

Indeksy, optymalizator

Lab 5

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

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:

Zapytanie nr1: czas: 1/100 sekundy koszt: ~0.14

Zapytanie nr2: czas: 1/100 sekundy koszt: ~0.14

W każdym z zapytań wykonano full table scan

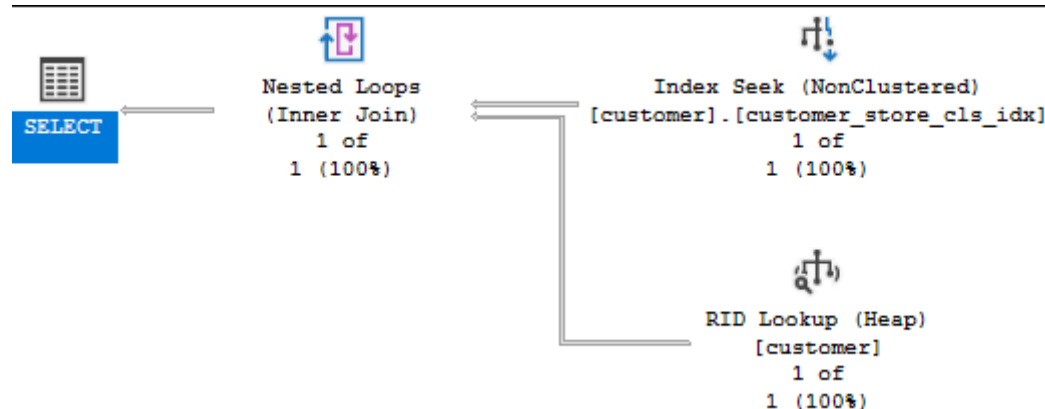
Dodaj indeks:

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

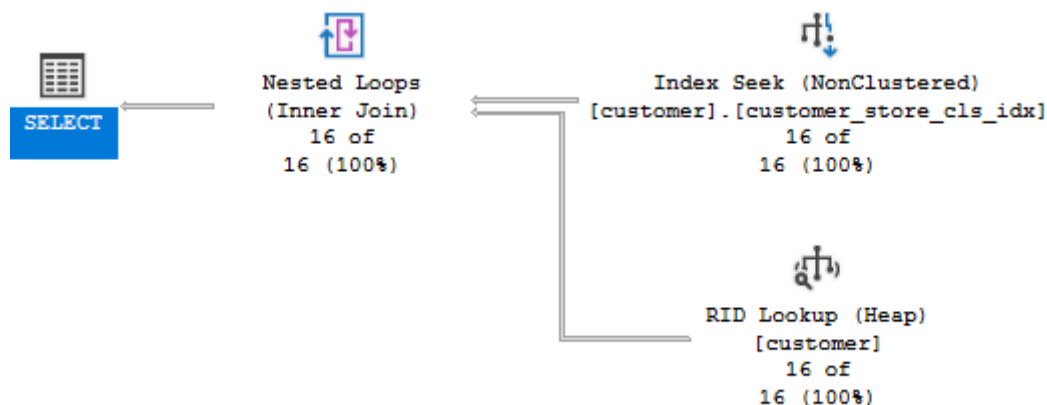
Jak zmienił się plan i czas? Czy jest możliwość optymalizacji?

Po dodaniu indeksu:

Zapytanie nr1: czas: 1/100 sekundy koszt: ~0.0065



Zapytanie nr2: czas: 1/100 sekundy cost: ~0.05



Jak widać zamiast full table scan, mamy index scan a następnie RID lookup

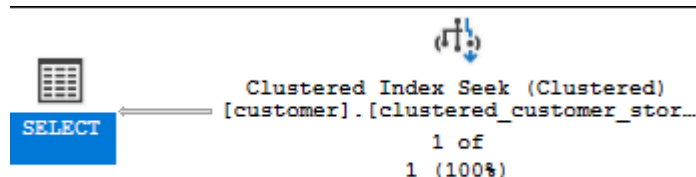
O ile czas wykonania znacząco się nie różni, to możemy zauważyć znaczące mniejszenie kosztu wykonania.

Dodaj indeks klastrowany:

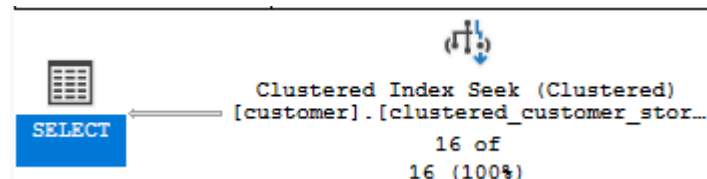
```
create clustered index clustered_customer_store_cls_idx on customer(storeid)
```

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

Zapytanie nr1: czas: 1/100 sekundy koszt: ~0.003



Zapytanie nr2: czas: 1/100 sekundy koszt: ~0.003



W planie mamy tylko clustered index seek. W przypadku zapytania nr1 nie widzimy dużej korzyści z zastosowania clustered index. Różnice widać za to w przypadku wybieraniu zakresu. Ponieważ clustered index fizycznie sortuje dane na dysku, kolejne wiersze z zakresu są umieszczone blisko siebie.

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:

SQL Query:

```

,lastname
,firstname
,modifieddate
into person
from adventureworks2017.f

select top 1000 * from person
select * from [person]
select * from [person]
select * from [person]

```

Query 1: Query cost (re)

SELECT * FROM [person]

Table Scan

Cost: 100

0.002s

1 of 2 (50%)

Query executed successfully.

