

# Indeksy, optymalizator

## Lab 2

---

**Imię i nazwisko:** Damian Torbus, Adam Woźny

---

Celem ćwiczenia jest zapoznanie się z planami wykonania zapytań (execution plans), oraz z budową i możliwością wykorzystaniem indeksów

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
  - ewentualnie inne narzędzie umożliwiające komunikację z MS SQL Server i analizę planów zapytań
- przykładowa baza danych AdventureWorks2017.

Oprogramowanie dostępne jest na przygotowanej maszynie wirtualnej

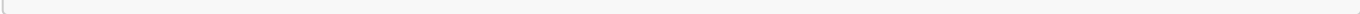
## Przygotowanie

Uruchom Microsoft SQL Managment Studio.

Stwórz swoją bazę danych o nazwie lab2.

```
create database lab2
go

use lab2
go
```



# Zadanie 1

---

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:

ResultsMessages

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

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

SELECT  
0.128s

Table Scan  
[person]  
0.128s  
1 of  
2 (50%)

Estimated query progress:100%

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

SELECT

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

Estimated query progress:100%

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

SELECT

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

4 / 34

Results Messages Live Query Statistics Execution plan

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

Table Scan  
 [person]  
 Cost: 100 %  
 0.001s  
 1 of  
 2 (50%)

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

Table Scan  
 [person]  
 Cost: 100 %  
 0.002s

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

Table Scan  
 [person]  
 Cost: 100 %  
 0.002s

### Komentarz:

Dla wszystkich trzech zapytań wynik jest identyczny - zwracany jest jeden rekord osoby:

- firstname = 'Osarumwense'
- lastname = 'Agbonile'

Analiza pokazuje, że mimo identycznego planu wykonania (Table Scan) dla wszystkich trzech zapytań, różnią się one selektywnością. Zapytanie 2 (WHERE lastname = 'Agbonile' AND firstname = 'Osarumwense') jest najbardziej precyzyjne, bo trafia dokładnie w jeden wiersz. Zapytanie 3 (WHERE firstname = 'Osarumwense') przeszukuje aż 14 wierszy i ma najgorszą selektywność. Brak indeksów powoduje, że SQL Server musi skanować całą tabelę niezależnie od dokładności zapytania.

Przygotuj indeks obejmujący te zapytania:

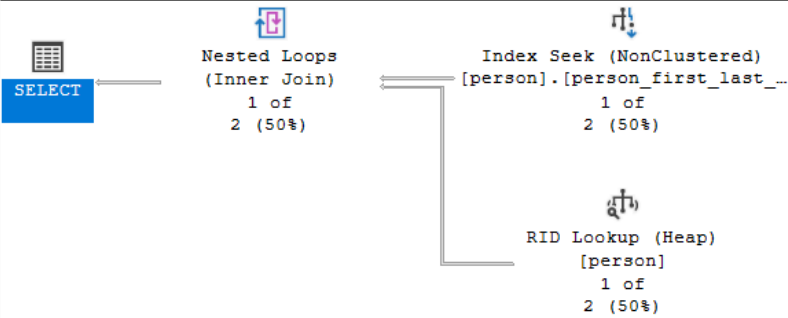
```
create index person_first_last_name_idx
on person(lastname, firstname)
```

Sprawdź plan zapytania. Co się zmieniło?

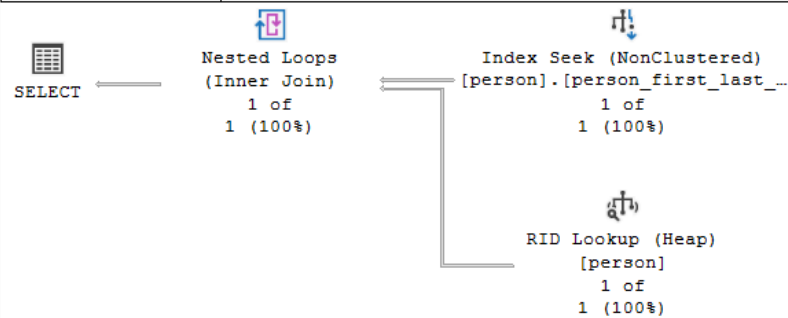
Wyniki:

Results Messages Live Query Statistics Execution plan															
1	4388	IN	0	NULL	Osarumwense	Uwaifiokun	Agbonile	NULL	0	3309A53F-3020-495A-A423-C1DBCCCAC66C	2013-11-30 00:00:00.000				
Rows	Executes	StmtText				StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	DefinedVa			
1	1	SELECT * FROM [person] WHERE [lastname]=@1				1	1	0	NULL	NULL	NULL	NULL			
2	1	--Nested Loops (Inner Join, OUTER REFERENCES: ([lab2].[dbo].[person].[person_first_last_na...				1	2	1	Nested Loops	Inner Join	OUTER REFERENCES: ([Bmk1000])	NULL			
3	1	--Index Seek (OBJECT: ([lab2].[dbo].[person].[person_first_last_na...				1	3	2	Index Seek	Index Seek	OBJECT: ([lab2].[dbo].[person].[person_first_last_na...	[Bmk1000]			
4	1	--RID Lookup (OBJECT: ([lab2].[dbo].[person]), SEEK: ([Bmk1000])=...				1	5	2	RID Lookup	RID Lookup	OBJECT: ([lab2].[dbo].[person]), SEEK: ([Bmk1000])=...	[lab2].[dbc			
1	businessentityid	persontype	namestyle	title	firstname	middlename	lastname	suffix	emailpromotion	rowguid	modifieddate				
1	4388	IN	0	NULL	Osarumwense	Uwaifiokun	Agbonile	NULL	0	3309A53F-3020-495A-A423-C1DBCCCAC66C	2013-11-30 00:00:00.000				
Rows	Executes	StmtText				StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	Defined			
1	1	SELECT * FROM [person] WHERE [lastname]=@1 AND ...				3	1	0	NULL	NULL	NULL	NULL			
2	1	--Nested Loops (Inner Join, OUTER REFERENCES: ([B...				3	2	1	Nested Loops	Inner Join	OUTER REFERENCES: ([Bmk1000])	NULL			
3	1	--Index Seek (OBJECT: ([lab2].[dbo].[person].[person...				3	3	2	Index Seek	Index Seek	OBJECT: ([lab2].[dbo].[person].[person_first_last_na...	[Bmk10			
4	1	--RID Lookup (OBJECT: ([lab2].[dbo].[person]), SEEK:...				3	5	2	RID Lookup	RID Lookup	OBJECT: ([lab2].[dbo].[person]), SEEK: ([Bmk1000])=...	[lab2].[d			
1	businessentityid	persontype	namestyle	title	firstname	middlename	lastname	suffix	emailpromotion	rowguid	modifieddate				
1	4388	IN	0	NULL	Osarumwense	Uwaifiokun	Agbonile	NULL	0	3309A53F-3020-495A-A423-C1DBCCCAC66C	2013-11-30 00:00:00.000				
Rows	Executes	StmtText				StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	DefinedValues	EstimateRows	EstimateIO	EstimateCPU
1	1	SELECT * FROM [person] WHERE [firstname]=@1				5	1	0	NULL	NULL	NULL	NULL	13,57143	NULL	NULL
2	1	--Nested Loops (Inner Join, OUTER REFEREN...				5	2	1	Nested L...	Inner Join	OUTER ...	NULL	13,57143	0	5,672857E...
3	1	--Index Scan (OBJECT: ([lab2].[dbo].[person].[I...				5	3	2	Index Scan	Index Sc...	OBJEC...	[Bmk1000], [I...	13,57143	0,081643...	0,0221262
4	1	--RID Lookup (OBJECT: ([lab2].[dbo].[person])...				5	5	2	RID Look...	RID Loo...	OBJEC...	[lab2].[dbo].[...	1	0,003125	0,0001581

