

# Indeksy, optymalizator

## Lab 5

**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

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
```

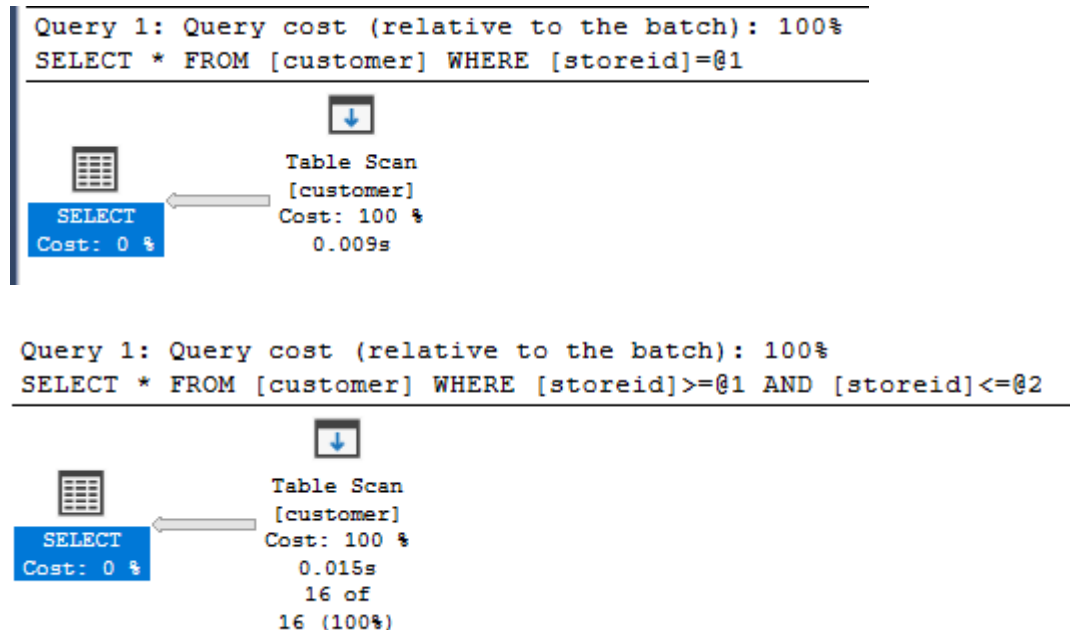
Wykonaj analizy zapytań:

```
select * from customer where storeid = 594
```

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

Zanotuj czas zapytania oraz jego koszt:

Wyniki:



Pierwsze zapytanie: 0.139158

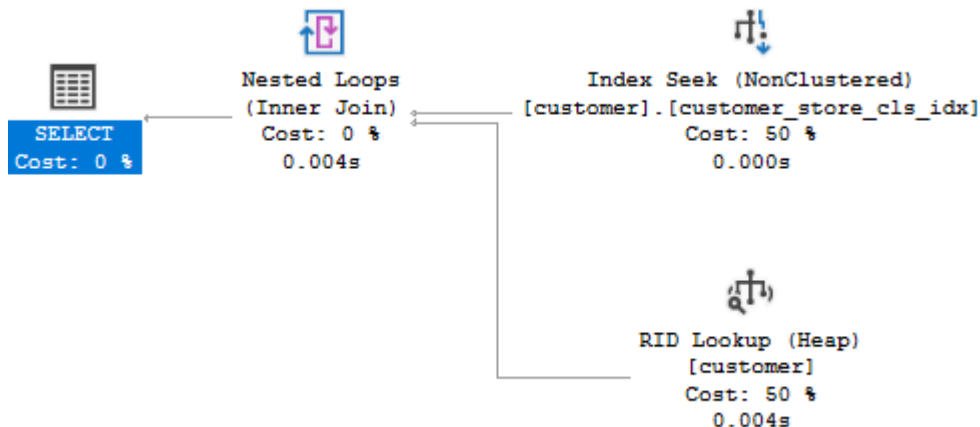
Drugie zapytanie: 0.139158

Dodaj indeks:

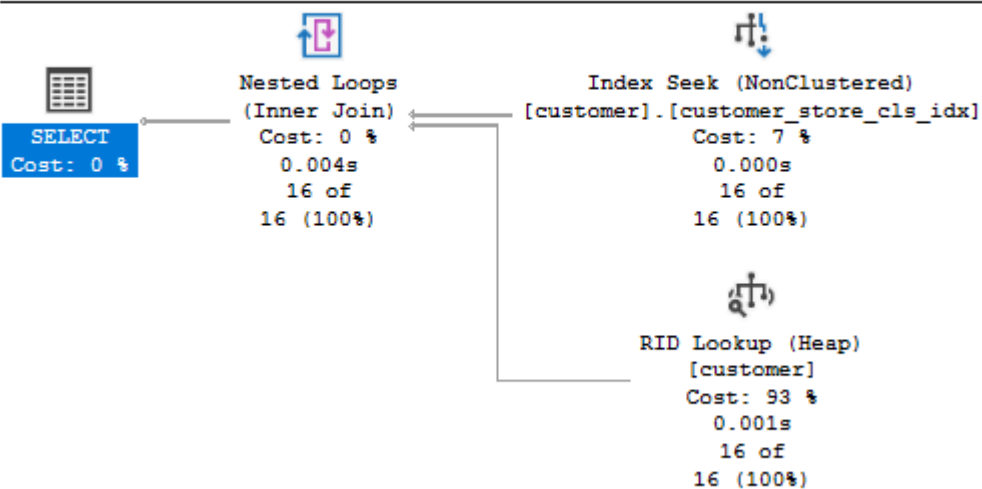
```
create index customer_store_cls_idx on customer(storeid)
```

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

Wyniki:



Query 1: Query cost (relative to the batch): 100%  
SELECT \* FROM [customer] WHERE [storeid]>=@1 AND [storeid]<=@2



Pierwsze zapytanie: 0.0065704

Drugie zapytanie: 0.0507122

Na planie widać dodatkową strukturę heap, plan jest bardziej skomplikowany od podstawowego. Można zauważyć nieznaczne obniżenie czasu i znaczne obniżenie kosztu. Warto zaznaczyć, że dla zapytań bez użycia indeksów, ich koszt był identyczny. Z kolei dla zapytań z użyciem indeksu nieklastrowego, drugie zapytanie miało znacznie większy koszt, co wynika z wielokrotnego

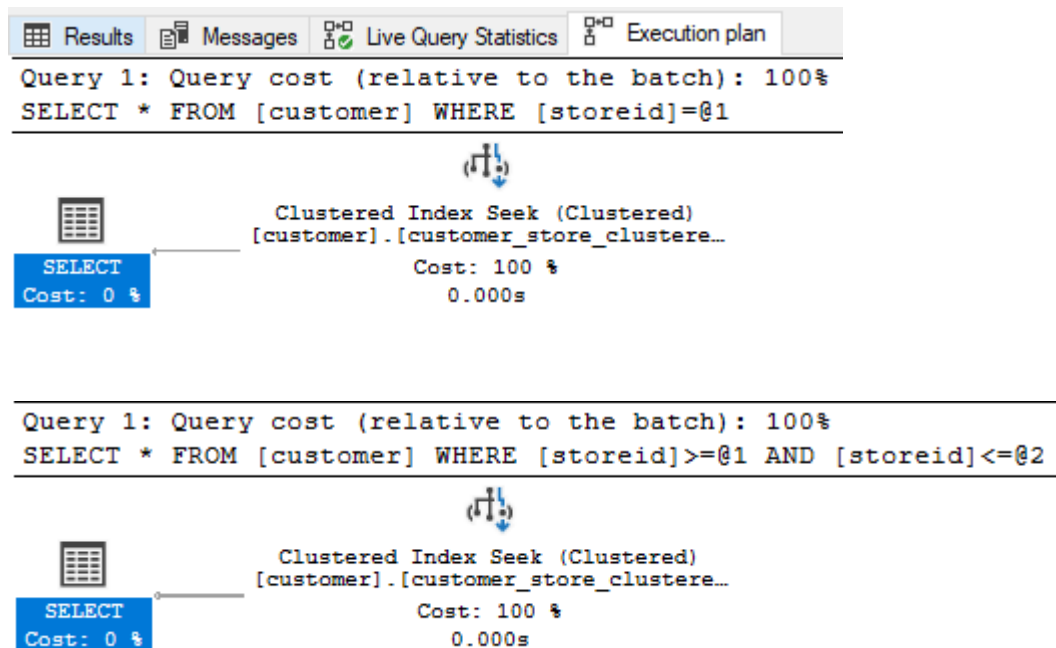
odwoływania się do stworzonej struktury. Dalszą możliwością optymalizacji może być stworzenie indeksu klastrowego.

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:



Pierwsze zapytanie: 0.0032831

Drugie zapytanie: 0.0032996

Koszt oraz czas wykonania znacznie zmniejszył się w stosunku do zastosowania indeksu nieklastrowego. Na planie również widać znaczne uproszczenie operacji. Nie ma dodatkowej struktury heap, jest tylko jedna operacja clustered index seek. Problemem jednak jest fakt, że możemy utworzyć tylko jedną taką strukturę na całą tabelę, dlatego indeks taki najlepiej nałożyć na często używane kolumny.

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

Celem zadania jest poznanie indeksów z przechowywujących 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'
```

