

Indeksy, optymalizator

Lab 5

Imię i nazwisko: Ewa Pelc, Kacper Sobczyk

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
```

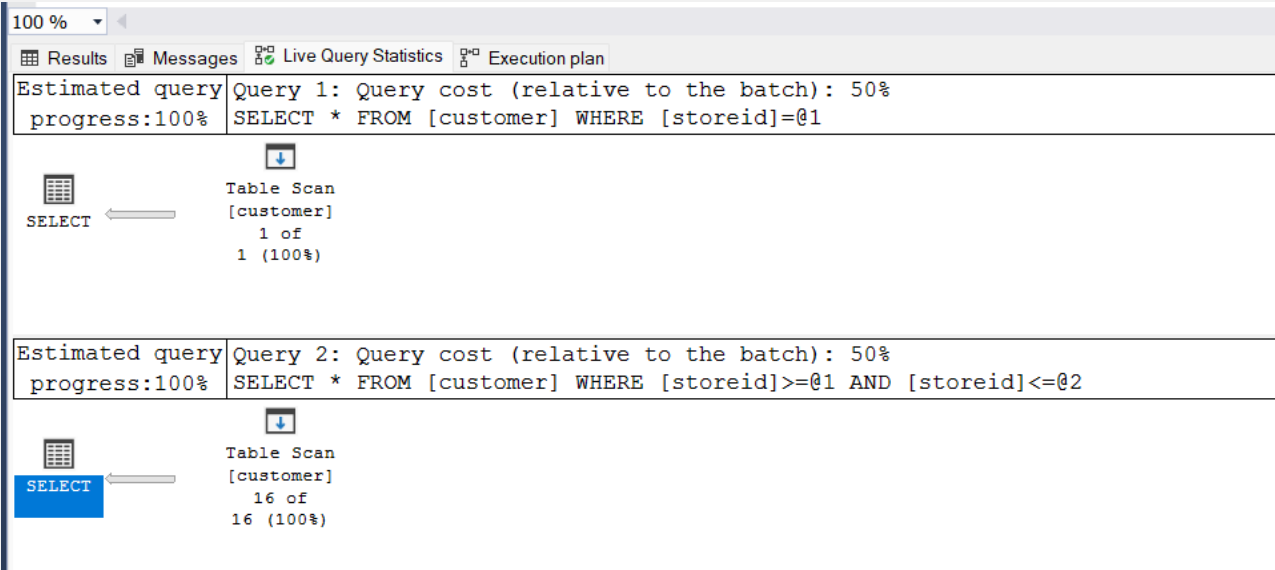
Wykonaj analizy zapytań:

```
select * from customer where storeid = 594

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

Zanotuj czas zapytania oraz jego koszt koszt:

Wyniki:



Koszt pierwszego zapytania:

| | |
|------------------------|-----------|
| Estimated I/O Cost | 0,117278 |
| Estimated Subtree Cost | 0,139158 |
| Estimated CPU Cost | 0,0218805 |

Koszt drugiego zapytania:

| | |
|------------------------|-----------|
| Estimated I/O Cost | 0,117278 |
| Estimated Subtree Cost | 0,139158 |
| Estimated CPU Cost | 0,0218805 |

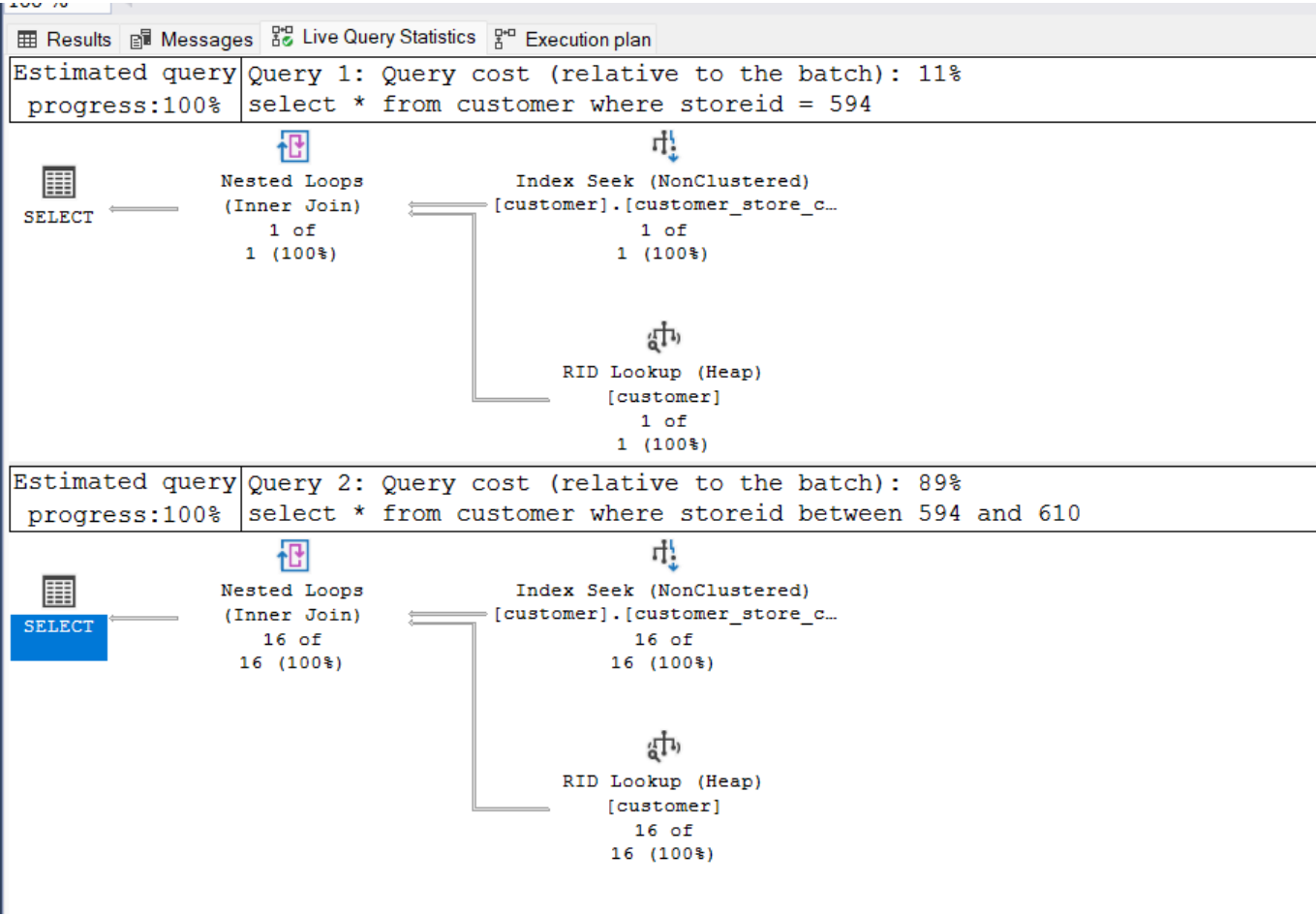
Komentarz:

W obu przypadkach koszty zapytania są takie same. Zapewne ma to związek z wykonaną operacją table scan. W obu przypadkach zapytanie odczytało wszystkie 19820 rekordów znajdujących się w tabeli. Pozyskanie tych, które pasują do klauzuli where nie wiąże się z dodatkowym kosztem ze względu na ilość rekordów.