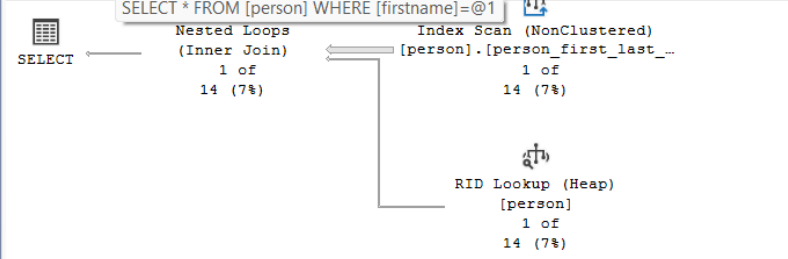
Results	Messages	Live Query Statistics	Execution plan
Estimated query progress:100%	Query 1: Query cost (relative to the batch): 5% select * from [person] where lastname = 'Agbonile'		



Estimated query progress:100%	Query 2: Query cost (relative to the batch): 4% SELECT * FROM [person] WHERE [lastname]=@1 AND [firstname]=@2
-------------------------------	--

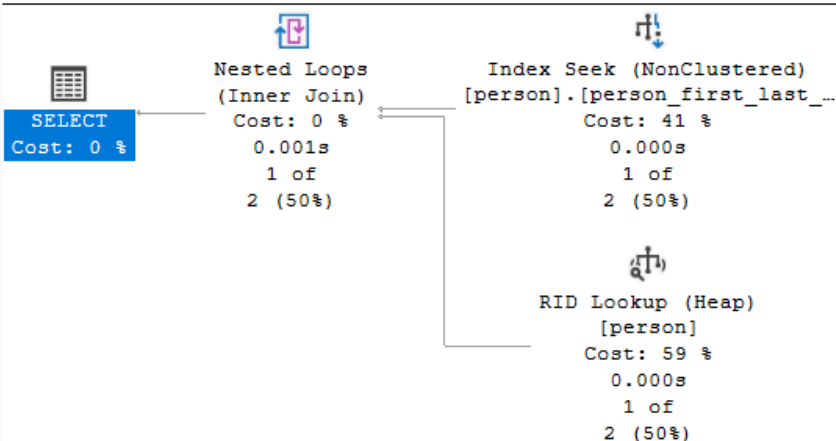


Estimated query progress:100%	Query 3: Query cost (relative to the batch): 91% SELECT * FROM [person] WHERE [firstname]=@1 Missing Index (Impact 97.4859): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [lab2].[dbo].[person] ([firstname])
-------------------------------	---



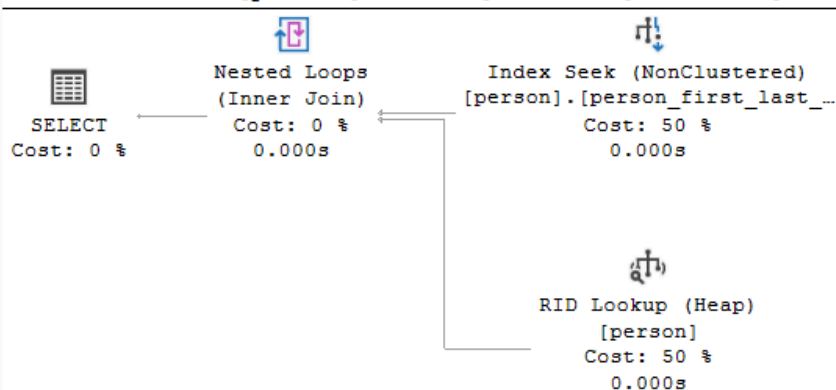
Query 1: Query cost (relative to the batch): 5%

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



Query 2: Query cost (relative to the batch): 4%

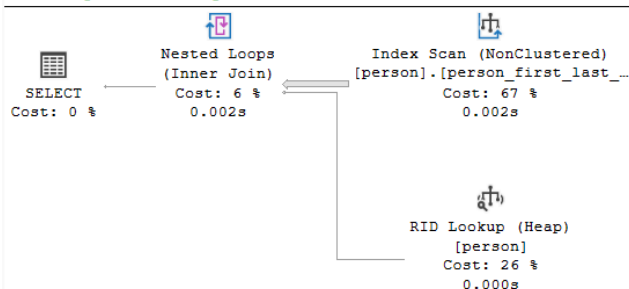
SELECT \* FROM [person] WHERE [lastname]=@1 AND [firstname]=@2



Query 3: Query cost (relative to the batch): 91%

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

Missing Index (Impact 97.4859): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[perso...



## Komentarz:

Po utworzeniu indeksu na kolumnach (lastname, firstname) zapytania filtrujące po nazwisku oraz po nazwisku i imieniu zostały znacząco zoptymalizowane – zaczęły korzystać z operacji Index Seek, co obniżyło ich koszt do 4–5%. Zapytanie filtrujące tylko po imieniu nadal korzysta z Index Scan, ponieważ nie może użyć indeksu zaczynającego się od lastname. W efekcie jego koszt wynosi aż 91%, a SQL Server sam zasugerował utworzenie dodatkowego indeksu na firstname. Aby zoptymalizować wszystkie trzy zapytania, warto więc dodać również osobny indeks na kolumnie firstname.

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:

100 %

Results Messages Live Query Statistics Execution plan

businessentityid	persontype	namestyle	title	firstname	middleinitial	lastname	suffix	emailpromotion	rowguid	modifieddate	
1	6772	IN	0	NULL	Caroline	NULL	Price	NULL	0	C7DC57C5-DA29-4FDC-BFE3-8F9C8A702317	2014-05-04 00:00:00.000
2	6807	IN	0	NULL	Cassidy	A	Price	NULL	0	39CD7220-49D6-48EC-AA5C-2B84EBFD83DC	2013-10-19 00:00:00.000
3	3407	IN	0	NULL	Haley	A	Price	NULL	1	E65C7A6F-81FA-4BB7-935A-4EEA9297D104	2014-02-06 00:00:00.000
4	3578	IN	0	NULL	Taylor	NULL	Price	NULL	0	FF74286D-8B3E-499F-AB5C-357766D1384F	2013-06-02 00:00:00.000
5	3653	IN	0	NULL	Anna	NULL	Price	NULL	0	4745AA4F-E44B-47E3-807B-CA0C8BE5CB56	2011-07-13 00:00:00.000
6	3950	IN	0	NULL	Abigail	M	Price	NULL	1	5C7543CF-FC69-45E7-830D-A6E6D8277699	2013-05-31 00:00:00.000
7	1641	VC	0	Mr.	Jeff	NULL	Price	NULL	2	DA26B021-97BB-4057-8FA8-98EBBAC63387	2012-02-02 00:00:00.000
8	2146	GC	0	NULL	Jeff	NULL	Price	NULL	1	C872FB3E-7B78-4E23-BF79-B0CCAC9CB2A6	2009-01-20 00:00:00.000

Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	DefinedValues
1	84	SELECT * FROM [person] WHERE [lastname]=@1	1	1	0	NULL	NULL	NULL	NULL
2	84	[-Table Scan(OBJECT:([lab2].[dbo].[person]), W...	1	2	1	Table Scan	Table Scan	OBJECT:([lab2].[dbo].[person]), WHERE:([lab2].[d...	[lab2].[dbo].[person].[bu

businessentityid	persontype	namestyle	title	firstname	middleinitial	lastname	suffix	emailpromotion	rowguid	modifieddate	
1	7642	IN	0	NULL	Angela	D	Price	NULL	0	FBB7CF01-97B4-4315-B080-0DE770316BB9	2013-12-14 00:00:00.000

Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	DefinedValues	EstimateRows	EstimateIO	EstimateCPU
1	1	SELECT * FROM [person] WHERE [lastname]=@1 AND ...	3	1	0	NULL	NULL	NULL	NULL	1.023365	NULL	NULL
2	1	[-Nested Loops(Inner Join, OUTER REFERENCES:([B...	3	2	1	Nested L...	Inner Join	OUTER ...	NULL	1.023365	0	4.277667
3	1	[-Index Seek(OBJECT:([lab2].[dbo].[person].[person...	3	3	2	Index Seek	Index Se...	OBJEC...	[Bmk1000], [...	1.023365	0.003125	0.000158
4	1	[-RID Lookup(OBJECT:([lab2].[dbo].[person]), SEEK:...	3	5	2	RID Look...	RID Loo...	OBJEC...	[lab2].[dbo].[...	1	0.003125	0.000158


businessentityid	persontype	namestyle	title	firstname	middleinitial	lastname	suffix	emailpromotion	rowguid	modifieddate	
40	7681	IN	0	NULL	Angela	NULL	Griffin	NULL	0	42CC6211-576C-49D4-9F15-A368CA94DD3B	2014-06-01 00:00:00.000
41	7683	IN	0	NULL	Angela	K	Diaz	NULL	0	59CEBE73-FBED-476D-867E-88093682C30B	2012-11-18 00:00:00.000
42	7684	IN	0	NULL	Angela	NULL	Hayes	NULL	0	1316BC8D-0FDD-4C9D-923E-26EA76C168AA	2013-09-24 00:00:00.000
43	7948	IN	0	NULL	Angela	NULL	Brooks	NULL	0	13241316-57B3-4C9D-8E42-363367F49904	2014-06-04 00:00:00.000
44	8006	IN	0	NULL	Angela	J	Kelly	NULL	0	E1C9B7A6-8AD1-4983-89F9-1EFC467815C4	2014-06-15 00:00:00.000
45	8065	IN	0	NULL	Angela	NULL	Sanders	NULL	0	C9EF519B-2C9D-44D6-9E9A-69D5E56B5DC8	2013-09-19 00:00:00.000
46	8274	IN	0	NULL	Angela	T	Gray	NULL	0	D7AEA030-20DC-4807-BA8F-9D4468F8F314	2014-01-16 00:00:00.000
47	8330	IN	0	NULL	Angela	NULL	Ramirez	NULL	2	4EE45333-F8E4-47A0-91DD-3A03B0FBDF9A	2013-10-27 00:00:00.000


Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	DefinedValues	EstimateRows	EstimateIO	EstimateCPU
1	50	SELECT * FROM [person] WHERE [firstname]=@1	5	1	0	NULL	NULL	NULL	NULL	50	NULL	NULL
2	50	[-Table Scan(OBJECT:([lab2].[dbo].[person]), W...	5	2	1	Table Scan	Table Sc...	OBJEC...	[lab2].[dbo].[...	50	0.1557176	0.0221262

Results Messages Live Query Statistics Execution plan

Estimated query progress:0%


Query 1: Query cost (relative to the batch): 33%  
select \* from [person] where lastname = 'Price'


Table Scan  
[person]  
- of  
84 (0%)

SELECT

Estimated query progress:100%

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

Table Scan  
[person]  
84 of  
84 (100%)

SELECT



Estimated query progress:100% Query 3: Query cost (relative to the batch): 1%  
 select \* from [person] where lastname = 'Price' and firstname = 'Angela'

Query 4: Query cost (relative to the batch): 33%  
 SELECT \* FROM [person] WHERE [firstname]=@1  
 Missing Index (Impact 97.9375): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[person] ([firstname])

Query 1: Query cost (relative to the batch): 49%  
 SELECT \* FROM [person] WHERE [lastname]=@1

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

Query 3: Query cost (relative to the batch): 49%  
 SELECT \* FROM [person] WHERE [firstname]=@1  
 Missing Index (Impact 97.9375): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[person] ([firstname])

## Komentarz:

W przypadku zapytań dla Angela i Price SQL Server częściej używał pełnego skanowania tabeli (Table Scan), ponieważ wartości Price i Angela występują bardzo często, co czyni warunki mało selektywnymi. Dla Osarumwense i Agbonile, gdzie dane były unikalne, wykorzystywany był Index Seek, co było bardziej wydajne. Różnica wynika z selektywności - im rzadsze dane, tym większa szansa na użycie indeksu.

## Zadanie 2

Skopiuj tabelę Product do swojej bazy danych:

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

Stwórz indeks z warunkiem przedziałowym:

```
create nonclustered index product_range_idx
on product (productsubcategoryid, listprice) include (name)
where productsubcategoryid >= 27 and productsubcategoryid <= 36
```

Sprawdź, czy indeks jest użyty w zapytaniu:

```
select name, productsubcategoryid, listprice
from product
where productsubcategoryid >= 27 and productsubcategoryid <= 36
```

Sprawdź, czy indeks jest użyty w zapytaniu, który jest dopełnieniem zbioru:

```
select name, productsubcategoryid, listprice
from product
where productsubcategoryid < 27 or productsubcategoryid > 36
```

Skomentuj oba zapytania. Czy indeks został użyty w którymś zapytaniu, dlaczego? Jak działają indeksy z warunkiem?

Postanowiliśmy również dołożyć zapytanie 3, którego celem jest sprawdzenie przypadku, kiedy zbiór danych, które pokrywa zapytanie będzie częściowo rozłączny z danymi na których został założony indeks.

```
select name, productsubcategoryid, listprice
from product
where productsubcategoryid >= 26 and productsubcategoryid <= 36;
```

---

Wyniki:

Zapytanie 1

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the query execution progress at 100% and the query text: `SELECT [name],[productsubcategoryid],[listprice] FROM [product] WHERE [productsubcategoryid]>=1 AND [productsubcategoryid]<=2`. The bottom pane shows the execution plan, which is an `Index Scan (NonClustered)` on `[product].[product_range_idx]`. The status bar indicates the query executed successfully on localhost (15.0 RTM) with 19 rows returned.

Query 1: Query cost (relative to the batch): 100%  
progress:100%  
`SELECT [name],[productsubcategoryid],[listprice] FROM [product] WHERE [productsubcategoryid]>=1 AND [productsubcategoryid]<=2`

Index Scan (NonClustered)  
[product].[product\_range\_idx]  
17 of 17 (100%)

Query executed successfully. localhost (15.0 RTM) SA (62) LAB5 00:00:01 19 rows

Results Messages Live Query Statistics Execution plan

(17 rows affected)  
Table 'product'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page  
(2 rows affected)  
(1 row affected)

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

100 %

Query executed successfully. localhost (15.0 RTM) SA (62) LAB5 00:00:01 19 rows

Indeks został użyty w tym zapytaniu, dzieje się tak, dlatego, że zakres danych, które obejmuje zapytanie zawiera się w przedziale (dla danej kolumny) na którym został założony indeks. Z racji użycia indeksu zostały wykonane tylko 2 operacje logicznego odczytania, koszt wyniósł 0,0033007.

