 - przecież mamy indeks na nazwisko, czyli pole po którym filtrujemy. Query optimizer zdecydował, że nie opłaca się go używać, bo mamy dużo wystąpień wartości po której filtrujemy.
>
> Możemy wymusić użycie indeksu: `select * from [person] WITH (INDEX(person_first_last_name_idx)) where lastname = 'Price'`



> Faktycznie widać, że tak jest wolniej.

Zadanie 3

Skopiuj tabelę `PurchaseOrderDetail` do swojej bazy danych:

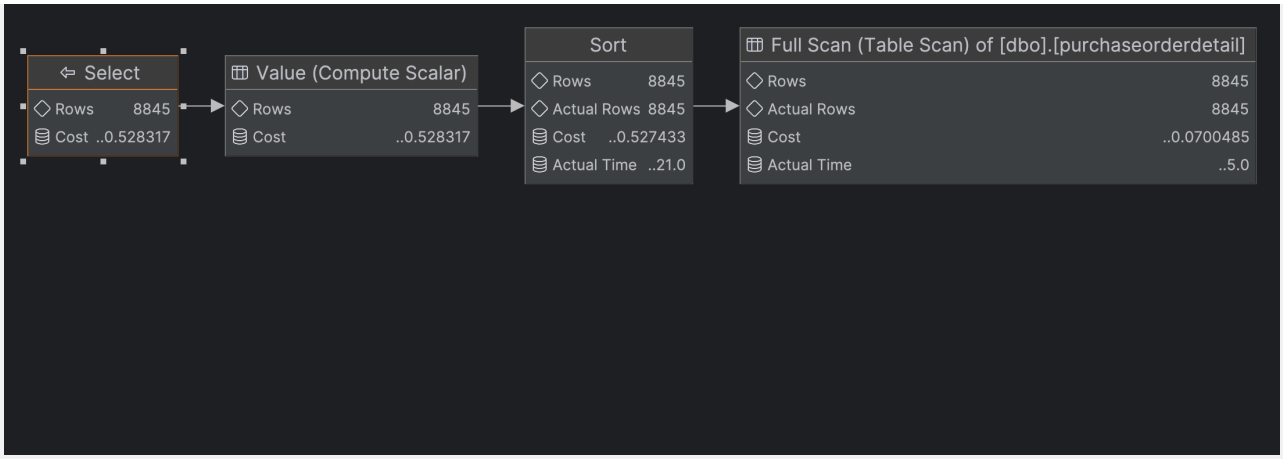
```
select * into purchaseorderdetail from
adventureworks2017.purchasing.purchaseorderdetail
```

Wykonaj analizę zapytania:

```
select rejectedqty, ((rejectedqty/orderqty)*100) as rejectionrate,
productid, duedate
from purchaseorderdetail
order by rejectedqty desc, productid asc
```

Która część zapytania ma największy koszt?

> Wyniki:



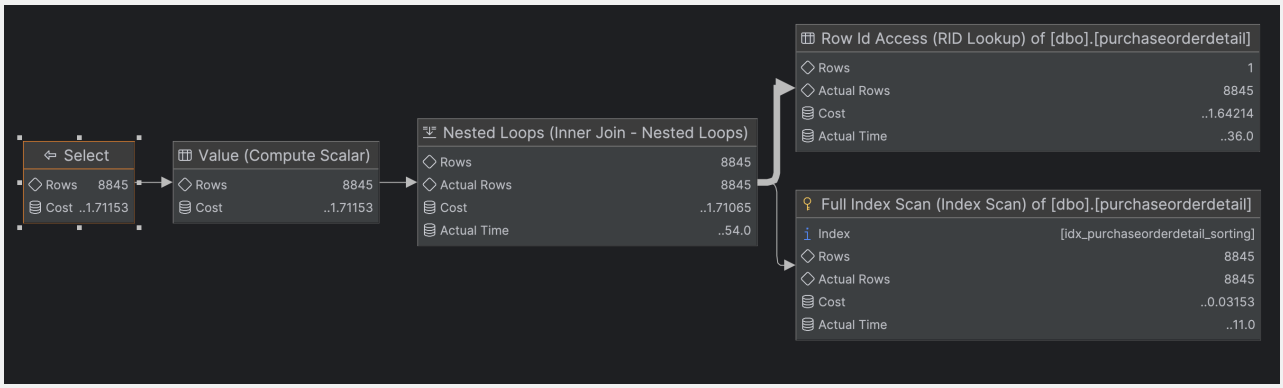
Widać, że sortowanie ma największy koszt.