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 Read	19972
Actual Number of Rows for All Executions	1
Actual Number of Batches	0
Estimated I/O Cost	0.153574
Estimated Operator Cost	0.175622 (100%)
Estimated Subtree Cost	0.175622
Estimated CPU Cost	0.0220477
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	1.86667
Estimated Number of Rows Per Execution	1.86667
Estimated Number of Rows to be Read	19972
Estimated Row Size	186 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	0
Predicate	
[xyz].[dbo].[person].[lastname]=CONVERT_IMPLICIT(nvarchar(4000),[@1],0)	
Object	
[xyz].[dbo].[person]	
Output List	
[xyz].[dbo].[person].businessentityid, [xyz].[dbo].[person].persontype, [xyz].[dbo].[person].namestyle, [xyz].[dbo].[person].title, [xyz].[dbo].[person].firstname, [xyz].[dbo].[person].middlename, [xyz].[dbo].[person].lastname, [xyz].[dbo].[person].suffix, [xyz].[dbo].[person].emailpromotion, [xyz].[dbo].[person].rowguid, [xyz].[dbo].	

SQL Query:

```

,lastname
,firstname
,modifieddate
into person
from adventureworks2017.f

select top 1000 * from person
select * from [person]
select * from [person]
select * from [person]

```

Query 1: Query cost (re)

SELECT * FROM [person]

Table Scan

Cost: 100

0.002s

1 of 2 (50%)

Query executed successfully.

Table Scan	
Scan rows from a table.	
Estimated operator progress: 100%	
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	1
Actual Number of Rows Read	19972
Actual Number of Batches	0
Estimated Operator Cost	0.175622 (100%)
Estimated I/O Cost	0.153574
Estimated Subtree Cost	0.175622
Estimated CPU Cost	0.0220477
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows to be Read	19972
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	147 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	0
Predicate	
[xyz].[dbo].[person].[lastname]=CONVERT_IMPLICIT(nvarchar(4000),[@1],0)	
Object	
[xyz].[dbo].[person]	
Output List	
[xyz].[dbo].[person].businessentityid, [xyz].[dbo].[person].persontype, [xyz].[dbo].[person].namestyle, [xyz].[dbo].[person].title, [xyz].[dbo].[person].firstname, [xyz].[dbo].[person].middlename, [xyz].[dbo].[person].lastname, [xyz].[dbo].[person].suffix, [xyz].[dbo].[person].emailpromotion, [xyz].[dbo].[person].rowguid, [xyz].[dbo].	

SELECT [person]
1 of
1 (100%)

Actual Rewinds
Actual Rewinds 0
Ordered False
Node ID 0

Predicate
[xyz].[dbo].[person].[lastname]=CONVERT_IMPLICIT(nvarchar(4000),
[@1],0) AND [xyz].[dbo].[person].[firstname]=CONVERT_IMPLICIT
(nvarchar(4000),[@2],0)

Object
[xyz].[dbo].[person]

Output List
[xyz].[dbo].[person].businessentityid, [xyz].[dbo].[person].persontype,
[xyz].[dbo].[person].namestyle, [xyz].[dbo].[person].title, [xyz].[dbo].
[person].firstname, [xyz].[dbo].[person].middlename, [xyz].[dbo].
[person].lastname, [xyz].[dbo].[person].suffix, [xyz].[dbo].
[person].emailpromotion, [xyz].[dbo].[person].rowguid, [xyz].[dbo].
[person].modifieddate

Query executed successfully.

Table Scan
Scan rows from a table.
Estimated operator progress: 100%

Physical Operation Table Scan
Logical Operation Table Scan
Estimated Execution Mode Row
Storage RowStore
Actual Number of Rows for All Executions 1
Estimated I/O Cost 0.153574
Estimated Operator Cost 0.175622 (100%)
Estimated CPU Cost 0.0220477
Estimated Subtree Cost 0.175622
Number of Executions 1
Estimated Number of Executions 1
Estimated Number of Rows for All Executions 13.5714
Estimated Number of Rows Per Execution 13.5714
Estimated Number of Rows to be Read 19972
Estimated Row Size 186 B
Ordered False
Node ID 0

Predicate
[xyz].[dbo].[person].[firstname]=CONVERT_IMPLICIT(nvarchar(4000),
[@1],0)

Object
[xyz].[dbo].[person]

Output List
[xyz].[dbo].[person].businessentityid, [xyz].[dbo].[person].persontype,
[xyz].[dbo].[person].namestyle, [xyz].[dbo].[person].title, [xyz].[dbo].
[person].firstname, [xyz].[dbo].[person].middlename, [xyz].[dbo].
[person].lastname, [xyz].[dbo].[person].suffix, [xyz].[dbo].
[person].emailpromotion, [xyz].[dbo].[person].rowguid, [xyz].[dbo].
[person].modifieddate

Query executed successfully.