Dodaj indeks:

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

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



Koszt pierwszego:

| | |
|------------------------|-----------|
| Estimated Subtree Cost | 0,0065704 |
|------------------------|-----------|

Koszt drugiego:

| | |
|------------------------|-----------|
| Estimated Subtree Cost | 0,0507122 |
|------------------------|-----------|

Wyniki:

Komentarz: Po dodaniu klastra zmienił się plan wykonania zapytania. Table Scan został zastąpiony bardziej optymalnym blokiem Index Scan, który przeszukuje indeks w poszukiwaniu konkretnych wartości, które są wymagane w zapytaniu. Ta operacja szybko ograniczy liczbę wierszy, które będą rozpatrywane dalej.

Następnie jest blok RID Lookup, aby znaleźć pozostałe kolumny (Index Scan wyszukuje tylko wartości z kolumny indeksu, a potem muszą zostać do tego "dołożone" wartości z pozostałych kolumn).

Następnie występuje blok Nested Loops (Inner Join). Blok ten występuje wspólnie z blokiem RID Lookup do znalezienia pozostałych wartości wierszy i kolumn.

Ważne: Koszt Index Scan to zaledwie około 0.0001, dla porównania koszt Table Scan to 0.02

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:

ResultsMessagesLive Query StatisticsExecution plan

Estimated query progress:100%

Query 1: Query cost (relative to the batch): 50%
(@1 smallint)SELECT * FROM [customer] WHERE [storeid]=@1

SELECT

Clustered Index Seek (Cluste...
[customer].[customer_store_c...
1 of
1 (100%)

Estimated query progress:100%

Query 2: Query cost (relative to the batch): 50%
(@1 smallint,@2 smallint)SELECT * FROM [customer] WHERE [storeid]>=@1 AND [storeid]<=@2

SELECT

Clustered Index Seek (Cluste...
[customer].[customer_store_c...
16 of
16 (100%)

Koszt pierwszego:

| | |
|---|-----------|
| Clustered Index Seek (Cluste... | |
| [customer].[customer store c... | |
| SELECT | |
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 1 |
| Cached plan size | 24 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,0032831 |
| Estimated Number of Rows Per Execution | 1 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| Est pr (@1 smallint)SELECT * FROM [customer] WHERE [storeid]=@1 | |

Koszt drugiego:

| | |
|---|-----------|
| SELECT | |
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 16 |
| Cached plan size | 32 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,0032996 |
| Estimated Number of Rows Per Execution | 16 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| (@1 smallint,@2 smallint)SELECT * FROM [customer] WHERE | |

Komentarz:

Tak jak poprzednio czasy przy tak małej ilości rekordów w tabeli zawierają się w milisekundach. Jednak patrząc na koszt to można zauważyć, że zmniejszył się on dwukrotnie w porównaniu do poprzedniego indeksu.

W planie wykonania zapytania pojawił się blok Clustered Index Seek, który wyszukuje odpowiedni wiersz w indeksie. Indeks klastrowany zorganizował fizyczną kolejność danych na dysku zgodnie z kluczem głównym tabeli. Organizacja danych opiera się na strukturze B-drzewa, które jest bardzo wydajne w wyszukiwaniu wartości. W przypadku B-drzewa nie wszystkie wiersze muszą zostać przeskanowane tak jak w przypadku Table Scan. Dodatkowo, dzięki temu że dane są posortowane wg klucza głównego, nie ma konieczności sortowania całej tabeli.

Koszt wykonania obu zapytań to około 0.003, jest to więcej niż w przypadku zwykłego indeksu, ale mniej niż bez zastosowania żadnych indeksów.

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:

Każde z trzech zapytań zwraca jeden rekord w bazie danych i jest nim ta sama osoba.

ResultsMessages

| | businessentityid | persontype | namestyle | title | firstname | middlename | lastname | suffix | emailpromotion | rowguid | modifieddate |
|---|------------------|------------|-----------|-------|-------------|-------------|----------|--------|----------------|--------------------------------------|-------------------------|
| 1 | 4388 | IN | 0 | NULL | Osarumwense | Uwai fiokun | Agbonile | NULL | 0 | 3309A53F-3020-495A-A423-C1DBCCCAC66C | 2013-11-30 00:00:00.000 |

Czasy wykonywania zapytań są niemal identyczne jednak różnią się:

- Zapytanie 1 pokazuje, że został wykonany pełen skan tabeli (Table Scan) i że jedna z dwóch możliwych ścieżek została wybrana do wykonania zapytania. Jest to 50% całej tabeli, co może oznaczać, że zapytanie zostało zoptymalizowane pod kątem wybranego warunku.

- Zapytanie 2 również pokazuje pełen skan tabeli, ale w tym przypadku użyto jednej ścieżki na jedną możliwą, co wskazuje na 100% efektywność w kontekście dostępnych opcji. Oznacza to, że filtr na 'lastname' i 'firstname' skutecznie zawęził wyniki do jednego rekordu bez potrzeby przeglądania większej liczby rekordów.
- Zapytanie 3 także przedstawia pełny skan tabeli, ale w tym przypadku skanuje on tylko 7% tabeli, co sugeruje, że tabeli person jest większa, a zapytanie było w stanie znaleźć odpowiedni rekord szybciej, być może ze względu na lepsze warunki wyszukiwania lub organizację danych.

ResultsMessagesLive Query StatisticsExecution plan

Estimated query progress:100%

Query 1: Query cost (relative to the batch): 33%
(@1 varchar(8000))SELECT * FROM [person] WHERE [lastname]=@1

Table Scan
[person]
0.379s
1 of
2 (50%)

SELECT
0.379s

Estimated query progress:100%

Query 2: Query cost (relative to the batch): 33%
SELECT * FROM [person] WHERE [lastname]=@1 AND [firstname]=@2

Table Scan
[person]
1 of
1 (100%)

SELECT

Estimated query progress:100%

Query 3: Query cost (relative to the batch): 33%
(@1 varchar(8000))SELECT * FROM [person] WHERE [firstname]=@1

Table Scan
[person]
1 of
14 (7%)

SELECT

Koszt pierwszego:

| SELECT | |
|--|----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 1 |
| Cached plan size | 32 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,180807 |
| Estimated Number of Rows Per Execution | 1,86667 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| (@1 varchar(8000))SELECT * FROM [person] WHERE [lastname]=@1 | |

Koszt drugiego:

| | |
|---|----------|
| SELECT | |
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 1 |
| Cached plan size | 32 KB |
| Degree of Parallelism | 1 |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,180807 |
| Estimated Number of Rows for All Executions | 0 |
| Estimated Number of Rows Per Execution | 1 |
| Statement | |
| SELECT * FROM [person] WHERE [lastname]=@1 AND [firstname]=@2 | |

Koszt trzeciego:

| | |
|---|----------|
| SELECT | |
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 1 |
| Cached plan size | 32 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,180807 |
| Estimated Number of Rows Per Execution | 13,5714 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| (@1 varchar(8000))SELECT * FROM [person] WHERE [firstname]=@1 | |

Komentarz:

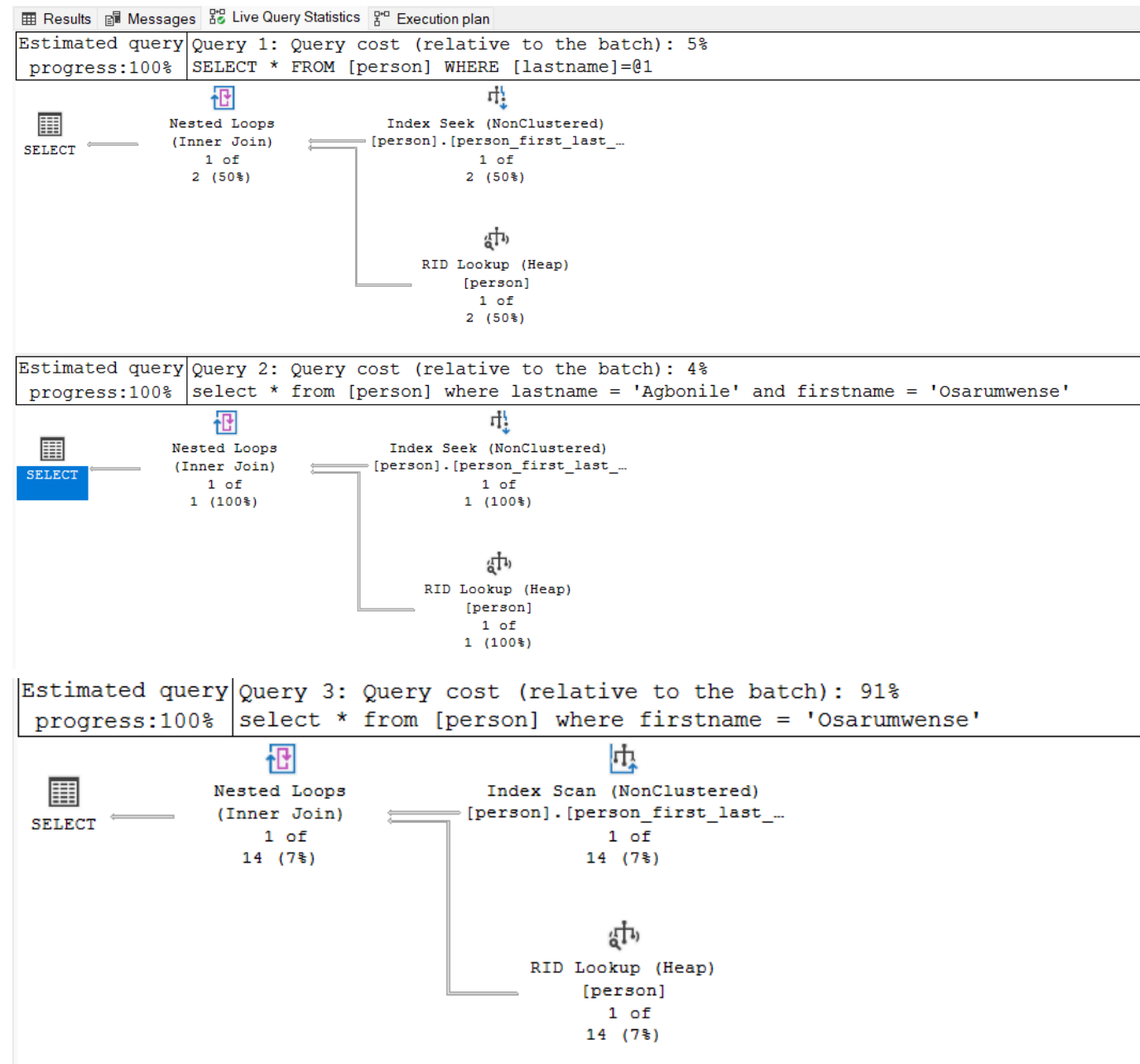
W planie wykonania zapytania występuje tylko operacja Table Scan, która skanuje całą tabelę. Koszty wykonania są takie same dla wszystkich zapytań i wynoszą około 0.18

Przygotuj indeks obejmujący te zapytania:

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

Sprawdź plan zapytania. Co się zmieniło?

Wyniki:



| SELECT | |
|---|-----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 1 |
| Cached plan size | 40 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,0065804 |
| Estimated Number of Rows Per Execution | 1,02337 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| select * from [person] where lastname = 'Agbonile' and firstname = 'Osarumwense' | |

Koszt trzeciego:

| SELECT | |
|--|----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 1 |
| Cached plan size | 32 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,153785 |
| Estimated Number of Rows Per Execution | 13,5714 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| select * from [person] where firstname = 'Osarumwense' | |

Results Messages

| | businessentityid | persontype | namestyle | title | firstname | middlename | lastname | suffix | emailpromotion | rowguid | modifieddate |
|---|------------------|------------|-----------|-------|-------------|------------|----------|--------|----------------|--------------------------------------|-------------------------|
| 1 | 4388 | IN | 0 | NULL | Osarumwense | Uwafiokun | Agbonile | NULL | 0 | 3309A53F-3020-495A-A423-C1DBCCCAC66C | 2013-11-30 00:00:00.000 |

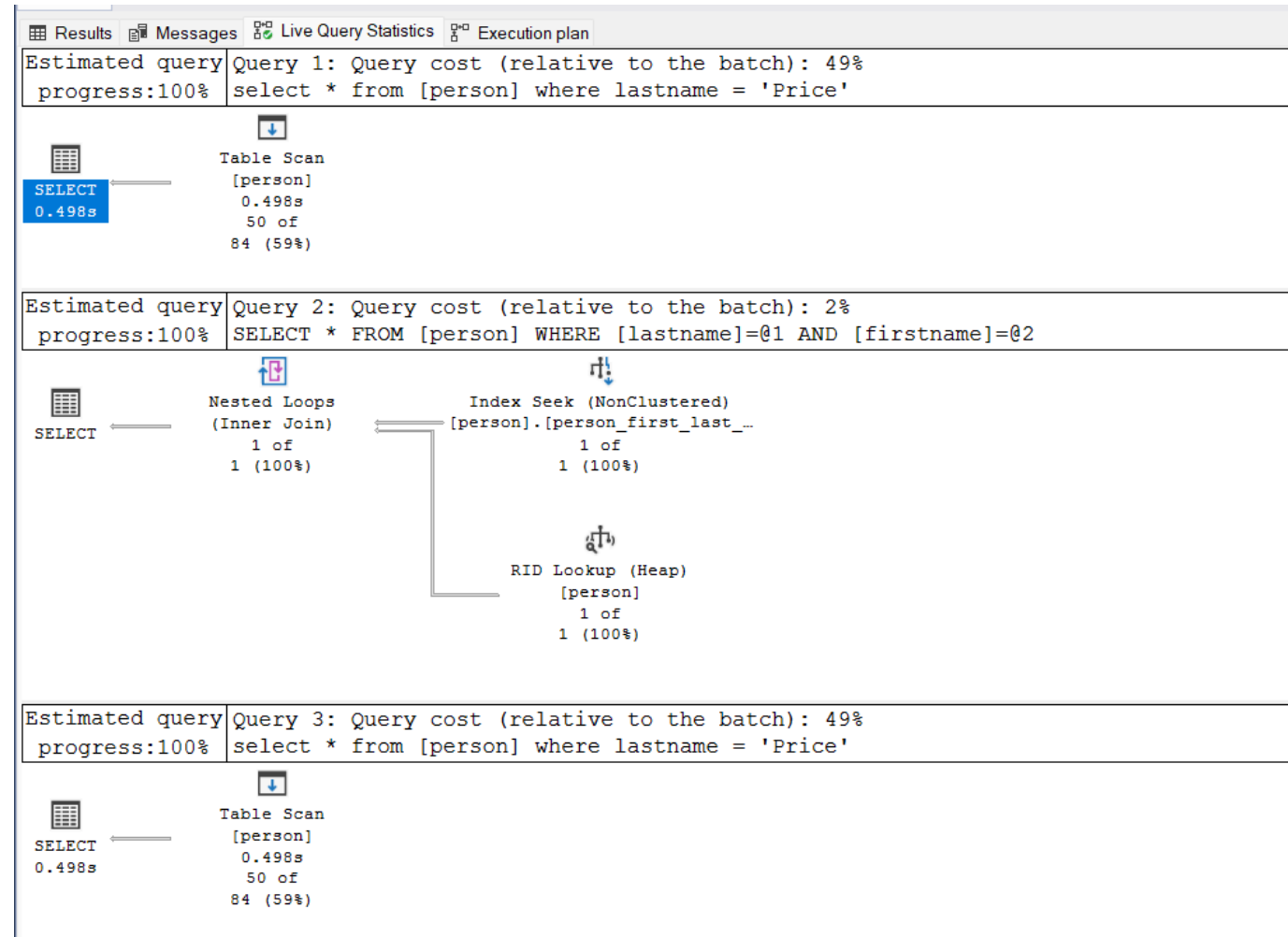
Komentarz:

Tak jak w poprzednim zadaniu, bloki Table Scan po zastosowaniu indeksu zostały zastąpione Index Seek, RID Lookup oraz Nested Loops (Inner Join). Index Seek przeszukuje index, aby zawęzić ilość rozpatrywanych wierszy zgodnie z warunkiem. Następnie RID Lookup oraz Nested Loops wspólnie służą temu, aby znaleźć pozostałe kolumny, o których mowa w zapytaniu.