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

Wyniki:

Próba nieudana:

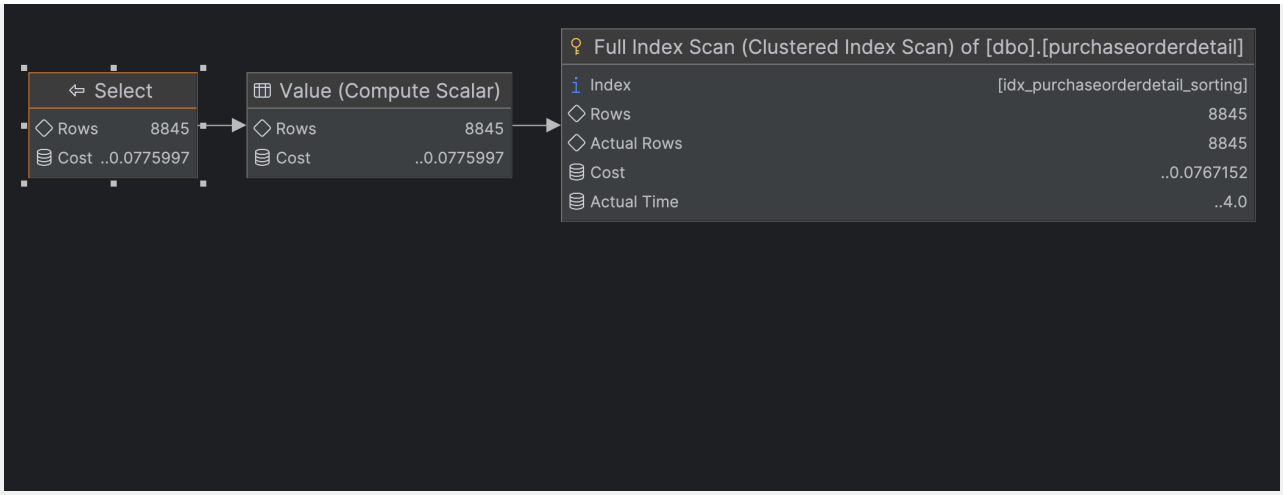
```
CREATE INDEX idx_purchaseorderdetail_sorting ON
purchaseorderdetail(rejectedqty desc, productid asc)
```



Zwykły indeks w tym przypadku nie pomaga, tylko psuje - lookup kosztuje 3 razy więcej niż posortowanie tego. Choć wydaje mi się że mógłby dla dużo większej tabeli.

Próba udana:

```
CREATE CLUSTERED INDEX idx_purchaseorderdetail_sorting ON
purchaseorderdetail(rejectedqty desc, productid asc)
```



Indeks klastrowany zmienia organizację pamięci tak, żeby była ułożona zgodnie z tamtym sortowaniem, dzięki czemu jest ono darmowe (nawet nie ma go w planie zapytania).

Zadanie 4

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

Skopiuj tabelę `Address` do swojej bazy danych:

```
select * into address from  adventureworks2017.person.address
```