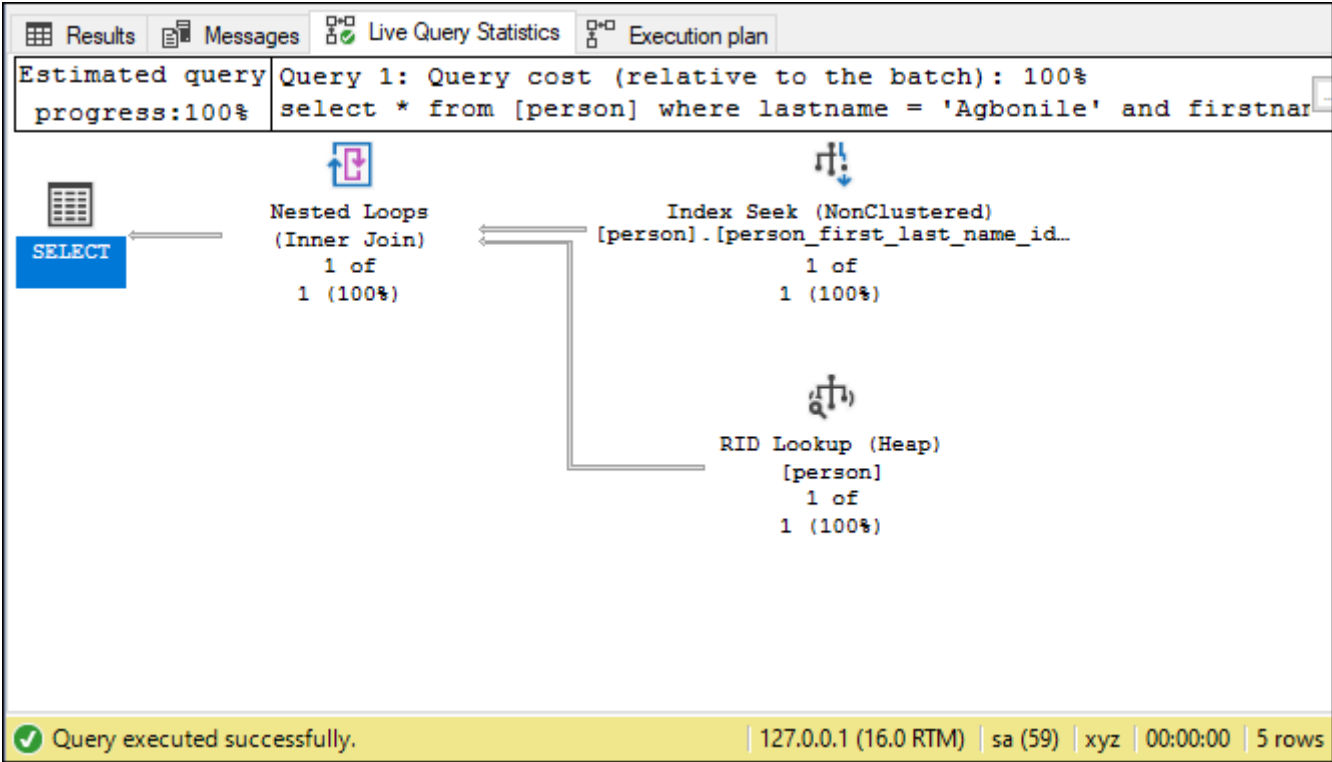
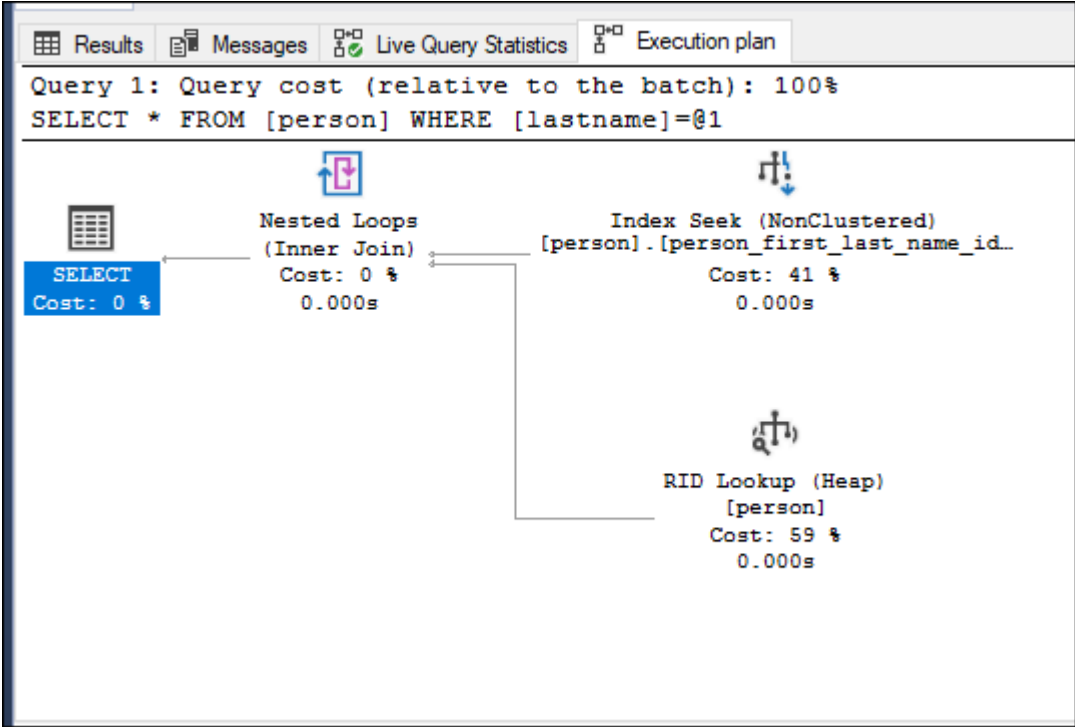
Dla query szukającym po imieniu zwrócone zostało 5 wierszy. W każdym z przypadków system bazy danych jest zmuszony do przeskanowania całej tabeli w celu znalezienia wierszy spełniających kryterium.