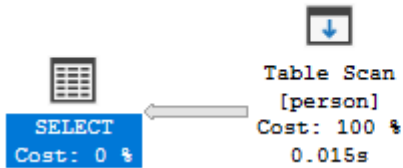
```
select * from [person] where lastname = 'Agbonile' and firstname = 'Osarumwense'
```

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

Co można o nich powiedzieć?

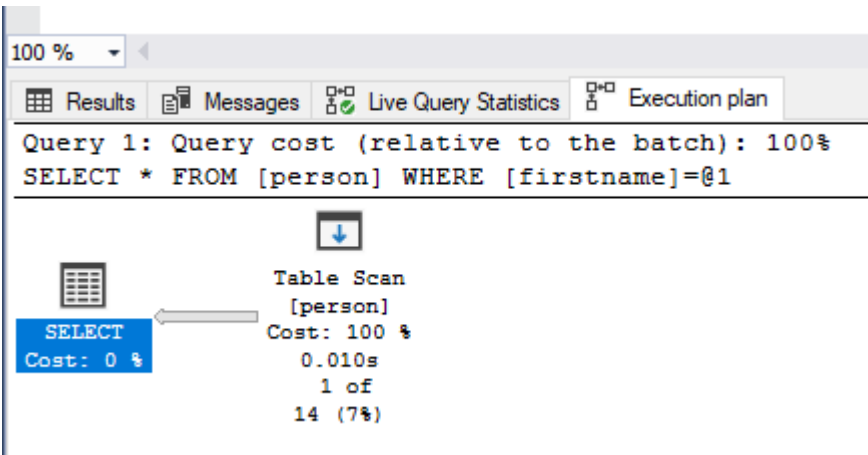
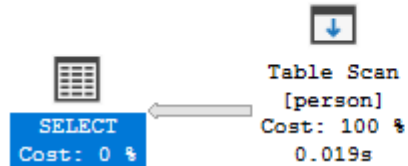
Wyniki:

Query 1: Query cost (relative to the batch): 100%  
SELECT \* FROM [person] WHERE [lastname]=@1



Results Messages Live Query Statistics Execution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT \* FROM [person] WHERE [lastname]=@1 AND [firstname]=@2