## Zapytanie 2

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the query execution progress at 100% and the query text: `select name, productsubcategoryid, listprice from product where productsubcategoryid < 27 or productsubcategoryid > 36`. The bottom pane shows the execution plan, which is a `Table Scan` on `[product]`. The status bar indicates the query executed successfully on localhost (15.0 RTM) with 280 rows returned.

Query 1: Query cost (relative to the batch): 100%  
progress:100%  
`select name, productsubcategoryid, listprice from product where productsubcategoryid < 27 or productsubcategoryid > 36`

Table Scan  
[product]  
278 of 278 (100%)

Query executed successfully. localhost (15.0 RTM) SA (62) LAB5 00:00:01 280 rows

Results Messages Live Query Statistics Execution plan

Table 'product'. Scan count 1, logical reads 13, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page  
(2 rows affected)  
(1 row affected)

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 1 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

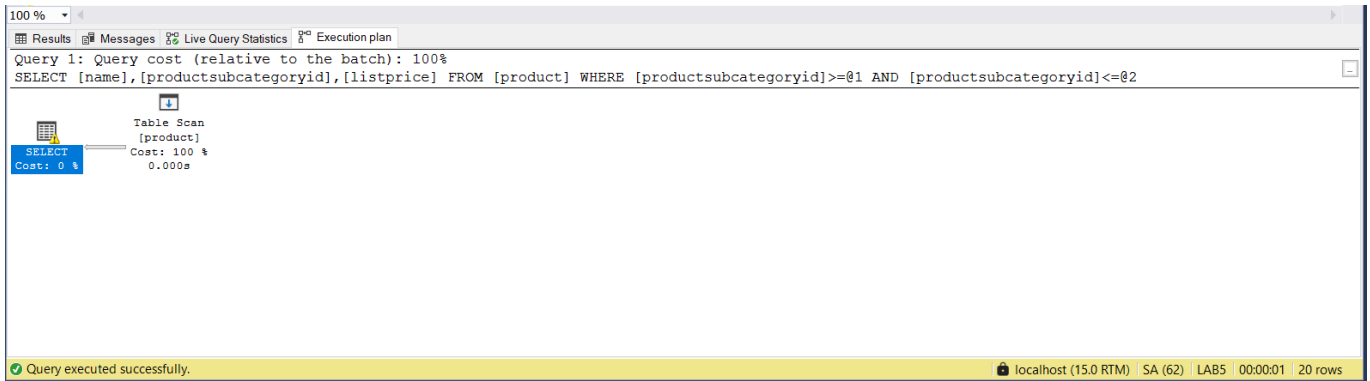
Completion time: 2025-05-23T11:56:14.1317457+02:00

100 %

Query executed successfully. localhost (15.0 RTM) SA (62) LAB5 00:00:01 280 rows

Indeks nie został użyty w tym zapytaniu, dzieje się tak, dlatego, że zakres danych, które obejmuje zapytanie jest rozłączny z przedziałem, na którym został założony indeks. Z racji braku użycia indeksu zostały wykonane 13 operacje logicznego odczytania, koszt wyniósł 0,01272529.

## Zapytanie 3



Pomimo częściowej zgodności danymi które obejmuje indeks i zapytanie, została wykonana operacja pełnego skanowania tabeli, koszty porównywalne jak w zapytaniu 2.

Oba zapytania wybierają kolumny `name`, `productsubcategoryId`, `listprice` z tabeli `product`, oba dodatkowo posiadają w swojej strukturze klauzulę `WHERE`. Pierwsze zapytanie skorzystało z już istniejącego indeksu `product_range_idx`, założonego na kolumnach `productsubcategoryId`, `listprice` z warunkiem `productsubcategoryId >= 27 and productsubcategoryId <= 36`, ponieważ warunki zapytania obejmowały dane, które spełniają warunek indeksu. W przypadku drugiego zapytania sytuacja była odwrotna, dane, które były objęte zapytaniem nie zawierały się w danych, na których był założony indeks. Dzięki temu, że indeks był założony lub uwzględniał każdą z kolumn nie musiały być wykonywane żadne inne operacje poza `index seek`, co widać w kosztach zapytań (0,0033007 vs 0,01272529). Indeksy z warunkiem to indeksy nieklastrowane, które w swojej strukturze obejmują tylko podzbiór rekordów spełniający wspomniany warunek, przez co pozwalają na efektywne wyszukiwania danych tylko z tego podzbioru. Wykorzystuje się je ponieważ są tanie w budowie, utrzymaniu, zajmują mniej miejsca oraz pozwalają na szybsze wykonanie operacji `index seek` z racji na swój mniejszy rozmiar. Trzeba być jednak ostrożnym podczas zakładania, ponieważ jakikolwiek brak pokrycia danych z zapytania poprzez indeks wymusza pełne skanowanie tabeli.