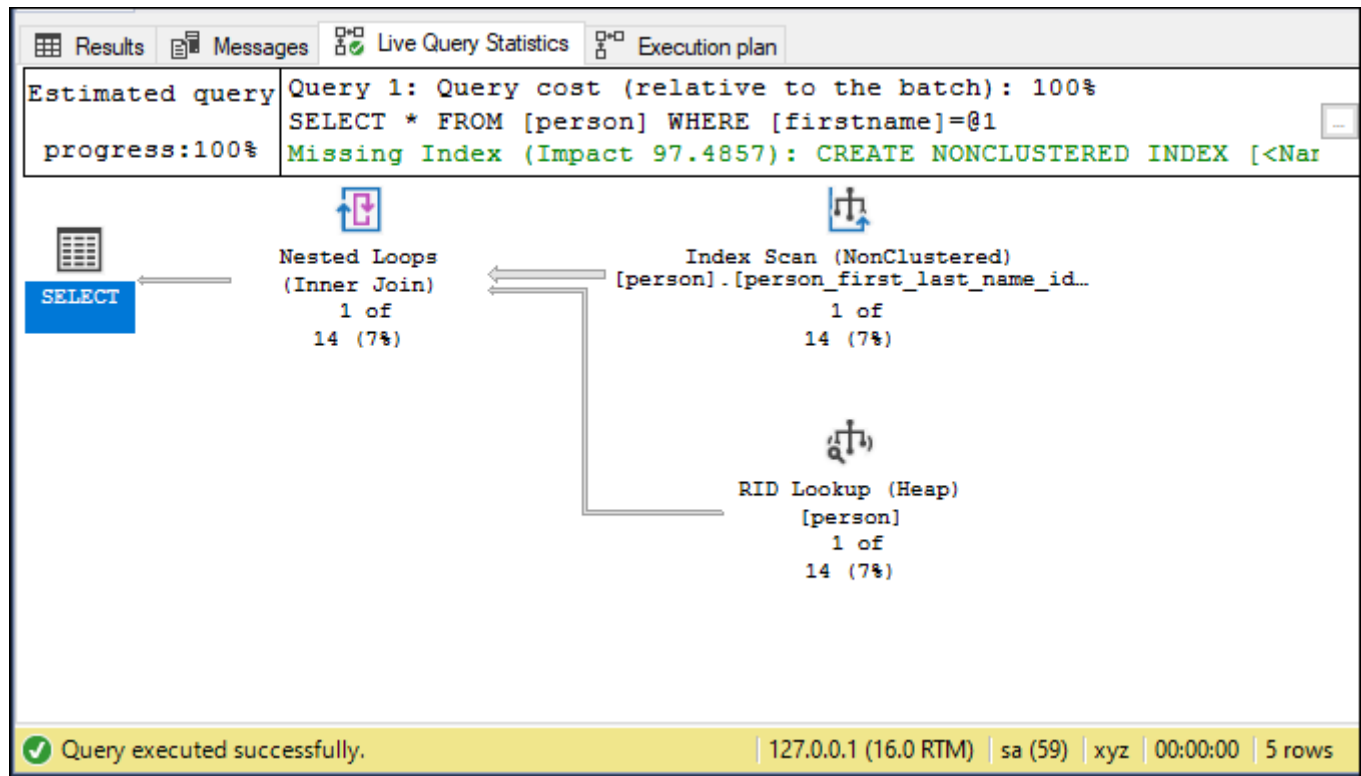
Przygotuj indeks obejmujący te zapytania:

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

Sprawdź plan zapytania. Co się zmieniło?

Wyniki:





Jak widać teraz najpierw używany jest indeks a następnie RID lookup. RID lookup jest wykonywany w celu uzyskania dodatkowych informacji o wierszu, które nie są przechowywane w indeksie.

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:

```
select * from [person] where lastname = 'Price';
select * from [person] where lastname = 'Price' and firstname = 'Angela' ;
select * from [person] where firstname = 'Angela';
```

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

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

The query execution plan for the query 'SELECT * FROM [person] WHERE [lastname]=@1 AND [firstname]=@2' is shown. The plan consists of three main components: a 'SELECT' operator, a 'Nested Loops (Inner Join)' operator, and an 'Index Seek (NonClustered)' operator. The 'SELECT' operator is the root of the plan, with a cost of 0% and 0.000s. It feeds into the 'Nested Loops (Inner Join)' operator, which also has a cost of 0% and 0.000s. The 'Nested Loops' operator has two inputs: one from the 'SELECT' operator and another from the 'Index Seek' operator. The 'Index Seek (NonClustered)' operator has a cost of 50% and 0.000s, and it feeds into the 'Nested Loops' operator. The 'Index Seek' operator is labeled with the index '[person].[person_first_last_name_id...]'.

SELECT
Cost: 0 %
0.000s

Nested Loops
(Inner Join)
Cost: 0 %
0.000s

Index Seek (NonClustered)
[person].[person_first_last_name_id...]
Cost: 50 %
0.000s

RID Lookup (Heap)
[person]
Cost: 50 %
0.000s

Query executed successfully. | 127.0.0.1 (16.0 RTM) | sa (59) | xyz | 00:00:01 | 5 rows

```

select * from [person] wh
select * from [person] wh
select * from [person] wh

create index person_first
on person(lastname, first

```

Table Scan

Scan rows from a table.

Estimated operator progress: 100%

Physical Operation	Table Scan
Logical Operation	Table Scan
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	50
Estimated I/O Cost	0.153495
Estimated Operator Cost	0.175622 (100%)
Estimated CPU Cost	0.0221262
Estimated Subtree Cost	0.175622
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	50
Estimated Number of Rows Per Execution	50
Estimated Number of Rows to be Read	19972
Estimated Row Size	186 B
Ordered	False
Node ID	0

Predicate
[xyz].[dbo].[person].[firstname]=N'Angela'

Object
[xyz].[dbo].[person]

Output List
[xyz].[dbo].[person].businessentityid, [xyz].[dbo].[person].persontype, [xyz].[dbo].[person].namestyle, [xyz].[dbo].[person].title, [xyz].[dbo].[person].firstname, [xyz].[dbo].[person].middlename, [xyz].[dbo].[person].lastname, [xyz].[dbo].[person].suffix, [xyz].[dbo].[person].emailpromotion, [xyz].[dbo].[person].rowguid, [xyz].[dbo].[person].modifieddate

100 %

Results Messages Live Q

Estimated query progress:100%

Query 1: select * Missing I:

Table Scan [person] 50 of 50 (100%)

SELECT

Query executed successfully.