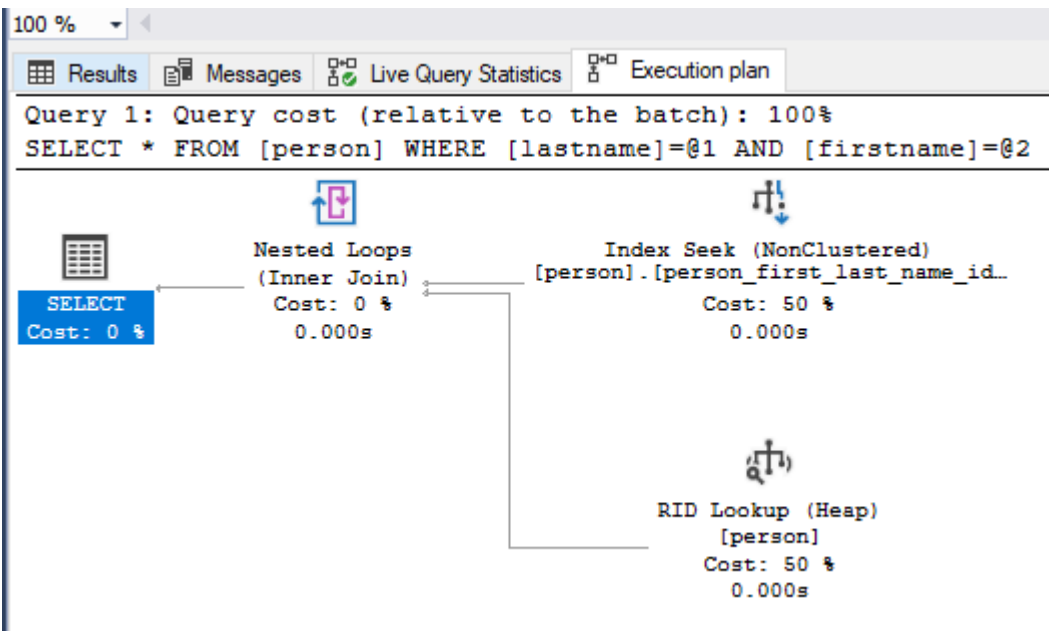
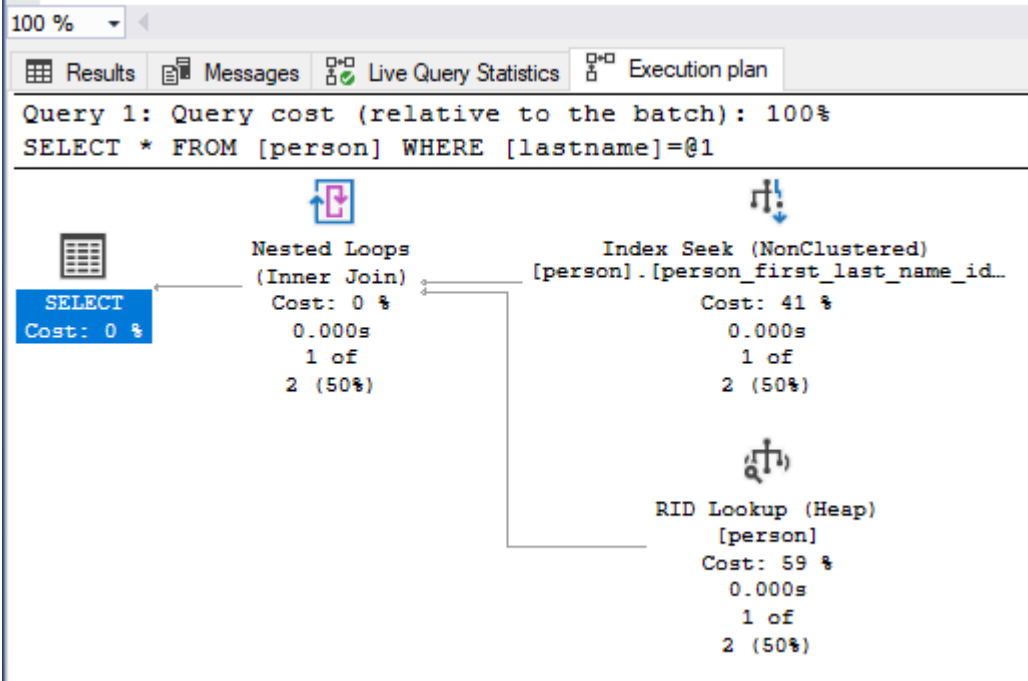
Kazde zapytanie ma taką samą strukturę i taki sam koszt.