## Zadanie 3

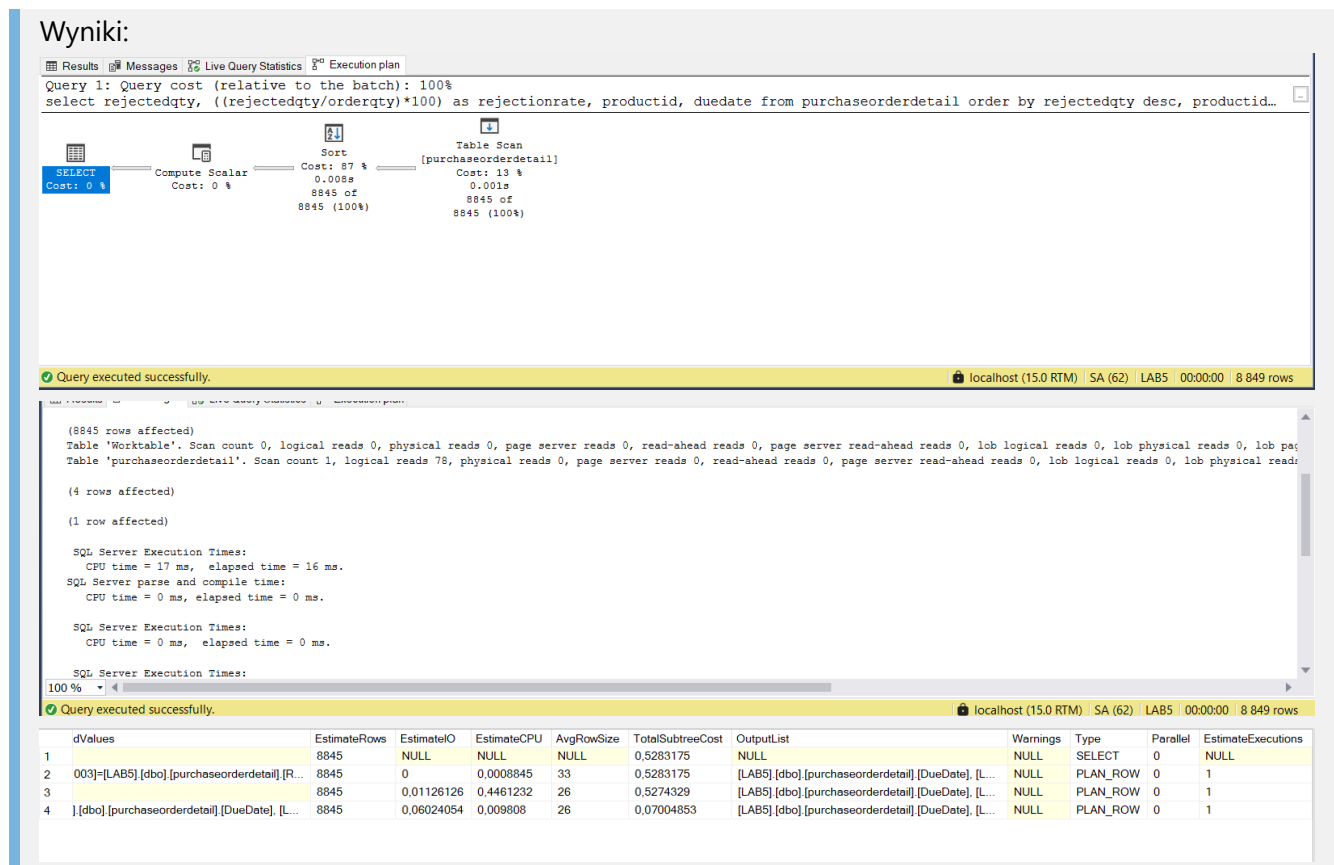
Skopiuj tabelę `PurchaseOrderDetail` do swojej bazy danych:

```
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?



Zapytanie wybiera z tabeli `purchaseorderdetail` wartości kolumn `rejectedqty`, `orderqty`, `productid`, `duedate` (lub też różne wyrażenia zbudowane na ich podstawie), posortowane według `rejectedqty` malejąco i `productid` rosnąco. Analizując plan zapytania oraz jego koszt można zauważyć, że najbardziej kosztowną operacją podczas całego zapytania było `sort` (87% kosztów całego). Zostało przeprowadzone pełne skanowanie tabeli, które wymagało jedynie 13% kosztów całego zapytania. Zostało przeprowadzone 78 operacji logicznego odczytu, sumaryczny koszt zapytania wynosił 0,5274329 i trwało 17ms.

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

Jako, że występuje tu sortowanie na dwóch kolumnach, niemonotoniczne wobec siebie nawzajem (przy jednej kolumnie jest DESC, a przy drugiej ASC), proponujemy, aby stworzyć indeks nieklastrowany na tych właśnie kolumnach, uwzględniający inne kolumny zwracane przez zapytanie oraz uwzględniający też monotoniczności kolumn, według których sortujemy.

## Wyniki:

```
CREATE NONCLUSTERED INDEX purchaseorderdetail_rejectedqty_productid_idx
ON purchaseorderdetail (rejectedqty DESC, productid ASC)
INCLUDE (orderqty, duedate);
```

Ponownie wykonaj analizę zapytania:

Wyniki:

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...

SELECT  
Cost: 0 %

Compute Scalar  
Cost: 2 %

Index Scan (NonClustered)  
[purchaseorderdetail].[IX\_pu...]  
Cost: 98 %  
0.001s

Query executed successfully. localhost (15.0 RTM) SA (62) LAB5 00:00:01 8 848 rows

(8845 rows affected)  
 Table 'purchaseorderdetail'. Scan count 1, logical reads 39, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0

(3 rows affected)

(1 row affected)

SQL Server Execution Times:  
 CPU time = 3 ms, elapsed time = 7 ms.  
 SQL Server parse and compile time:  
 CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:  
 CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:  
 CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2025-05-23T12:38:55.0716598+02:00

dValues	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExecutions
1	8845	NULL	NULL	NULL	0,04056267	NULL	NULL	SELECT	0	NULL
2	003=[LAB5] [dbo] [purchaseorderdetail] [R...	8845	0	0,0008845	33	0,04056267	NULL	PLAN_ROW	0	1
3	[ [dbo] [purchaseorderdetail] [DueDate], [L...	8845	0,02979167	0,0098885	26	0,03967817	NULL	PLAN_ROW	0	1

Analizując zapytanie wykonane po utworzeniu indeksu można zauważyć, że nie ma operacji **sort** oraz operacji pełnego skanowania. Stało się tak, dlatego, że indeks pozwolił na przechowywanie struktury tabeli wraz z monotonicznością, którą chcemy uzyskać w zapytaniu. Pełne skanowanie też nie musiało być wykonane, chociaż w tym wypadku z racji na to, że rekordy nie są filtrowane nie obeszliśmy zbytniego spadku kosztu na tej operacji. W przeciwieństwie do kosztu całego zapytania, który teraz wynosi 0,04056267, poprzez usunięcie kosztu sortowania oraz zredukowania (nieznacznego, ale jednak) kosztu skanowania tabeli (z pełnego skanowania => **index seek**)

## Zadanie 4 – indeksy column store

Celem zadania jest poznanie indeksów typu column store

Utwórz tabelę testową:

```
create table dbo.saleshistory(
  salesorderid int not null,
  salesorderdetailid int not null,
  carriertrackingnumber nvarchar(25) null,
  orderqty smallint not null,
  productid int not null,
  specialofferid int not null,
  unitprice money not null,
```

```
unitpricediscount money not null,  
linetotal numeric(38, 6) not null,  
rowguid uniqueidentifier not null,  
modifieddate datetime not null  
)
```

Założ indeks:

```
create clustered index saleshistory_idx  
on saleshistory(salesorderdetailid)
```

Wypełnij tablicę danymi:

(UWAGA GO 100 oznacza 100 krotne wykonanie polecenia. Jeżeli podejrzewasz, że Twój serwer może to zbyt przeciążyć, zacznij od GO 10, GO 20, GO 50 (w sumie już będzie 80))

```
insert into saleshistory  
select sh.*  
from adventureworks2017.sales.salesorderdetail sh  
go 100
```

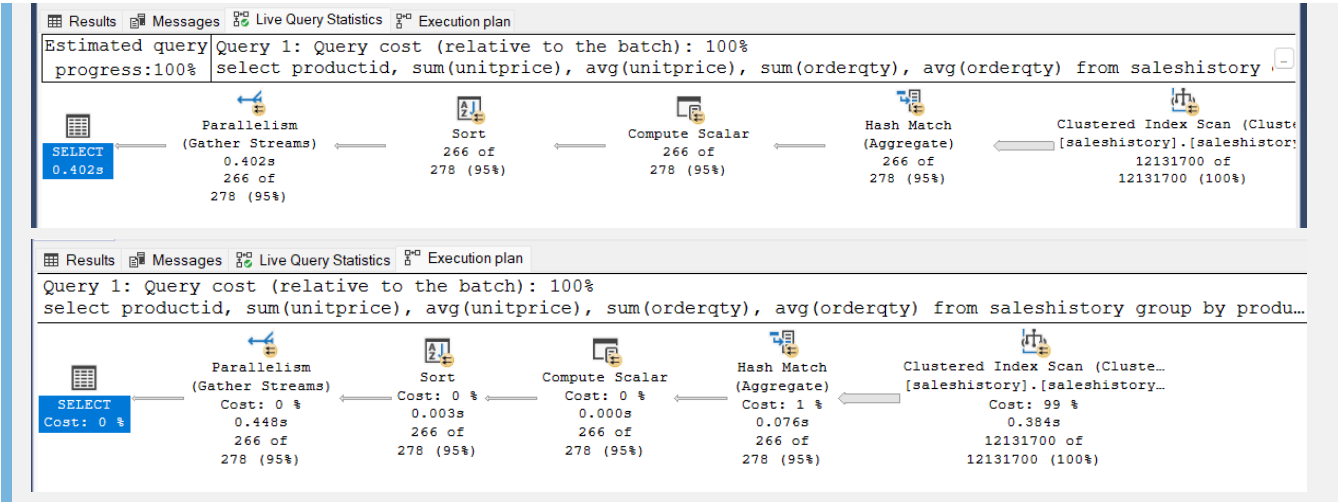
Sprawdź jak zachowa się zapytanie, które używa obecny indeks:

```
select productid, sum(unitprice), avg(unitprice), sum(orderqty), avg(orderqty)  
from saleshistory  
group by productid  
order by productid
```

Wyniki:

Results						Messages	Live Query Statistics	Execution plan	
	productid	(No column name)	(No column name)	(No column name)	(No column name)				
1	707	9522317.98	30.8865	626600	2				
2	708	9156173.10	30.4495	653200	2				
3	709	106333.50	5.656	110700	5				
4	710	25080.00	5.70	9000	2				
5	711	9382785.66	30.365	674300	2				
6	712	2593395.71	7.6682	831100	2				
7	713	2144571.00	49.99	42900	1				
8	714	4479942.95	36.7811	363600	2				
9	715	5706317.12	34.901	659200	4				
10	716	3998954.56	37.165	298000	2				
11	717	17746102.26	814.0413	48500	2				
12	718	17810841.95	813.2804	48600	2				
13	719	3615188.22	821.6336	10900	2				
14	722	7383582.92	188.3567	94000	2				
15	723	1003766.78	193.032	12900	2				
16	725	7275245.88	194.5252	99100	2				
17	726	5634271.96	195.6344	67100	2				
18	727	960157.32	200.0327	10000	2				
19	729	7303388.42	194.757	99600	2				
20	730	5642794.42	195.9303	69700	2				

Rows	Executes	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp	Argument	De
1	266	select productid, sum(unitprice), avg(unitprice), s...	1	1	0	NULL	NULL	NULL	NI
2	266	-Parallelism(Gather Streams, ORDER BY:([lab...	1	2	1	Parallelism	Gather Streams	ORDER BY:([lab2].[dbo].[saleshistory].[productid] ASC)	NI
3	266	-Sort(ORDER BY:([lab2].[dbo].[saleshistory].[...	1	3	2	Sort	Sort	ORDER BY:([lab2].[dbo].[saleshistory].[productid] ASC)	NI
4	266	-Compute Scalar(DEFINE:([Expr1004]=C...	1	4	3	Compute Scalar	Compute Scalar	DEFINE:([Expr1004]=CASE WHEN [Expr1017]=(0) THEN ...	[E
5	266	-Hash Match(Aggregate, HASH:([lab2]...	1	5	4	Hash Match	Aggregate	HASH:([lab2].[dbo].[saleshistory].[productid])	[E
6	121...	-Clustered Index Scan(OBJECT:([la...	1	6	5	Clustered Inde...	Clustered Inde...	OBJECT:([lab2].[dbo].[saleshistory].[saleshistory_idx])	[la



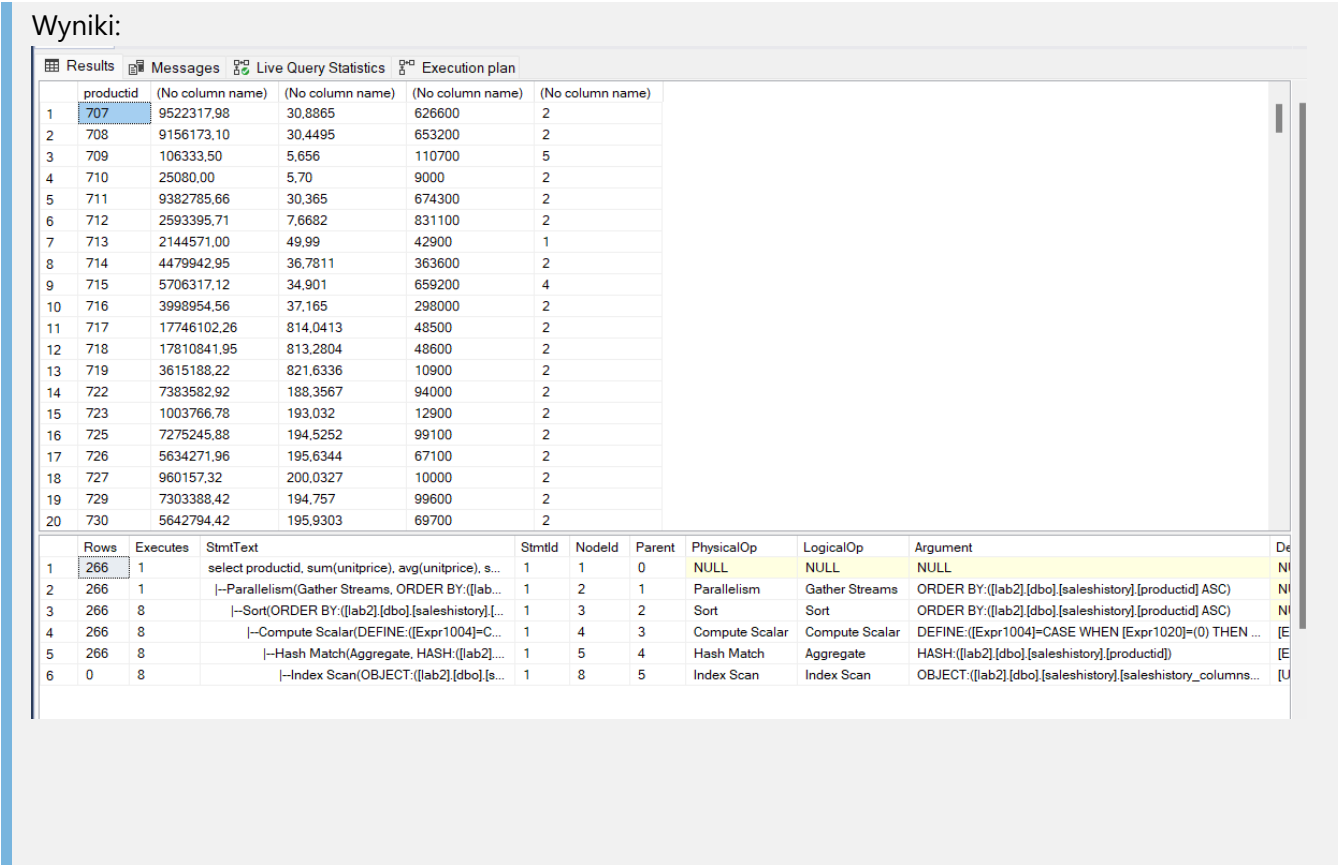
Komentarz:

Wyniki pokazują, że zapytanie musiało przetworzyć ponad 12 milionów wierszy za pomocą Clustered Index Scan, co wiązało się z dużym kosztem. SQL Server użył równoległości i operacji Hash Match, co potwierdza, że klasyczny indeks klastrowy nie jest wydajny przy analizach z agregacjami.

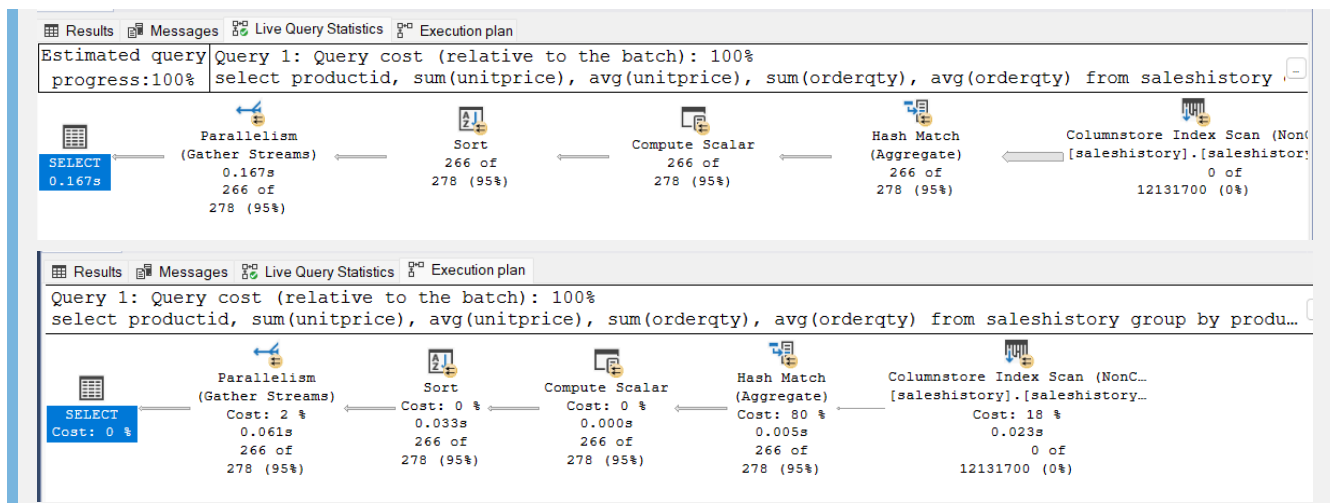
Założ indeks typu column store:

```
create nonclustered columnstore index saleshistory_columnstore
on saleshistory(unitprice, orderqty, productid)
```

Sprawdź różnicę pomiędzy przetwarzaniem w zależności od indeksów. Porównaj plany i opisz różnicę. Co to są indeksy columns store? Jak działają? (poszukaj materiałów w internecie/literaturze)







## Komentarz:

Zastosowanie indeksu typu columnstore znacząco przyspieszyło zapytanie agregujące – czas wykonania spadł z około 0,40 s do 0,16 s. SQL Server zamiast pełnego skanowania wierszy (Clustered Index Scan) użył odczytu kolumnowego (Columnstore Index Scan) i przetwarzania wsadowego, co pozwoliło na szybsze i bardziej efektywne wykonanie operacji. Columnstore indeksy przechowują dane w kolumnach, co ułatwia kompresję i redukuje ilość danych do przetworzenia. Tego typu indeksy są idealne do analiz dużych zbiorów danych z użyciem agregacji i GROUP BY.

## Zadanie 5 – własne eksperymenty

Należy zaprojektować tabelę w bazie danych, lub wybrać dowolny schemat danych (poza używanymi na zajęciach), a następnie wypełnić ją danymi w taki sposób, aby zrealizować poszczególne punkty w analizie indeksów. Warto wygenerować sobie tabele o większym rozmiarze.

Do analizy, proszę uwzględnić następujące rodzaje indeksów:

- Klastrowane (np. dla atrybutu nie będącego kluczem głównym)
- Nieklastrowane
- Indeksy wykorzystujące kilka atrybutów, indeksy include
- Filtered Index (Indeks warunkowy)
- Kolumnowe

## Analiza

Proszę przygotować zestaw zapytań do danych, które:

- wykorzystują poszczególne indeksy
- które przy wymuszeniu indeksu działają gorzej, niż bez niego (lub pomimo założonego indeksu, tabela jest w pełni skanowana) Odpowiedź powinna zawierać:
- Schemat tabeli
- Opis danych (ich rozmiar, zawartość, statystyki)
- Opis indeksu
- Przygotowane zapytania, wraz z wynikami z planów (zrzuty ekranów)
- Komentarze do zapytań, ich wyników

- Sprawdzenie, co proponuje Database Engine Tuning Advisor (porównanie czy udało się Państwu znaleźć odpowiednie indeksy do zapytania)

## Eksperymenty

Eksperymenty będą przeprowadzone na tabeli AccountingDocuments wzorowanej na strukturach danych z systemów SAP/EPR. Zdecydowaliśmy się na taki przykład z uwagi na powszechność obsługi tego typu danych biznesowych w hurtowniach baz danych, co może prowadzić nas do bliższych prawdziwemu zastosowaniu.

Tabela zawiera dane n.t. faktur pewnej firmy. Warto zwrócić uwagę na przyrostowy charakter danych oraz na zastosowanie *soft delete'u* - dane są oznaczane jako usunięte w miejsce fizycznych operacji, nie zawsze można sobie pozwolić na utratę danych (n.p. prowadzimy archiwum), ale posiadanie wszystkich danych może być uciążliwe.

DDL tabeli:

```
CREATE TABLE AccountingDocuments (  
    DocumentID INT IDENTITY,  
    CompanyCode VARCHAR(10),  
    FiscalYear INT,  
    DocumentNumber VARCHAR(20),  
    PostingDate DATE,  
    DocumentDate DATE,  
    Amount DECIMAL(18,2),  
    Currency VARCHAR(3),  
    DocumentType VARCHAR(5),  
    IsDeleted BIT DEFAULT 0,  
    CreatedAt DATETIME DEFAULT GETDATE()  
);
```

Generowanie danych:

```
INSERT INTO AccountingDocuments (  
    CompanyCode, FiscalYear, DocumentNumber,  
    PostingDate, DocumentDate, Amount,  
    Currency, DocumentType, IsDeleted  
)  
SELECT  
    CHOOSE(ABS(CHECKSUM(NEWID())) % 3 + 1, 'PL01', 'DE02', 'US03'),  
    2020 + ABS(CHECKSUM(NEWID())) % 5,  
    FORMAT(ABS(CHECKSUM(NEWID())) % 999999, '000000'),  
    DATEADD(DAY, -ABS(CHECKSUM(NEWID())) % 1000, GETDATE()),  
    DATEADD(DAY, -ABS(CHECKSUM(NEWID())) % 1000, GETDATE()),  
    ROUND(RAND(CHECKSUM(NEWID())) * 10000, 2),  
    CHOOSE(ABS(CHECKSUM(NEWID())) % 3 + 1, 'PLN', 'EUR', 'USD'),  
    CHOOSE(ABS(CHECKSUM(NEWID())) % 4 + 1, 'KR', 'DR', 'SA', 'AB'),  
    IIF(ABS(CHECKSUM(NEWID())) % 100 < 5, 1, 0)  
FROM sys.all_objects a CROSS JOIN sys.all_objects b  
WHERE a.object_id < 200 AND b.object_id < 100;
```

Generowanie danych odbywa się w sposób losowy, inspirowane danymi faktycznymi. Ok 5% rekordów jest oznaczonych jako usuniętych