Dla query szukającym po imieniu zwrócone zostało 86 wierszy.

Do realizacji zapytań używających tylko jednego pola w klauzuli **WHERE** indeks nie jest wykorzystywany. Prawdopodobnie optymalizator mając do dyspozycji statystyki tabeli i indeksu stwierdził, że znalezienie wszystkich wierszy spełniających klauzulę **WHERE** użycie dostępnego indeksu nie jest opłacalne. System baz danych musi utrzymywać statystyki rozkładu danych z których wynika, że **Angele** jest dużo popularniejszym imieniem od **Osarumwense**.

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:

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

	rejectedqty	rejectionrate	productid	duedate
1	1250.00	100.00000000	325	2013-09-12 00:00:00.000
2	1250.00	100.00000000	325	2014-03-11 00:00:00.000
3	1250.00	100.00000000	326	2014-03-11 00:00:00.000
4	1250.00	100.00000000	326	2013-09-12 00:00:00.000
5	550.00	100.00000000	317	2012-05-16 00:00:00.000
6	550.00	100.00000000	317	2012-10-05 00:00:00.000
7	550.00	100.00000000	317	2014-02-06 00:00:00.000
8	550.00	100.00000000	317	2014-04-18 00:00:00.000
9	550.00	100.00000000	318	2014-04-18 00:00:00.000
10	550.00	100.00000000	318	2014-02-06 00:00:00.000
11	550.00	100.00000000	318	2012-10-05 00:00:00.000
12	550.00	100.00000000	318	2012-05-16 00:00:00.000
13	550.00	100.00000000	319	2013-05-08 00:00:00.000
14	550.00	100.00000000	332	2013-06-12 00:00:00.000
15	550.00	100.00000000	351	2011-12-29 00:00:00.000
16	550.00	100.00000000	351	2013-10-02 00:00:00.000
17	550.00	100.00000000	352	2013-10-02 00:00:00.000
18	550.00	100.00000000	352	2011-12-29 00:00:00.000
19	550.00	100.00000000	355	2014-01-06 00:00:00.000
20	550.00	100.00000000	488	2013-09-09 00:00:00.000
21	550.00	100.00000000	489	2013-09-09 00:00:00.000
22	550.00	100.00000000	507	2012-03-22 00:00:00.000
23	550.00	100.00000000	507	2014-06-04 00:00:00.000
24	550.00	100.00000000	508	2014-06-04 00:00:00.000
25	550.00	100.00000000	508	2012-03-22 00:00:00.000
26	550.00	100.00000000	510	2012-03-23 00:00:00.000
27	550.00	100.00000000	511	2012-03-23 00:00:00.000

100 %

Results Messages Live Query Statistics Execution plan

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

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

SELECT
Cost: 0 %

Compute Scalar
Cost: 0 %

Sort
Cost: 85 %
0.024s
8845 of
8845 (100%)

Table Scan
[purchaseorderdetail]
Cost: 15 %
0.005s
8845 of
8845 (100%)

100 %

Results Messages Live Query Statistics Execution plan

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

select rejectedqty, ((rejectedqty/orderqty)*100) as rejectionrate, productid, duedate from purchaseorderdetail

```

graph RL
    TS[Table Scan  
[purchaseorderdetail]  
8845 of 8845 (100%)] --> S[Sort  
8845 of 8845 (100%)]
    S --> CS[Compute Scalar  
8845 of 8845 (100%)]
    CS --> SEL[SELECT]
  
```

Table Scan

Scan rows from a table.
Estimated operator progress: 100%

Physical Operation	Table Scan
Logical Operation	Table Scan
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	8845
Estimated I/O Cost	0.0720924
Estimated Operator Cost	0.0819004 (15%)
Estimated CPU Cost	0.009808
Estimated Subtree Cost	0.0819004
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	8845
Estimated Number of Rows Per Execution	8845
Estimated Number of Rows to be Read	8845
Estimated Row Size	26 B
Ordered	False
Node ID	2

Object

[master].[dbo].[purchaseorderdetail]

Output List

[master].[dbo].[purchaseorderdetail].DueDate, [master].[dbo].[purchaseorderdetail].OrderQty, [master].[dbo].[purchaseorderdetail].ProductID, [master].[dbo].[purchaseorderdetail].RejectedQty

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

Można dodać indeks na kolumny `rejectedqty` oraz `productid` wskazując kolejność sortowania tak jak w zapytaniu wraz z sekcją `include`, w której uwzględnimy kolumny, których dodatkowo potrzebujemy, czyli `orderqty` oraz `duedate`, aby nie trzeba było realizować dodatkowych operacji pobierających te wartości.