Istnieje tylko jeden rekord z tabeli Person, o imieniu 'Osarumwense' lub nazwisku 'Agbonile', a jednak koszty wykonania różnią się dla zapytania pierwszego i trzeciego (0.008 vs 0.15).

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?



Koszt pierwszego:

| SELECT | |
|---|----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 50 |
| Cached plan size | 32 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,180807 |
| Estimated Number of Rows Per Execution | 84 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| select * from [person] where lastname = 'Price' | |

Koszt drugiego:

| SELECT | |
|---|-----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 1 |
| Cached plan size | 40 KB |
| Degree of Parallelism | 1 |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,0065804 |
| Estimated Number of Rows for All Executions | 0 |
| Estimated Number of Rows Per Execution | 1,02337 |
| Statement | |
| SELECT * FROM [person] WHERE [lastname]=@1 AND [firstname]=@2 | |

Koszt trzeciego:

| SELECT | |
|---|----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 50 |
| Cached plan size | 32 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,180807 |
| Estimated Number of Rows Per Execution | 84 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| select * from [person] where lastname = 'Price' | |

Ilość wierszy spełniających warunek:

SQLQuery4.sql - D...-5ODD36I\Ewa (74))* SQLQuery3.sql - D...-5ODD36I\Ewa (60))* SQLQue

```
select count(*) from [person] where firstname = 'Angela' ;
select count(*) from [person] where lastname = 'Price' ;
```

100 %

Results Messages

| | |
|---|------------------|
| | (No column name) |
| 1 | 50 |

| | |
|---|------------------|
| | (No column name) |
| 1 | 84 |

Komentarz:

Dla nazwiska 'Price' i imienia 'Angela', tylko zapytanie 2 posiadające warunek imienia i nazwiska jest wykonywane w optymalny sposób. Dla zapytania z warunkiem tylko dla imienia albo nazwiska występuje operacja Table Scan skanująca wszystkie wiersze.

Dlaczego?

- Warto zauważyć, że rekordów z wartością kolumny firstname = 'Angela' lub wartością kolumny lastname = 'Price' jest po kilkadziesiąt. Dane te mogą być rozproszone na wielu stronach.
- Wybór innych planów wykonania przez optymalizator może wynikać z tego ile rekordów pasuje do warunków zapytania. Dla Angela Price jest po kilkadziesiąt, a dla Osarumwense Agbonile tylko jeden rekord.

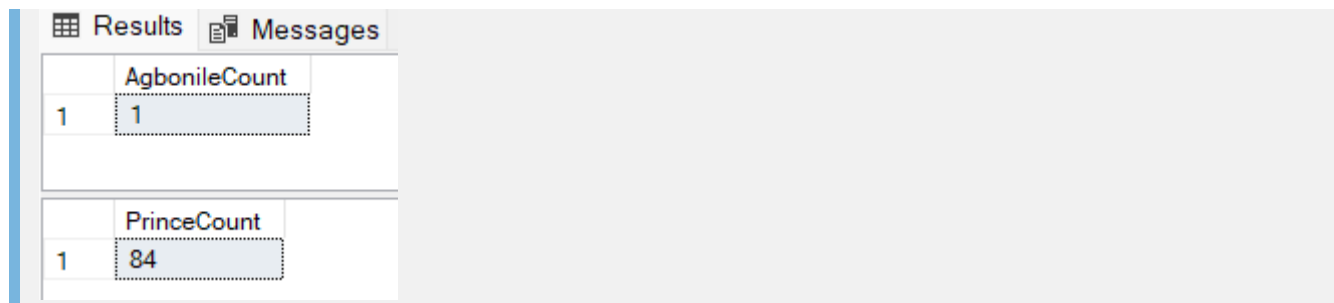
Eksperyment

Pierwsze i trzecie zapytanie nie korzystają z indeksów, co może być związane z "selektywnością" danych w kolumnach LastName i FirstName.

Kolumna o niskiej selektywności ma niewiele unikalnych wartości w stosunku do ogólnej liczby wpisów, podczas gdy kolumna o wysokiej selektywności zawiera wiele unikalnych wartości.

Analiza ilości wierszy z nazwiskami Price i Agbonile sugeruje, że większa liczba wystąpień nazwiska Price może tłumaczyć mniejszą efektywność indeksu, prowadząc do wyboru skanowania tabeli przez silnik bazy danych.

Dowód:



| AgbonileCount | |
|---------------|---|
| 1 | 1 |

| PrinceCount | |
|-------------|----|
| 1 | 84 |

Zadanie 3

Skopiuj tabelę `PurchaseOrderDetail` do swojej bazy danych:

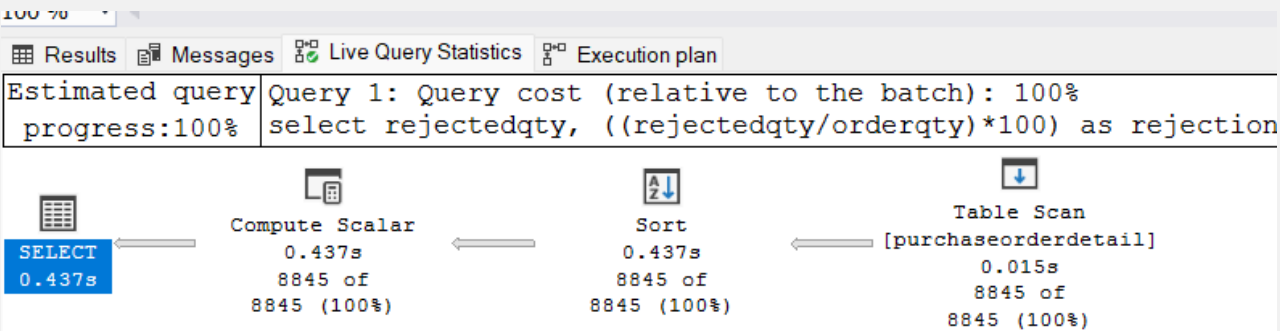
```
select * into purchaseorderdetail from
adventureworks2017.purchasing.purchaseorderdetail
```

Wykonaj analizę zapytania:

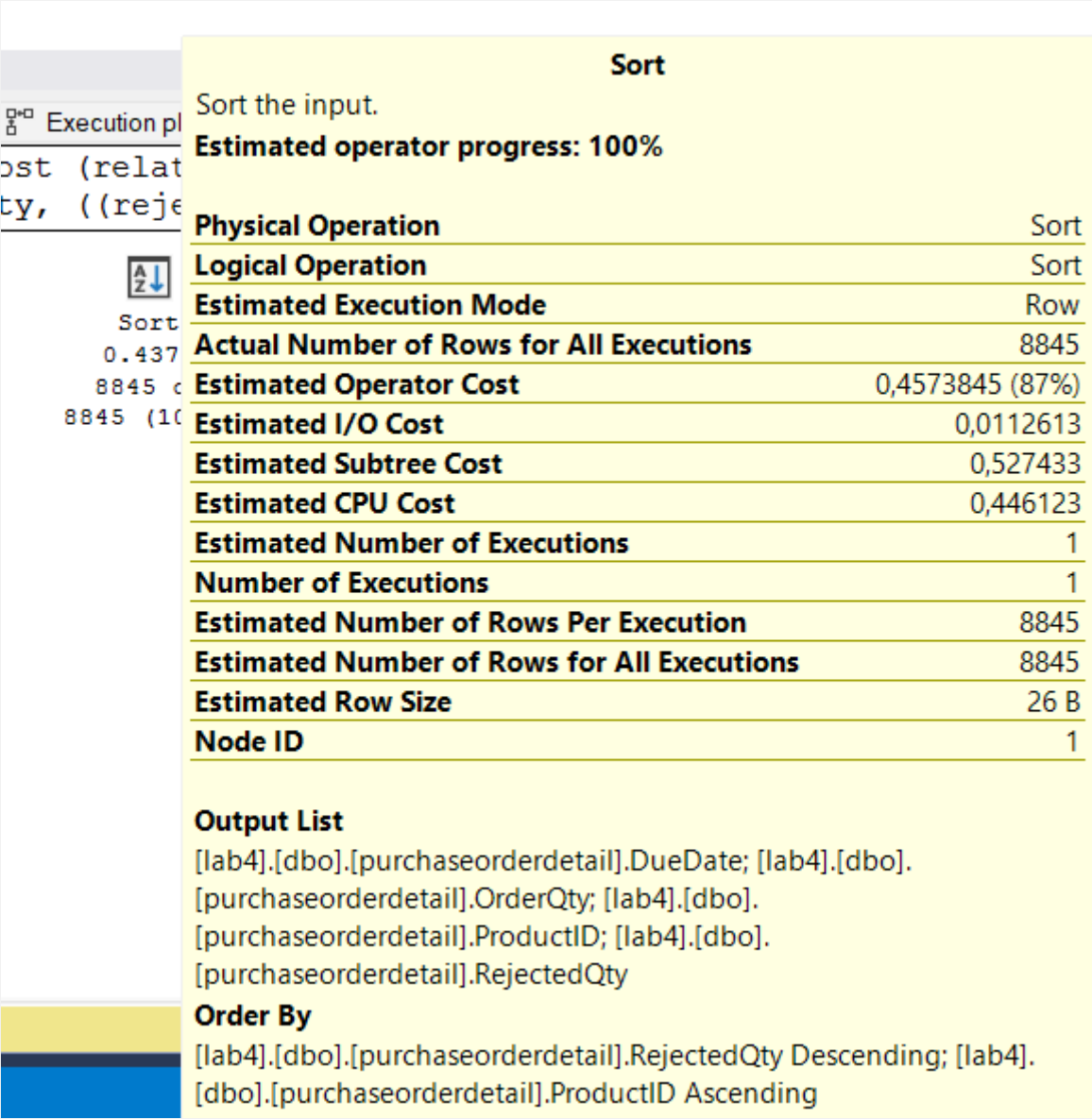
```
select rejectedqty, ((rejectedqty/orderqty)*100) as rejectionrate, productid,
duedate
from purchaseorderdetail
order by rejectedqty desc, productid asc
```

Która część zapytania ma największy koszt?

Wyniki:



Koszt:



Komentarz:

Najbardziej kosztowną operacją jest Sort, na którą przypada około 87% całkowitego kosztu zapytania, na Table Scan przypada około 13%, na Compute Scalar przypada blisko 0%. W planie, najpierw odbywa się operacja Table Scan, która skanuje wszystkie wiersze w tabeli. Potem ma miejsce sortowanie zgodnie z klauzulą order by. Na końcu jest operacja Compute Scalar, która jest używana do obliczenia pola 'rejectionrate' na podstawie innych kolumn.

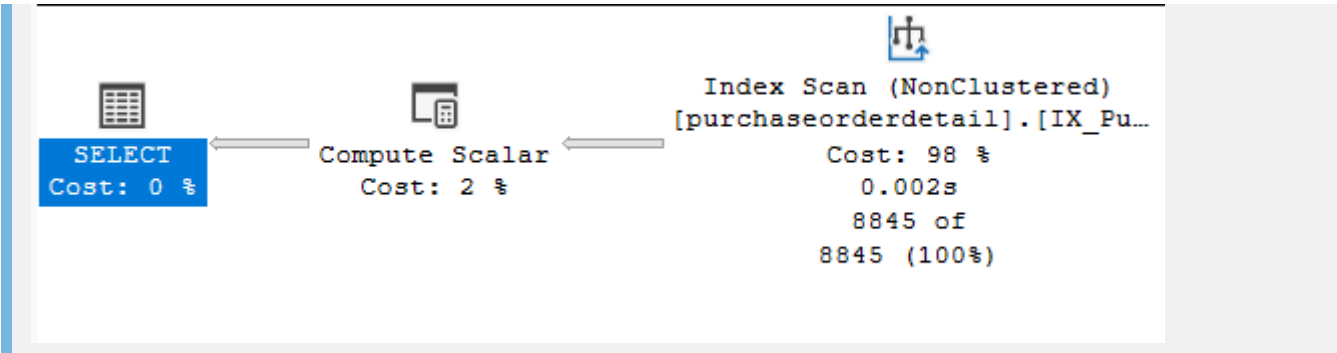
Jaki indeks można zastosować aby zoptymalizować koszt zapytania? Przygotuj polecenie tworzące index.

Komentarz:

Można spróbować nałożyć nieklastrujący indeks na kolumny, według których odbywa się sortowanie.

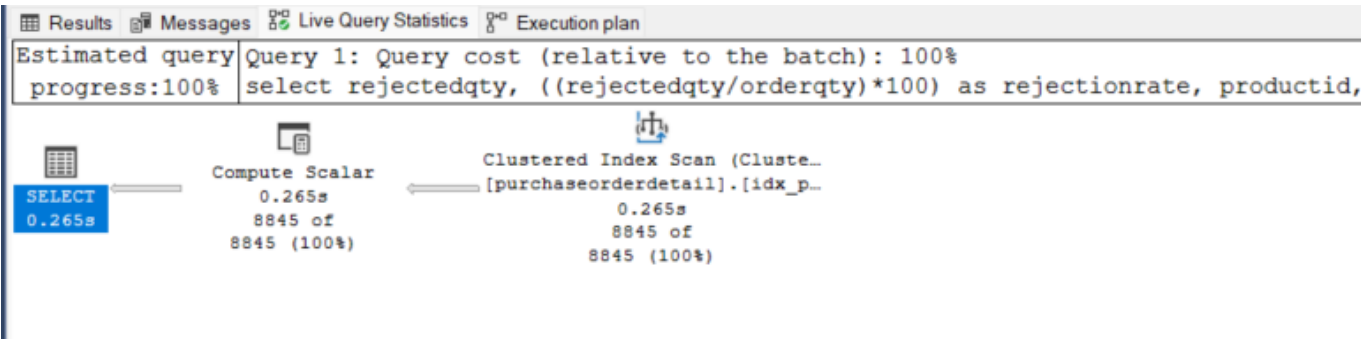
```
CREATE NONCLUSTERED INDEX IX_PurchaseOrderDetail_RejectedQty
ON PurchaseOrderDetail(RejectedQty DESC, ProductID ASC, DueDate, OrderQty);
```

Ponownie wykonaj analizę zapytania:

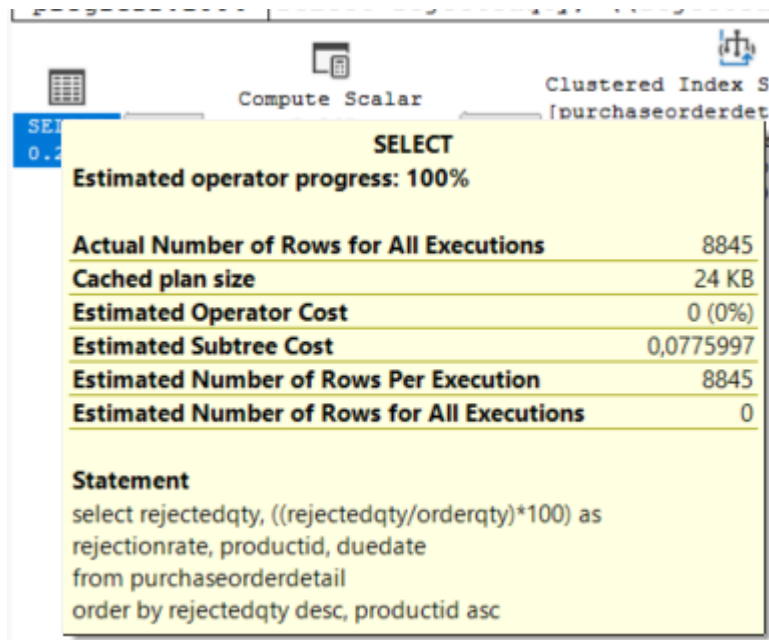


```
CREATE CLUSTERED INDEX idx_purchaseorderdetail_clust
ON purchaseorderdetail (rejectedqty DESC, productid ASC);
```

Plan:



Koszt:



| SELECT | |
|---|-----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 8845 |
| Cached plan size | 24 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,0775997 |
| Estimated Number of Rows Per Execution | 8845 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| select rejectedqty, ((rejectedqty/orderqty)*100) as rejectionrate, productid, due date from purchaseorderdetail order by rejectedqty desc, productid asc | |

Komentarz:

Bez zastosowania żadnego indeksu, oraz dla zwykłego indeksu koszt zapytania wynosił około 0.5, podczas gdy po zastosowaniu indeksu klastrowanego koszt wynosi zaledwie około 0.08

Plan wykonania zapytania zmienił się diametralnie. Najpierw występuje operacja przeszukania indeksu klastrowanego w celu znalezienia odpowiednich wierszy. Potem występuje Compute Scalar do policzenia wartości 'rejectionrate'. Zastosowanie indeksu klastrowanego okazało się najbardziej korzystne biorąc pod uwagę plany i koszty zapytań.

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 '98000' and '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
```

```
--modyfikacja z użyciem konkretnego indeksu
select addressline1, addressline2, city, stateprovinceid, postalcode
from dbo.address WITH(INDEX(Address_PostalCode_1))
where postalcode between '98000' and '99999';
```

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

Wyniki:

Bez zastosowania indeksu:

ResultsMessagesLive Query StatisticsExecution plan

Estimated query progress:100%Query 1: Query cost (relative to the ba
SELECT [addressline1],[addressline2],[c

SELECT

Table Scan
[address]
2638 of
2693 (97%)

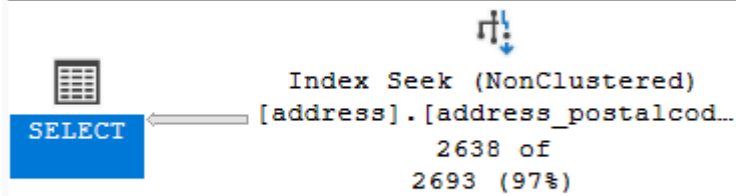
| SELECT | |
|---|----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 2638 |
| Cached plan size | 24 KB |
| Degree of Parallelism | 1 |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,280413 |
| Estimated Number of Rows for All Executions | 0 |
| Estimated Number of Rows Per Execution | 2693,25 |

Statement

SELECT [addressline1],[addressline2],[city],[stateprovinceid],
[postalcode] FROM [dbo].[address] WHERE [postalcode]
>=@1 AND [postalcode]<=@2

Z zastosowaniem indeksu pierwszego:

| | |
|-------------------------------|--|
| Estimated query progress:100% | Query 1: Query cost (relative to the) select addressline1, addressline2, cit |
|-------------------------------|--|

**SELECT**

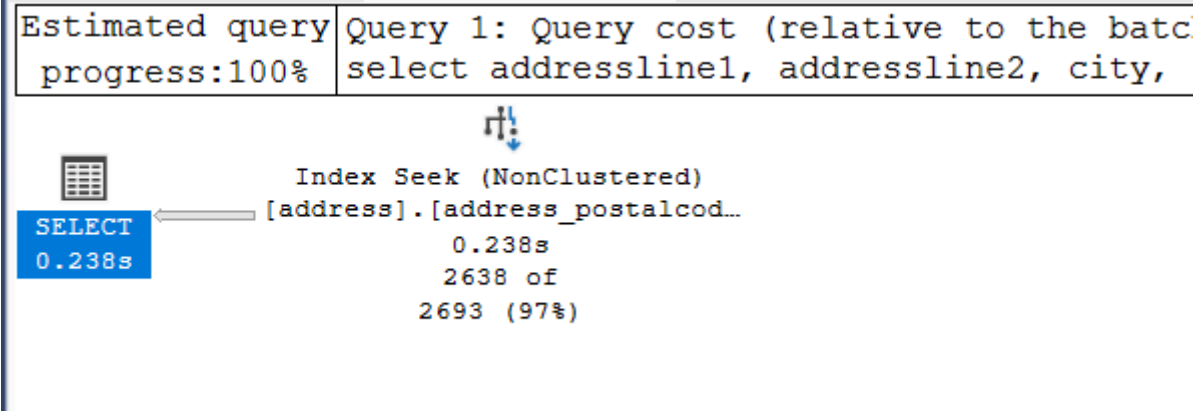
Estimated operator progress: 100%

| | |
|---|-----------|
| Actual Number of Rows for All Executions | 2638 |
| Cached plan size | 24 KB |
| Degree of Parallelism | 1 |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,0284668 |
| Estimated Number of Rows for All Executions | 0 |
| Estimated Number of Rows Per Execution | 2693,25 |

Statement