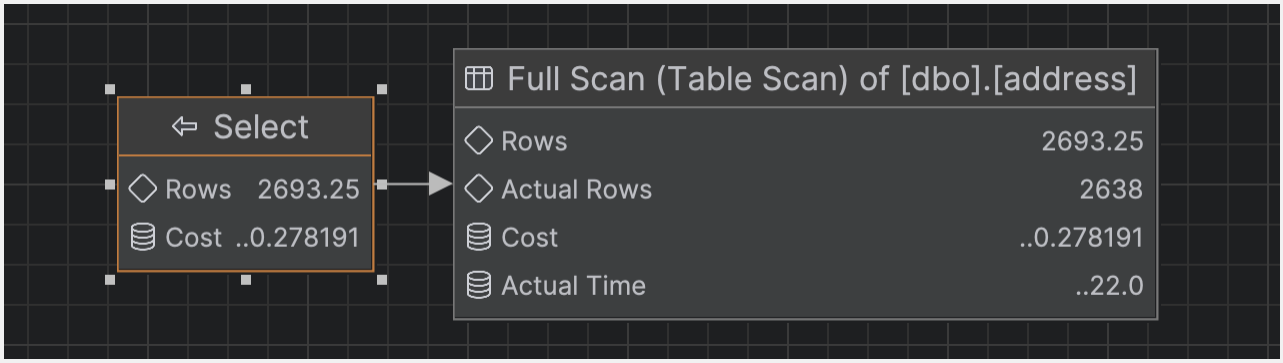
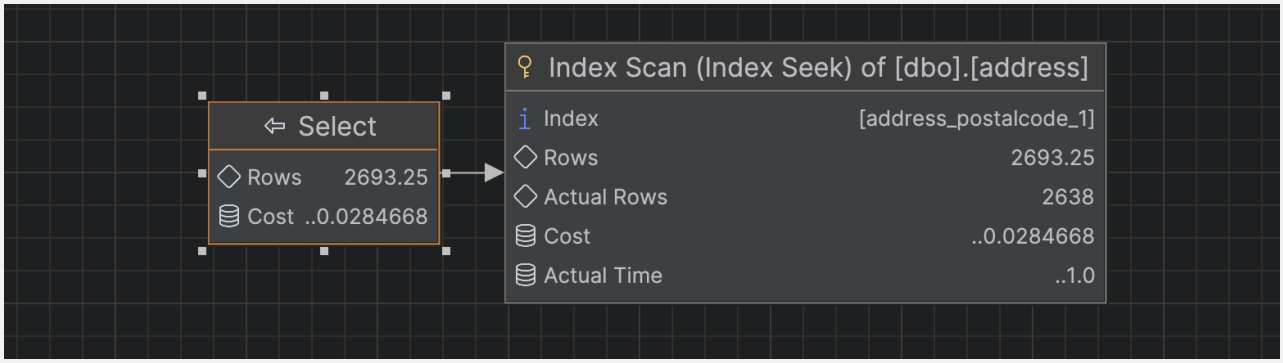
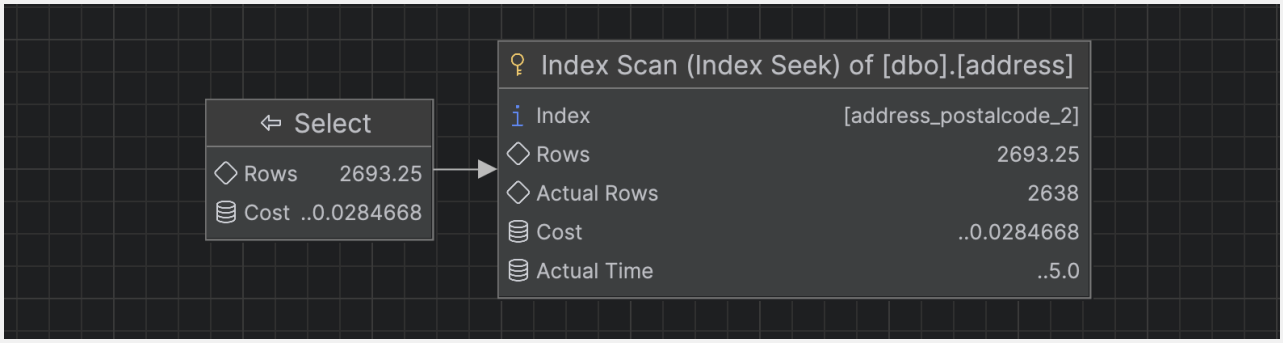
W tej części będziemy analizować następujące zapytanie:

```
select addressline1, addressline2, city, stateprovinceid, postalcode
from address
where postalcode between n'98000' and n'99999'
```

```
create index address_postalcode_1
on address (postalcode)
include (addressline1, addressline2, city, stateprovinceid);
go

create index address_postalcode_2
on address (postalcode, addressline1, addressline2, city,
stateprovinceid);
go
```