Przygotuj indeks obejmujący te zapytania:

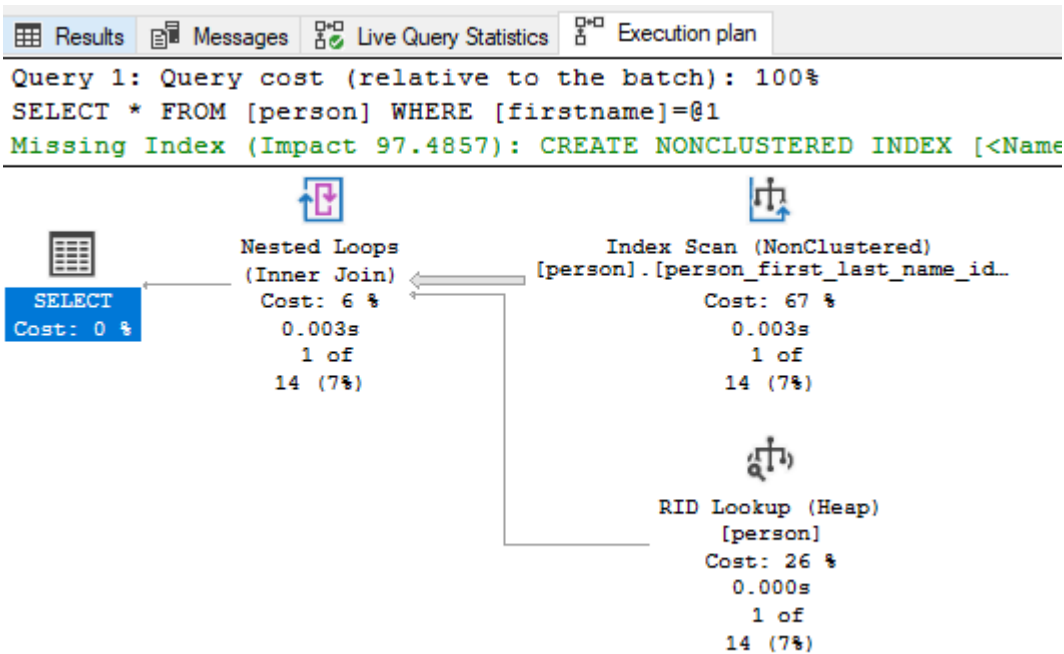
```
create index person_first_last_name_idx  
on person(lastname, firstname)
```

Sprawdź plan zapytania. Co się zmieniło?

Wyniki:





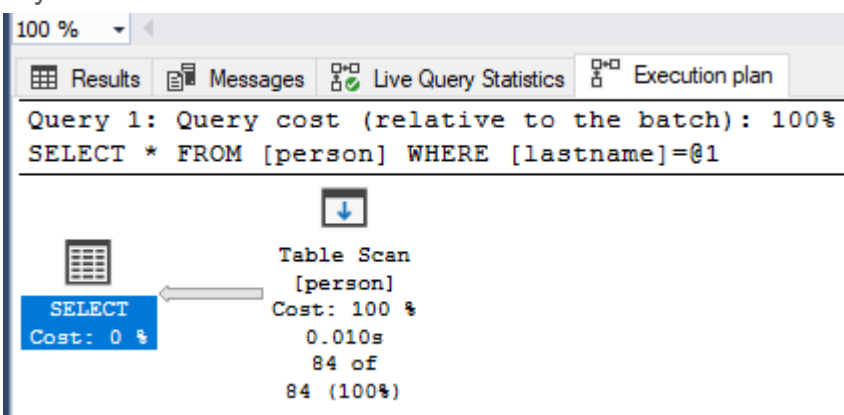


We wszystkich 3 zapytaniach pojawiła się pętla łącząca wyniki oraz skanowanie indeksów. Koszt wyraźnie zmalał.