13 / 26

```
duedate)
```

Ponownie wykonaj analizę zapytania:

Wyniki:

```

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

create index person_firspurchaseorderdetail_rejectedqty_productid_orderqty_duedate
on purchaseorderdetail (rejectedqty desc, productid asc) include (orderqty, duedate)

```

100 %

Results Messages Live Query Statistics Execution plan

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

select rejectedqty, ((rejectedqty/orderqty)*100) as rejectionrate, productid, duedate from purchaseorderdetail

```

graph RL
    A[Index Scan (NonClustered) [purchaseorderdetail]  
Cost: 9  
0.005] --> B[Compute Scalar  
Cost: 2 %]
    B --> C[SELECT  
Cost: 0 %]

```

Index Scan (NonClustered)	
Scan a nonclustered index, entirely or only a range.	
Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows for All Executions	8845
Actual Number of Rows Read	8845
Actual Number of Batches	0
Estimated Operator Cost	0.0396782 (98%)
Estimated I/O Cost	0.0297917
Estimated CPU Cost	0.0098865
Estimated Subtree Cost	0.0396782
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows to be Read	8845
Estimated Number of Rows for All Executions	8845
Estimated Number of Rows Per Execution	8845
Estimated Row Size	26 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	1
Object	
[master].[dbo].[purchaseorderdetail].	
[person_firspurchaseorderdetail_rejectedqty_productid_orderqty_duedate]	
Output List	
[master].[dbo].[purchaseorderdetail].DueDate, [master].[dbo].	
[purchaseorderdetail].OrderQty, [master].[dbo].	
[purchaseorderdetail].ProductID, [master].[dbo].	
[purchaseorderdetail].RejectedQty	

Zniknęła najbardziej kosztowna część zapytania obejmująca sortowanie dzięki zastosowaniu indeksu na kolumny po których odbywało się sortowanie, gdyż nie jest ono już potrzebne, bo jest zapewnione przez realizację indeksu. Teraz najbardziej kosztowną operacją jest odczyt wartości. Zmniejszył się także koszt operacji wejścia/wyjścia. Zmalał całkowity koszt i czas 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**:

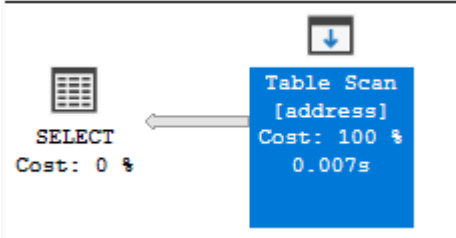
Wyniki:

WYNIKI START

Wynik zapytania:

	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

Plan:



Koszt: ~0.27

Wykonujemy pełny skan tabeli

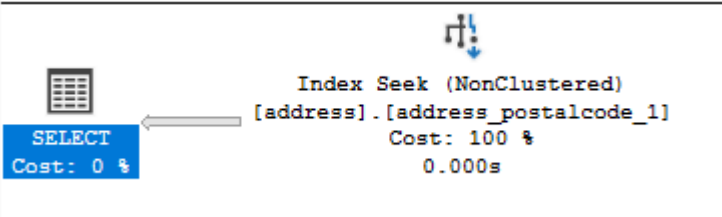
Zastosowanie indexu 1

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

Wynik zapytania:

	addressline1	addressline2	city	stateprovinceid	postalcode
1	225 South 314th Street	NULL	Federal Way	79	98003
2	25915 140th Ave Ne	NULL	Bellevue	79	98004
3	2284 Azalea Avenue	NULL	Bellevue	79	98004
4	108 Lakeside Court	NULL	Bellevue	79	98004
5	5863 Sierra	NULL	Bellevue	79	98004
6	8684 Military East	NULL	Bellevue	79	98004
7	7270 Pepper Way	NULL	Bellevue	79	98004
8	6058 Hill Street	# 4	Bellevue	79	98004
9	1648 Eastgate Lane	NULL	Bellevue	79	98004
10	3454 Bel Air Drive	NULL	Bellevue	79	98004
11	3067 Maya	NULL	Bellevue	79	98004
12	3197 Thomhill Place	NULL	Bellevue	79	98004
13	3919 Pinto Road	NULL	Bellevue	79	98004

Plan:



Koszt: ~0.028

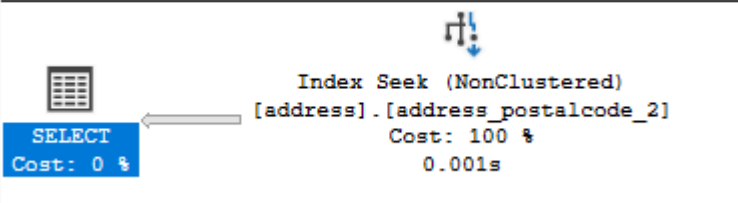
Zastosowanie indexu 2

```
select addressline1, addressline2, city, stateprovinceid, postalcode
from address
WITH(INDEX(Address_PostalCode_2))
where postalcode between '98000' and '99999'
```

Wynik zapytania:

	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 Thornhill 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

Plan:



Koszt: ~0.028

Nie widać różnicy pomiędzy poszczególnymi zapytaniami, koszt i plan jest taki sam dla obu indeksów.

Różnica w indeksach polega na na sposobie przetrzymywania wartości w b-drzewie. Klucze w include() są przechowywane w liściach b-drzewa, w drugim przypadku wszystkie wartości są przechowywane w każdym z nodeów.

WYNIKI END

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 START

Wyniki:

	indexname	indexsizekb
1	address_postalcode_1	1784
2	address_postalcode_2	1808

Jak widać rozmiar indeksów jest bardzo podobny. Może za to wystąpić różnica w ilości poziomów drzewa. Ponieważ, w nie-liściach nie przechowujemy kolumn zawartych w include(), to jesteśmy w stanie zmieścić więcej kluczy w jednym nodzie. Skutkuje to bardziej płaskim drzewem ale o podobnym fizycznym rozmiarze.

WYNIKI END

Zadanie 5 – Indeksy z filtrami

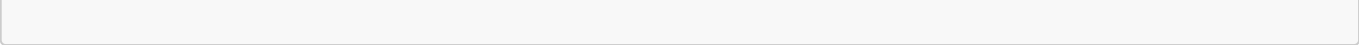
Celem zadania jest poznanie indeksów z filtrami.