Czy jest widoczna różnica w zapytaniach? Jeśli tak to jaka? Aby wymusić użycie indeksu użyj `WITH(INDEX(Address_PostalCode_1))` po `FROM`:

Wyniki:

> Bez indeksu:
>
> 
>
>
> Z indeksem address_postalcode_1
>
> 
>
>
> Z indeksem address_postalcode_2
>
> 
>
>
> Przy użyciu obu indeksów koszt taki sam.

Sprawdź rozmiar Indeksów:

```
select i.name as indexname, sum(s.used_page_count) * 8 as indexsizekb
from sys.dm_db_partition_stats as s
inner join sys.indexes as i on s.object_id = i.object_id and s.index_id =
i.index_id
where i.name = 'address_postalcode_1' or i.name = 'address_postalcode_2'
group by i.name
go
```

Który jest większy? Jak można skomentować te dwa podejścia do indeksowania? Które kolumny na to wpływają?

> Wyniki:

indexname	indexsizekb
address_postalcode_1	1784
address_postalcode_2	1808

Indeks 1 indeksuje po kolumnie `postalcode` a resztę kolumn w liściach indeksu. Indeks 2 indeksuje po `postalcode` a potem po wszystkich kolumnach pokolei. W obu przypadkach indeksy obejmują całe dane w tabeli (cover index) dzięki czemu nie potrzeba dostawać się do danych i jest szybko.

Indeks 2 przydałby się jeśli mielibyśmy wyszukiwać lub sortować kolejno po `postalcode`, `addressline1` itd. Indeks 2 jest genralnie trudniejszy do utrzymania plus ma odrobinę większy rozmiar.

Zadanie 5 – Indeksy z filtrami

Celem zadania jest poznanie indeksów z filtrami.

Skopiuj tabelę `BillOfMaterials` do swojej bazy danych:

```
select * into billofmaterials
from adventureworks2017.production.billofmaterials
```

W tej części analizujemy zapytanie:

```
select productassemblyid, componentid, startdate
from billofmaterials
where enddate is not null
    and componentid = 327
    and startdate >= '2010-08-05'
```

Zastosuj indeks:

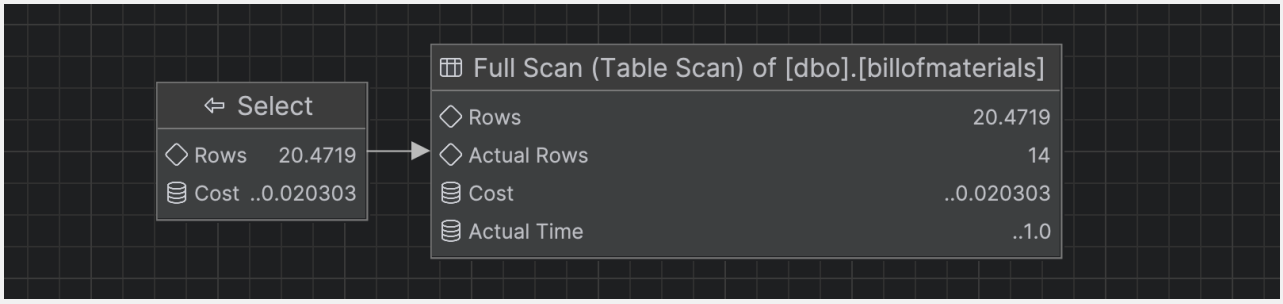
```
create nonclustered index billofmaterials_cond_idx
    on billofmaterials (componentid, startdate)
    where enddate is not null
```

Sprawdź czy działa.