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

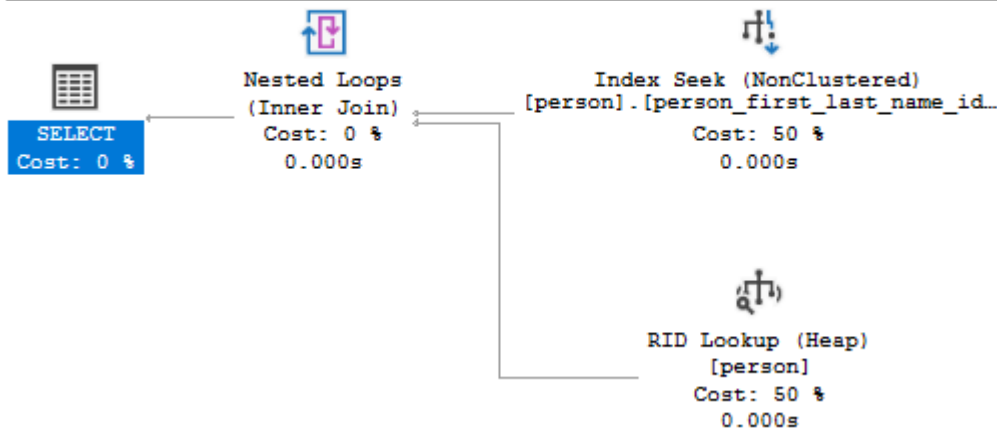
Czym różni się ten plan od zapytania o 'Osarumwense Agbonile' . Dlaczego tak jest?

Wyniki:



Query 1: Query cost (relative to the batch): 100%

SELECT \* FROM [person] WHERE [lastname]=@1 AND [firstname]=@2

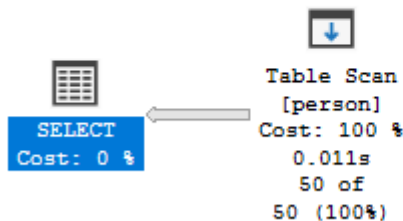


Results Messages Live Query Statistics Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT \* FROM [person] WHERE [firstname]=@1

Missing Index (Impact 97.9128): CREATE NONCLUSTERED INDEX



Tym razem różnica w wykonywanych operacjach oraz koszcie, pojawiła się tylko w przypadku drugiego zapytania. Wynika to z faktu, że w przypadku poprzedniego zapytania, mieliśmy tylko jeden zwracany rekord w każdym przypadku. W bazie istnieje jednak wiele osób o imieniu Angela lub nazwisku Price, co sprawia że dane są mało specyficzne. W tym przypadku planer stwierdził, że lepiej jest nie wykorzystywać utworzonego indeksu.

## Zadanie 3

Skopiuj tabelę PurchaseOrderDetail do swojej bazy danych:

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

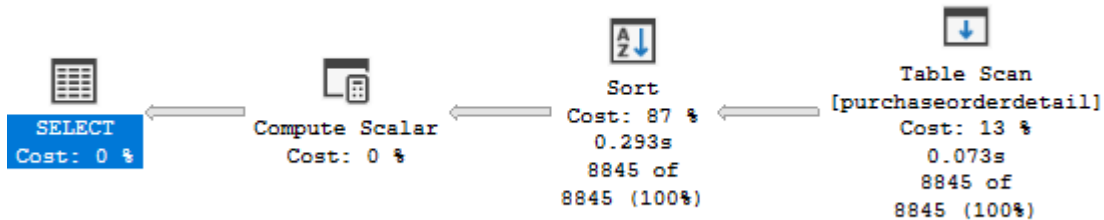
Wykonaj analizę zapytania:

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

Która część zapytania ma największy koszt?

Wyniki:

Query 1: Query cost (relative to the batch): 100%  
 select rejectedqty, ((rejectedqty/orderqty)\*100) as rejectionrate, p



Jak widać na załączonym diagramie, największy koszt generuje operacja sortowania danych, jest to aż 87% całego kosztu zapytania. Koszt całego zapytania to 0.528317.

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

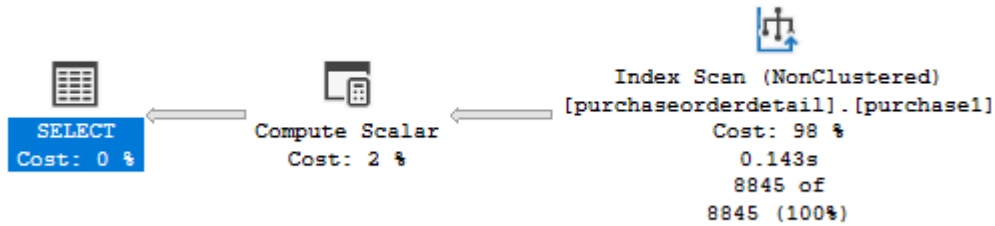
Można zastosować indeks na kolumny, które są użyte podczas operacji sortowania. Ważne jest dodanie klauzuli include, w której zawarte są kolumny używane bezpośrednio w select. Bez nich indeks nie jest brany pod uwagę.

```
create index purchase1
on purchaseorderdetail (rejectedqty desc, productid asc) include (orderqty, duedate);
```