```
select addressline1, addressline2, city, stateprovinceid,
postalcode
from dbo.address WITH(INDEX(Address_PostalCode_1))
where postalcode between '98000' and '99999'
```

Z zastosowaniem indeksu drugiego:



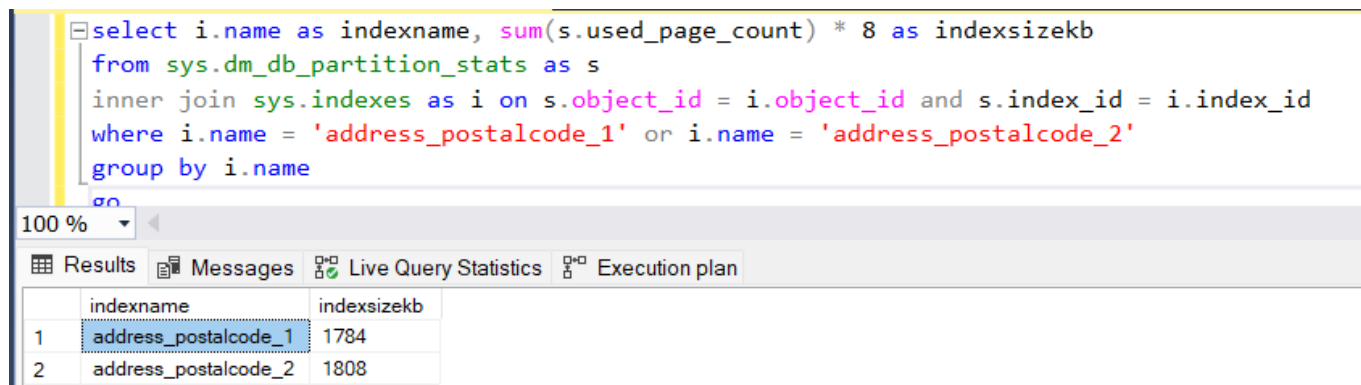
| SELECT | |
|--|-----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 2638 |
| Cached plan size | 24 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,0284668 |
| Estimated Number of Rows Per Execution | 2693,25 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| select addressline1, addressline2, city, stateprovinceid, postalcode from dbo.address WITH(INDEX(Address_PostalCode_2)) where postalcode between '98000' and '99999' | |

Komentarz:

- Różnica w planie zapytania to zmiana operacji Table Scan (skanującej całą tabelę) na bardziej wydajną operację Index Scan (skanującą indeks).
- Zastosowanie indeksu pierwszego i drugiego generuje taki sam plan wykonania i koszt.
- Zastosowanie indeksów zmniejsza koszt o 10 razy w porównaniu z brakiem ich zastosowania.

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
```



The screenshot shows a SQL query in the 'Query Editor' window. The query selects index names and their sizes in KB from the system catalog. Below the query, the 'Results' tab is active, displaying a table with two columns: 'indexname' and 'indexsizekb'. The table contains two rows of data.

```
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
```

| | indexname | indexsizekb |
|---|----------------------|-------------|
| 1 | address_postalcode_1 | 1784 |
| 2 | address_postalcode_2 | 1808 |

Komentarz:

To zapytanie zwraca rozmiar indeksu w kilobajtach. Mnoży sumę 'used_page_count' przez 8, ponieważ każda strona w SQL Server ma rozmiar 8KB.

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

Komentarz:

Większy jest indeks address_postalcode_2

Różnica w fizycznej strukturze obu indeksów to sposób w jaki dane są przechowywane i jakie kolumny są zawarte w samym indeksie.

- Indeks 1 obejmuje jedną kolumnę, a reszta jest uwzględniona za pomocą klauzuli include. Tylko pierwsza kolumna 'postalcode' jest uwzględniona w strukturze głównego drzewa indeksu, pozostałe kolumny są przechowywane oddzielnie. Taki sposób przechowywania może prowadzić do mniejszego zużycia miejsca na dysku, ponieważ główne drzewo indeksu jest mniejsze. Z drugiej strony, dostęp do danych z pozostałych kolumn wymaga odczytu dodatkowej struktury, co może wpłynąć negatywnie na wydajność operacji wyszukiwania.
- Indeks 2 jest utworzony na kilku kolumnach, te kolumny są uwzględnione w głównej strukturze drzewa. Węzły drzewa zawierają wartości tych kolumn oraz wskaźniki do wierszy w tabeli, co umożliwia szybkie wyszukiwanie rekordów. Taki sposób przechowywania może prowadzić do większego zużycia pamięci na dysku, bo główne drzewo jest większe. Jednocześnie ma korzystny wpływ na wydajność operacji wyszukiwania i sortowania, ponieważ dane są dostępne bez potrzeby dostępu do głównej tabeli.

Podsumowując:

W tym przypadku widać, że indeks 2 zajmuje więcej miejsca niż indeks 1, ale plany wykonania zapytań i koszty są takie same. To znaczy, że nic nie zyskujemy stosując indeks nałożony na kilka kolumnach, podczas gdy w klauzuli WHERE odwołujemy się tylko do kolumny 'postalcode'. W tym przypadku bardziej wydajne pamięciowo jest zastosowanie pierwszego indeksu nałożonego na kolumnę 'postalcode' i ewentualne dodanie pozostałych kolumn za pomocą klauzuli INCLUDE.

Eksperyment:

Nałożenie indeksu tylko na kolumnę 'postalcode' i nie uwzględnianie pozostałych kolumn przy pomocy klauzuli INCLUDE.

```
create index address_postalcode_3
on address (postalcode)
go
```

Rozmiar indeksu:

SQLQuery2.sql - D:\... SQLQuery1.sql - D:\...

```
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_3'
group by i.name
go
```

100 %

Results Messages Live Query Statistics Execution plan

| | indexname | indexsizekb |
|---|----------------------|-------------|
| 1 | address_postalcode_3 | 560 |

Plan:

--modyfikacja z użyciem konkretnego indeksu

```
select addressline1, addressline2, city, stateprovinceid, postalcode
from dbo.address WITH(INDEX(Address_PostalCode_3))
where postalcode between '98000' and '99999';
```

100 %

Results Messages Live Query Statistics Execution plan

Estimated query progress:100%

Query 1: Query cost (relative to the batch): 100%
--modyfikacja z użyciem konkretnego indeksu select addressli

SELECT
0.283s

Nested Loops
(Inner Join)
0.283s
2638 of
2693 (97%)

Index Seek (NonClustered)
[address].[address_postalcod...
0.283s
2638 of
2693 (97%)

RID Lookup (Heap)
[address]
0.283s
2638 of
2693 (97%)

Koszt:

22 / 27

| SELECT | |
|---|---------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 2638 |
| Cached plan size | 32 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 1,53078 |
| Estimated Number of Rows Per Execution | 2693,25 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| --modyfikacja z użyciem konkretnego indeksu | |
| select addressline1, addressline2, city, stateprovinceid, | |
| postalcode | |
| from dbo.address WITH(INDEX(Address_PostalCode_3)) | |
| where postalcode between '98000' and '99999' | |

Komentarz:

- Indeks ma rozmiar 560 KB. Jest to ponad 3-krotnie mniej niż pozostałe indeksy 1 i 2.
- Plan wykonania zapytania jest bardziej rozbudowany. Oprócz operacji Index Seek uwzględnia również RID Lookup oraz Nested Loops (Inner Join), które służą do "dołożenia" brakujących kolumn do wyniku (w indeksie jest tylko 'postalcode').
- Całościowy koszt zapytania to około 1.5. Jest to ponad 50 razy większy koszt niż dla indeksów 1 i 2 (koszt wynosił około 0.028). Dodatkowo, jest to o ponad 5 razy większy koszt niż bez zastosowania żadnego indeksu (?)
- Najbardziej kosztowną operacją jest RID Lookup do "doklejenia" pozostałych kolumn do wyniku. Na tą operację przypada 98% całościowego kosztu zapytania.

Podsumowując:

Indeks nieuwzględniający dodatkowych kolumn poza 'postalcode' nie jest odpowiednio dopasowany do zapytania. Nie dość, że nic nie poprawia, to jeszcze zwiększa całościowy koszt wykonania zapytania.

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.