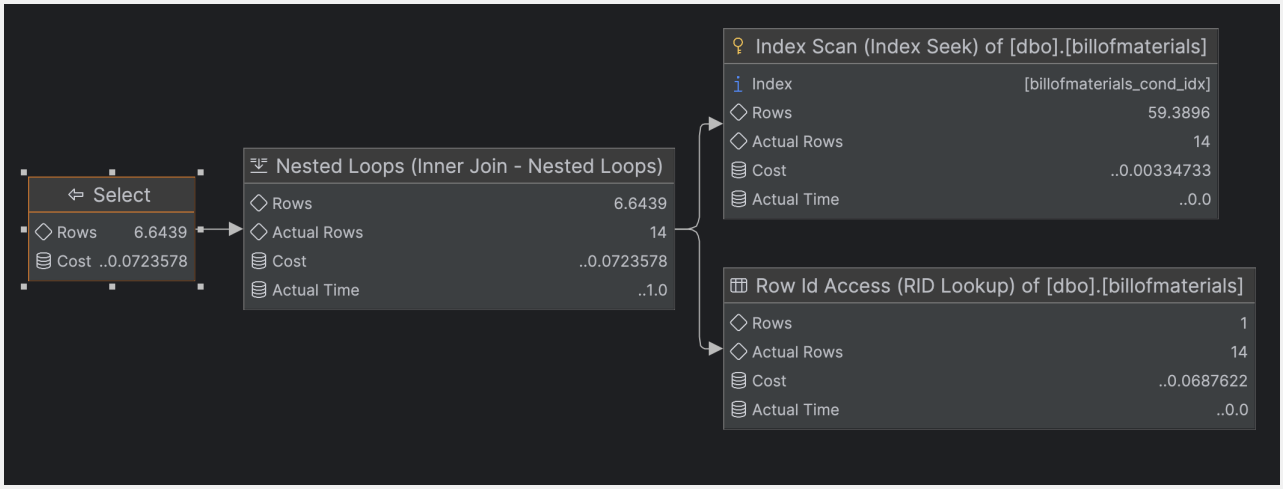
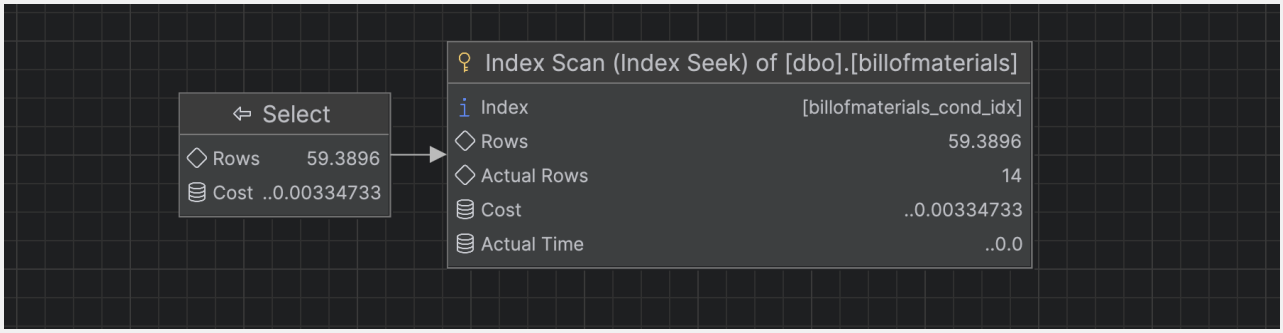
Przeanalizuj plan dla poniższego zapytania:

Czy indeks został użyty? Dlaczego?

> Wyniki:

> Plan zapytania bez indeksu
>
> 
>
>
> Domyślnie po stworzeniu indeksu zapytanie wykonuje się tym samym planem.

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

> Możemy wymusić użycie stworzonego przez nas indeksu:
>
> 
>
>
> Indeks nie został użyty, bo lookupy są wolniejsze. Query optimizer słusznie wybrał.
>
> Możemy pod to query wyeliminować lookupy includując `productassemblyid` do indeksu:
>
> ```
> create nonclustered index billofmaterials_cond_idx
>    on billofmaterials (componentid, startdate)
>    include (productassemblyid)
>    where enddate is not null
> ```
>
> Wtedy zapytanie robi się szybciutko.
>
> 
>

Punktacja:

zadanie	pkt

1	2
2	2
3	2
4	2
5	2
razem	10