Ponownie wykonaj analizę zapytania:

Wyniki:

```
Query 1: Query cost (relative to the batch): 100%
select rejectedqty, ((rejectedqty/orderqty)*100) as rejectio
```



Operacja skanowania i sortowania została zastąpiona przez pojedynczą operację Full Index Scan, a całkowity koszt wykonania zapytania wyniósł 0.0406, co jest znaczną poprawą w stosunku do braku indeksu (około 13 razy).

## 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

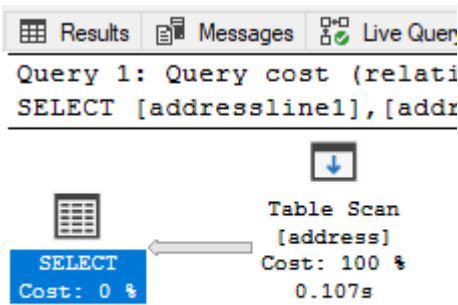
create index address_postalcode_2
on address (postalcode, addressline1, addressline2, city, stateprovinceid);
go
```

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

Wyniki:

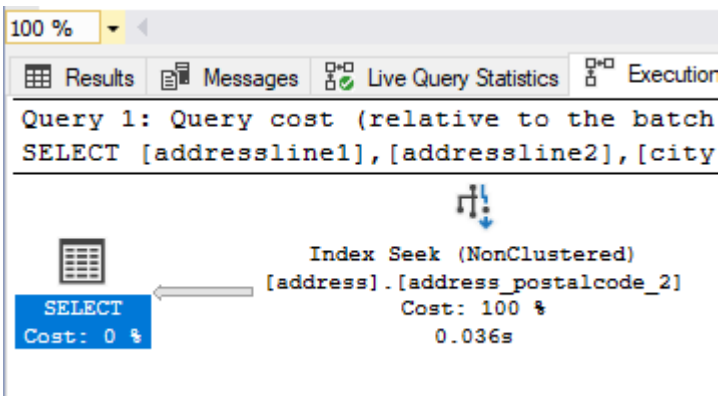
Bez uzycia indeksu

	addressline1	addressline2	city	stateprovinceid	postalcode
1	1970 Napa Ct.	NULL	Bothell	79	98011
2	9833 Mt. Dias Blv.	NULL	Bothell	79	98011
3	7484 Roundtree Drive	NULL	Bothell	79	98011
4	9539 Glenside Dr	NULL	Bothell	79	98011
5	1226 Shoe St.	NULL	Bothell	79	98011
6	1399 Firestone Drive	NULL	Bothell	79	98011
7	5672 Hale Dr.	NULL	Bothell	79	98011
8	6387 Scenic Avenue	NULL	Bothell	79	98011
9	8713 Yosemite Ct.	NULL	Bothell	79	98011
10	250 Race Court	NULL	Bothell	79	98011
11	1318 Lasalle Street	NULL	Bothell	79	98011
12	5415 San Gabriel Dr.	NULL	Bothell	79	98011
13	9265 La Paz	NULL	Bothell	79	98011
14	8157 W. Book	NULL	Bothell	79	98011
15	4912 La Vuelta	NULL	Bothell	79	98011
16	40 Ellis St.	NULL	Bothell	79	98011



Z indeksem

	addressline1	addressline2	city	stateprovinceid	postalcode
1	225 South 314th Street	NULL	Federal Way	79	98003
2	108 Lakeside Court	NULL	Bellevue	79	98004
3	1343 Prospect St	NULL	Bellevue	79	98004
4	1648 Eastgate Lane	NULL	Bellevue	79	98004
5	2284 Azalea Avenue	NULL	Bellevue	79	98004
6	25915 140th Ave Ne	NULL	Bellevue	79	98004
7	2681 Eagle Peak	NULL	Bellevue	79	98004
8	2947 Vine Lane	NULL	Bellevue	79	98004
9	3067 Maya	NULL	Bellevue	79	98004
10	3197 Thomhill Place	NULL	Bellevue	79	98004
11	3284 S. Blank Avenue	NULL	Bellevue	79	98004
12	332 Laguna Niguel	NULL	Bellevue	79	98004
13	3454 Bel Air Drive	NULL	Bellevue	79	98004
14	3670 All Ways Drive	NULL	Bellevue	79	98004
15	3708 Montana	NULL	Bellevue	79	98004
16	3711 Rollingwood Dr	NULL	Bellevue	79	98004



W przypadku zapytania z wykorzystaniem indeksów, kolumny są dodatkowo posortowane, według kolumny postalcode, adressline1, adressline2, city i stateprovinceid. Analizując plan zapytania możemy zobaczyć, że Full Scan został zastąpiony Index Scanem.

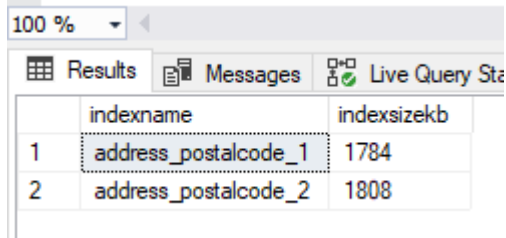
Sprawdź rozmiar Indeksów:

```

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:



The screenshot shows a SQL Server interface with a query results window. At the top, there's a zoom level of 100%. Below it are tabs for 'Results', 'Messages', and 'Live Query Sta'. The 'Results' tab is active, displaying a table with two columns: 'indexname' and 'indexsizekb'. There are two rows of data. The first row has 'address\_postalcode\_1' and '1784'. The second row has 'address\_postalcode\_2' and '1808'.

	indexname	indexsizekb
1	address_postalcode_1	1784
2	address_postalcode_2	1808

Drugi indeks, jest nieznacznie większy. Wynika to z faktu, że w przypadku pierwszego indeksu wykorzystana jest tylko kolumna postcode, z kolei w drugim indeksie wszystkie. Jeżeli w zapytaniu będziemy używać wszystkich kolumn, to szybkość zrekompensuje koszty pamięciowe. W przypadku gdy np. w klauzuli WHERE wykorzystujemy w większości przypadków tylko postcode, lepiej zastosować indeks pierwszy.

## 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:

The screenshot shows the 'Execution plan' tab in SQL Server Enterprise Manager. The query is: `SELECT [productassemblyid], [componentid], [startdate] FROM [billofmaterials]`. The execution plan consists of a single operator: 'Table Scan'. A tooltip is displayed over this operator, providing the following details:

Table Scan	
Scan rows from a table.	
Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	14
Actual Number of Rows Read	2679
Actual Number of Batches	0
Estimated Operator Cost	0.020303 (100%)

Indeks nie został użyty, co wynika z faktu, z małej różnorodności danych (dosyć często pojawiają się dane z takim samym componentid oraz startdate). Podobnie jak w zadaniu 2, wykorzystanie indeksu w tym przypadku nie powinno przyspieszyć zapytań.

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

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

The screenshot shows the 'Execution plan' tab in SQL Server Enterprise Manager. The query is: `select productassemblyid, componentid, startdate from bill`. The execution plan consists of two operators: 'Nested Loops (Inner Join)' and 'Index Seek (NonClustered) [billofmaterials].[billofmaterials]'. A tooltip is displayed over the 'Index Seek' operator, providing the following details:

SELECT	
Actual Number of Rows for All Executions	14
Cached plan size	32 KB
Degree of Parallelism	1
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0.0723578
Estimated Number of Rows for All Executions	0



#### Wyniki:

Po wymuszeniu indeksu koszt całego zapytania wzrósł 3 krotnie. Wynika to z opisanego wyżej problemu, dane często powtarzają się. Rozwiązaniem mogłoby być np. dodanie `INCLUDE(productassemblyid)`

#### Punktacja:

zadanie	pkt
1	2
2	2
3	2
4	2
5	2
razem	10