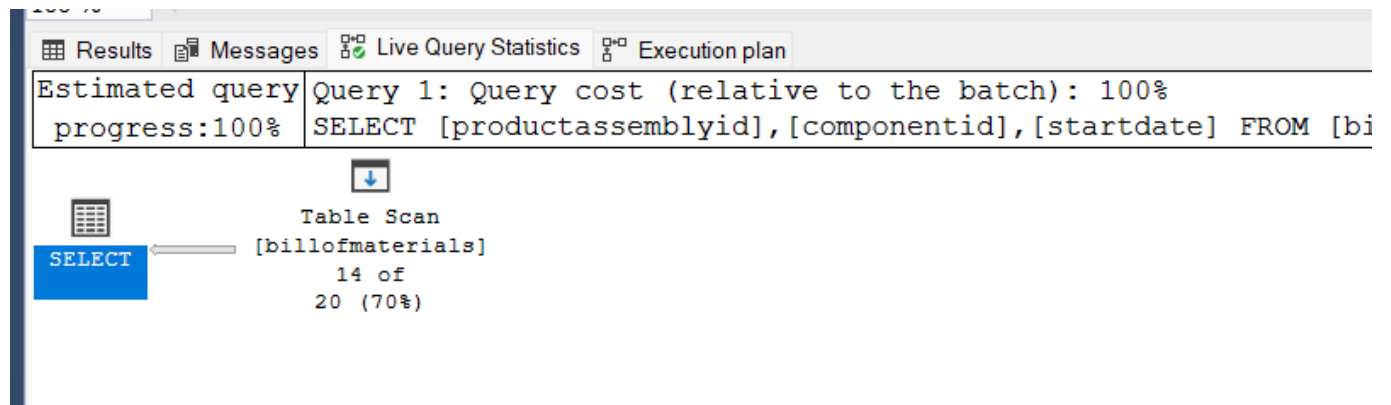
Przeanalizuj plan dla poniższego zapytania:

Czy indeks został użyty? Dlaczego?

Wyniki:

Bez indeksu:

Plan:

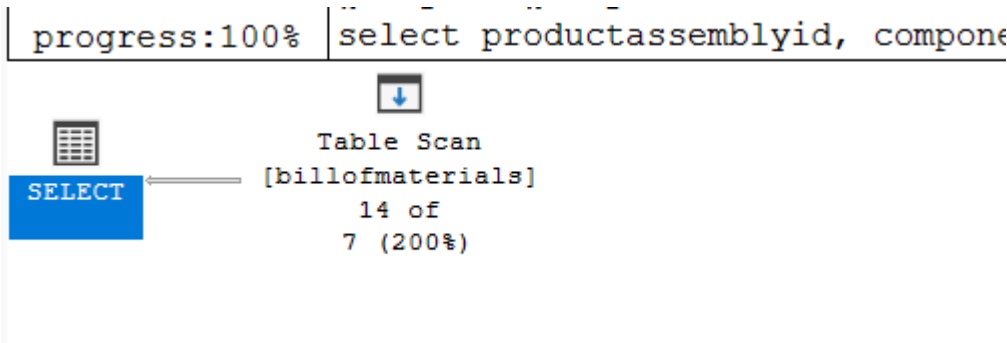


Koszt:

| SELECT | |
|--|----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 14 |
| Cached plan size | 24 KB |
| Degree of Parallelism | 1 |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,020303 |
| Estimated Number of Rows for All Executions | 0 |
| Estimated Number of Rows Per Execution | 20,4719 |
| Statement | |
| SELECT [productassemblyid],[componentid],[startdate] FROM [billofmaterials] WHERE [enddate] IS NOT NULL AND [componentid]=@1 AND [startdate]>=@2 | |

Po nałożeniu indeksu na tabelę:

Plan:



Koszt:

| SELECT | |
|---|----------|
| Estimated operator progress: 100% | |
| Actual Number of Rows for All Executions | 14 |
| Cached plan size | 24 KB |
| Estimated Operator Cost | 0 (0%) |
| Estimated Subtree Cost | 0,020303 |
| Estimated Number of Rows Per Execution | 6,6439 |
| Estimated Number of Rows for All Executions | 0 |
| Statement | |
| select productassemblyid, componentid, startdate from billofmaterials where enddate is not null and componentid = 327 and startdate >= '2010-08-05' | |

Komentarz:

Zarówno plan wykonania zapytania jak i koszt jest taki sam przed i po wprowadzeniu indeksu. To wskazuje, że indeks nie jest używany do optymalizacji powyższego zapytania.

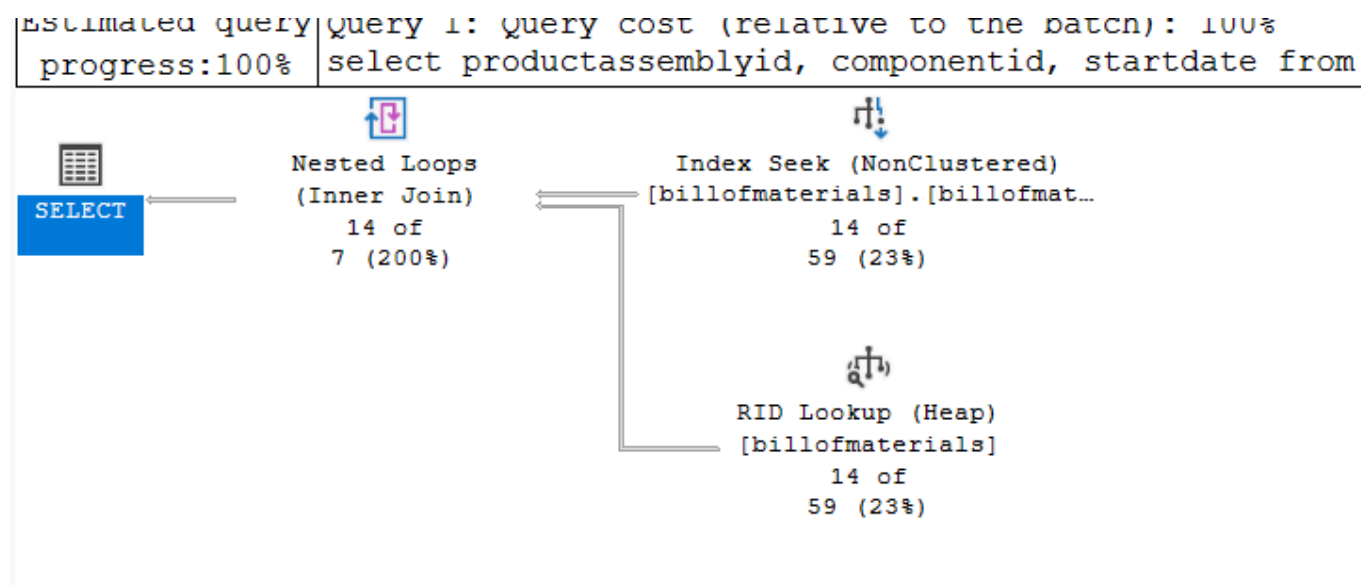
Spróbuj wymusić indeks. Co się stało, dlaczego takie zachowanie?

Wyniki:

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

Z wymuszeniem indeksu:

Plan:



Koszt:

| SELECT | |
|---|-----------|
| Estimated operator progress: 100% | |
| 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 |
| Estimated Number of Rows Per Execution | 6,6439 |
| Statement | |
| select productassemblyid, componentid, startdate | |
| from billofmaterials with (INDEX(billofmaterials_cond_idx)) | |
| where enddate is not null | |
| and componentid = 327 | |
| and startdate >= '2010-08-05' | |

Komentarz:

Koszt zapytania z wymuszeniem użycia indeksu jest około 3 razy większy niż bez użycia indeksu. W planie wykonania zapytania pojawiły się operacje Index Scan do skanowania indeksu, RID Lookup do "dołożenia" pozostałych kolumn, oraz Nested Loops używany razem z RID Lookup do dołożenia dodatkowych kolumn. W przypadku tego zapytania bardziej optymalne okazało się skanowanie całej tabeli, czyli domyślny plan bez zastosowania indeksu.

Dlaczego?

Z związku z 3-krotnie większym kosztem, być może optymalizator "stwierdził", że nic nie zyskamy wykorzystując indeks i bardziej optymalne będzie przeskanowanie całej tabeli.

Punktacja:

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