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'
```



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

100 %

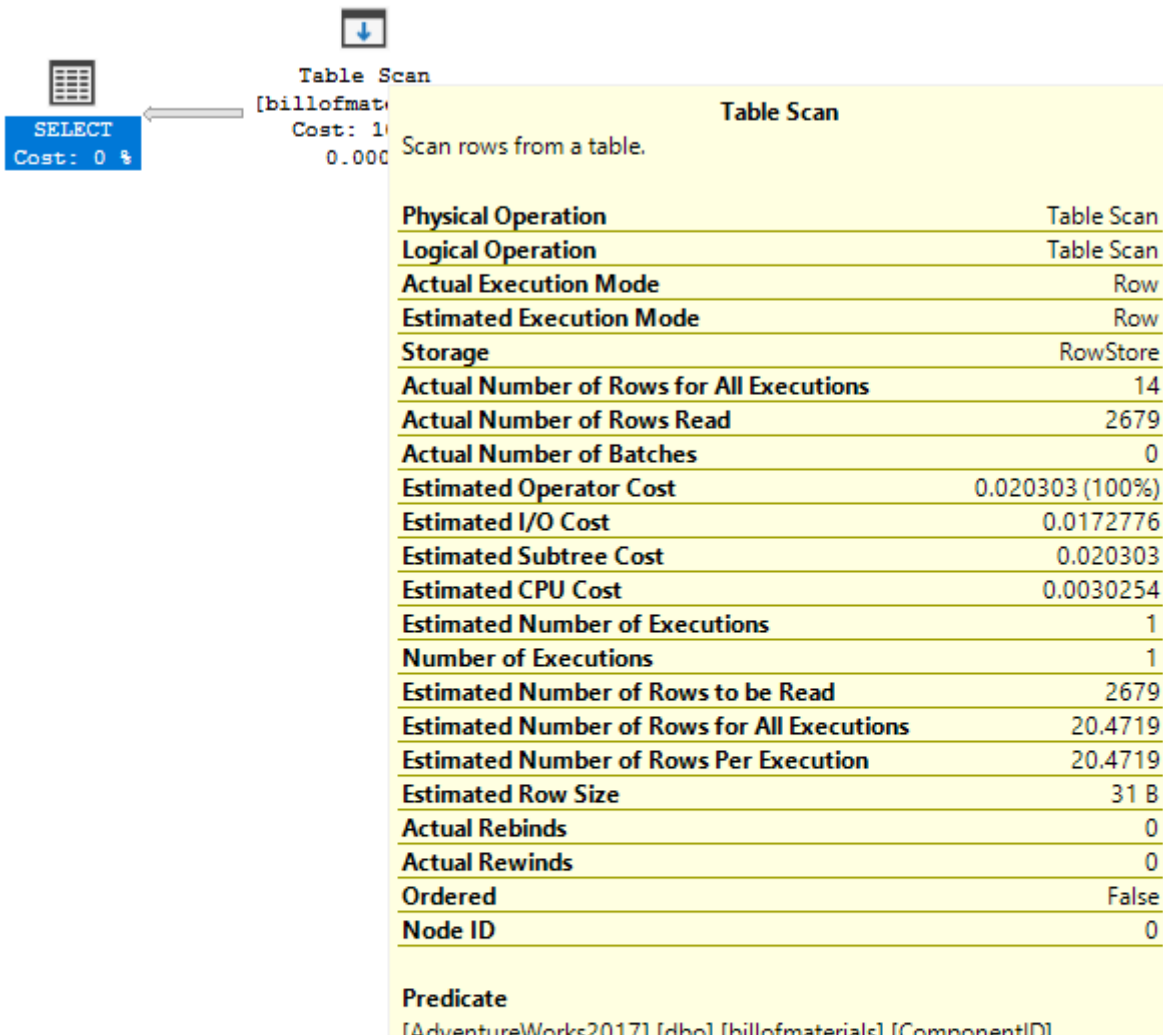
Results Messages


	productassemblyid	componentid	startdate
1	718	327	2010-08-05 00:00:00.000
2	725	327	2010-08-05 00:00:00.000
3	729	327	2010-08-05 00:00:00.000
4	731	327	2010-08-05 00:00:00.000
5	735	327	2010-08-05 00:00:00.000
6	738	327	2010-08-05 00:00:00.000
7	742	327	2010-08-05 00:00:00.000
8	745	327	2010-08-05 00:00:00.000
9	832	327	2010-08-05 00:00:00.000
10	840	327	2010-08-05 00:00:00.000
11	890	327	2010-08-05 00:00:00.000
12	898	327	2010-08-05 00:00:00.000
13	902	327	2010-08-05 00:00:00.000
14	920	327	2010-08-05 00:00:00.000

Results Messages Live Query Statistics Execution plan

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

```
SELECT [productassemblyid],[componentid],[startdate] FROM [billofmateri
```



 Query executed successfully.

```
[AdventureWorks2017].[dbo].[billofmaterials].[componentid]
=CONVERT_IMPLICIT(int,[@1],0) AND [AdventureWorks2017].[dbo].
[billofmaterials].[StartDate]>=CONVERT_IMPLICIT(datetime,[@2],0)
AND [AdventureWorks2017].[dbo].[billofmaterials].[EndDate] IS NOT
NULL
```

Object
[AdventureWorks2017].[dbo].[billofmaterials]

Output List
[AdventureWorks2017].[dbo].[billofmaterials].ProductAssemblyID,
[AdventureWorks2017].[dbo].[billofmaterials].ComponentID,
[AdventureWorks2017].[dbo].[billofmaterials].StartDate

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:

```

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

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

```

100 %

Results Messages Live Query Statistics Execution plan

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

SELECT [productassemblyid],[componentid],[startdate] FROM [billofmater:

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 Read 2679

Actual Number of Rows for All Executions 14

Actual Number of Batches 0

Estimated I/O Cost 0.0171991

Estimated Operator Cost 0.020303 (100%)

Estimated Subtree Cost 0.020303

Estimated CPU Cost 0.0031039

Estimated Number of Executions 1

Number of Executions 1

Estimated Number of Rows for All Executions 6.6439

Estimated Number of Rows Per Execution 6.6439

Estimated Number of Rows to be Read 2679

Estimated Row Size 31 B

Actual Rebinds 0

Actual Rewinds 0

Ordered False

Node ID 0

Predicate

[AdventureWorks2017].[dbo].[billofmaterials].[ComponentID]=(327)
AND [AdventureWorks2017].[dbo].[billofmaterials].[StartDate]
>='2010-08-05 00:00:00.000' AND [AdventureWorks2017].[dbo].
[billofmaterials].[EndDate] IS NOT NULL

Object

[AdventureWorks2017].[dbo].[billofmaterials]

Output List

[AdventureWorks2017].[dbo].[billofmaterials].ProductAssemblyID,
[AdventureWorks2017].[dbo].[billofmaterials].ComponentID,
[AdventureWorks2017].[dbo].[billofmaterials].StartDate

Query executed successfully.

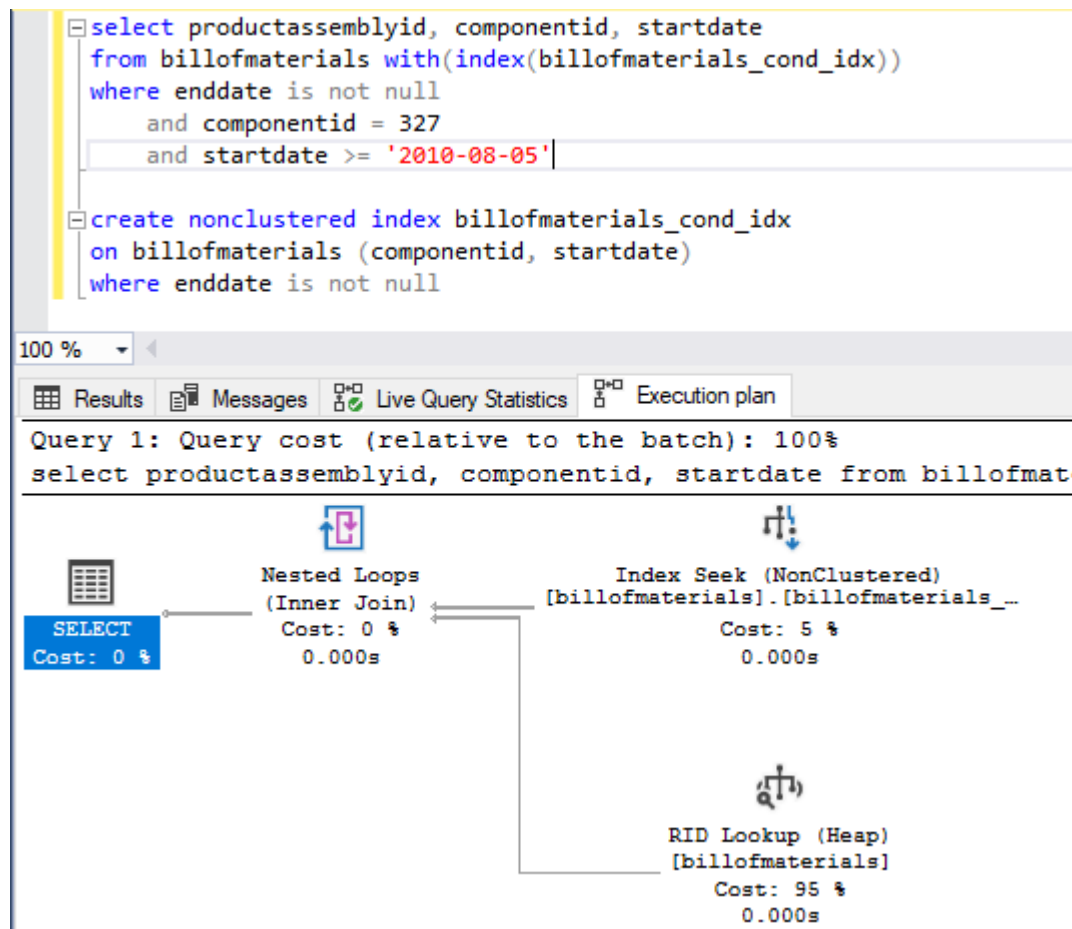
Jak widać wynik analizy zapytania po utworzeniu indeksu jest taki sam jak przed jego utworzeniem, a więc indeks nie został użyty.

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

Wyniki:

Wymuszenie indeksu

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



Porównanie

Koszt zapytania przed wymuszeniem indeksu:

Table Scan

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

Koszt zapytania po wymuszeniu indeksu:

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

select productassemblyid, componentid, startdate from billofmate

Nested Loops
(Inner Join)

Index Seek (NonClustered)
[billofmaterials].[billofmaterials_...

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
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'

Po wymuszeniu indeksu, widzimy, że został on wykorzystany, jednak koszt zapytania wzrósł z 0.02 do 0.07.

Bez wymuszania indeksu planer decyduje się na jego nie wykorzystanie, prawdopodobnie dlatego, że zapytanie korzystające z tego indeksu jest mniej wydajne.

Punktacja:

zadanie	pkt
1	2

2	2
3	2
4	2
5	2
razem	10