```
SELECT COUNT(1) FROM AccountingDocuments;
```

Zostało wygenerowane ok 5,7 miliona rekordów (dokładnie 5712100).

## Indeksy użyte podczas eksperymentów

- Indeks klastrowany założony na innej kolumnie niż klucz główny. Wybraliśmy kolumnę `PostingDate` z uwagi na to, że często filtruje się po dacie

```
CREATE CLUSTERED INDEX IX_AccountingDocuments_PostingDate  
ON AccountingDocuments(PostingDate);
```

- Indeks nieklastrowany założony na więcej niż jednej kolumnie

```
CREATE NONCLUSTERED INDEX IX_AD_Company_Year  
ON AccountingDocuments(CompanyCode, FiscalYear);
```

- Indeks nieklastrowany z użyciem klauzuli `include`

```
CREATE NONCLUSTERED INDEX IX_AD_DocumentNumber_Include  
ON AccountingDocuments(CompanyCode, FiscalYear)  
INCLUDE (Amount, Currency);
```

- Indeks warunkowy - tylko nieusunięte rekordy

```
CREATE NONCLUSTERED INDEX IX_AD_NotDeleted  
ON AccountingDocuments(CompanyCode, PostingDate)  
INCLUDE (Amount, Currency)  
WHERE IsDeleted = 0;
```

- Indeks *columnstore*

```
CREATE CLUSTERED COLUMNSTORE INDEX IX_AD_Columnstore  
ON AccountingDocuments;
```

## Zapytania testujące

Użycie ineksu klastrowego + pułapka

```
SELECT * FROM AccountingDocuments
WHERE PostingDate BETWEEN '2024-01-01' AND '2024-01-31';
```

```
CREATE CLUSTERED INDEX IX_AccountingDocuments_PostingDate
ON AccountingDocuments(PostingDate);
```

W tym teście będziemy analizować bardzo proste zapytanie, które wybiera wartość wszystkich atrybutów z tabeli oraz za pomocą klauzuli WHERE filtruje wynik, tak żeby PostingDate (na którym jest założony indeks klastrowy) był w styczniu 2024

Przed założeniem ineksu

ResultsMessagesLive Query StatisticsExecution plan

Estimated queryQuery 1: Query cost (relative to the batch): 100%  
(@1 varchar(8000),@2 varchar(8000))SELECT \* FROM [AccountingDocuments] WHERE [PostingDate]>=@1 AND [PostingDate]<=@2  
progress:100%Missing Index (Impact 60.2771): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[AccountingDocuments] ([PostingDate])

SELECT

Parallelism  
(Gather Streams)  
176907 of  
176556 (100%)

Table Scan  
[AccountingDocuments]  
176907 of  
176556 (100%)

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:01176 910 rows

ResultsMessagesLive Query StatisticsExecution plan

CPU time = 0 ms, elapsed time = 0 ms.  
  
(176907 rows affected)  
Table 'AccountingDocuments'. Scan count 9, logical reads 49228, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0  
  
(3 rows affected)  
  
(1 row affected)  
  
SQL Server Execution Times:  
CPU time = 961 ms, elapsed time = 407 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

110 %

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:01176 910 rows

ResultsMessagesLive Query StatisticsExecution plan

	DocumentID	CompanyCode	FiscalYear	DocumentNumber	PostingDate	DocumentDate	Amount	Currency	DocumentType	IsDeleted	CreatedAt
1	5267810	NULL	2024	996146	2024-01-01	2024-03-03	2615.81	NULL	KR	0	2025-05-26 15:17:57.933
2	5267591	NULL	2024	823619	2024-01-01	2022-09-13	6815.91	PLN	NULL	0	2025-05-26 15:17:57.933
3	5263875	US03	2022	772565	2024-01-01	2024-12-24	6752.19	EUR	NULL	0	2025-05-26 15:17:57.933
4	5260519	NULL	2023	666115	2024-01-01	2024-01-11	7228.64	USD	SA	0	2025-05-26 15:17:57.933
5	5258985	US03	2023	246243	2024-01-01	2023-09-28	5861.67	USD	KR	0	2025-05-26 15:17:57.933
6	5257572	NULL	2023	599507	2024-01-01	2023-05-21	6708.76	PLN	SA	0	2025-05-26 15:17:57.933
7	5254004	US03	2024	836714	2024-01-01	2024-07-11	3586.95	NULL	NULL	0	2025-05-26 15:17:57.933
8	5254530	NULL	2023	765312	2024-01-01	2024-03-26	4458.47	NULL	NULL	0	2025-05-26 15:17:57.933
9	5255072	DE02	2022	285345	2024-01-01	2023-03-08	956.51	PLN	NULL	1	2025-05-26 15:17:57.933

	Values	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExecutions
1		176556.4	NULL	NULL	NULL	39,79314	NULL	NULL	SELECT	0	NULL
2		176556.4	0	0,4979856	68	39,79314	[LAB5].[dbo].[AccountingDocuments].[DocumentID]...	NULL	PLAN_ROW	1	1
3	dbo].[AccountingDocuments].[DocumentID]...	176556.4	36,46765	1,570847	68	38,03849	[LAB5].[dbo].[AccountingDocuments].[DocumentID]...	NULL	PLAN_ROW	1	1

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:01176 907 rows

Po założeniu indeksu

ResultsMessagesLive Query StatisticsExecution plan

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

SELECT

Cost: 0 %

Clustered Index Seek (Cluste...  
[AccountingDocuments].[IX\_Ac...  
Cost: 100 %  
0.092s  
176907 of  
176921 (99%)

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:01176 909 rows

110 %

ResultsMessagesLive Query StatisticsExecution plan

CPU time = 0 ms, elapsed time = 0 ms.

(176907 rows affected)  
Table 'AccountingDocuments'. Scan count 1, logical reads 1532, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical re  
(2 rows affected)  
(1 row affected)  
SQL Server Execution Times:  
CPU time = 111 ms, elapsed time = 147 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:01176 909 rows

110 %

ResultsMessagesLive Query StatisticsExecution plan

	DocumentID	CompanyCode	FiscalYear	DocumentNumber	PostingDate	DocumentDate	Amount	Currency	DocumentType	IsDeleted	CreatedAt
1	30830	PL01	2023	494508	2024-01-01	2024-12-13	1525.08	PLN	KR	0	2025-05-26 15:17:57.933
2	30381	NULL	2022	958118	2024-01-01	2023-08-24	5485.53	PLN	KR	0	2025-05-26 15:17:57.933
3	30658	DE02	2024	970663	2024-01-01	2024-03-21	5328.82	EUR	KR	0	2025-05-26 15:17:57.933
4	16601	PL01	2020	104982	2024-01-01	2024-01-20	1825.93	PLN	NULL	0	2025-05-26 15:17:57.933
5	17253	NULL	2020	586297	2024-01-01	2024-09-12	195.45	EUR	DR	0	2025-05-26 15:17:57.933
6	16924	NULL	2022	779681	2024-01-01	2023-04-07	5740.38	NULL	KR	0	2025-05-26 15:17:57.933
7	17014	PL01	2022	507466	2024-01-01	2025-03-22	1734.67	PLN	NULL	0	2025-05-26 15:17:57.933
8	17960	DE02	2024	780679	2024-01-01	2023-05-27	6122.04	NULL	NULL	0	2025-05-26 15:17:57.933
9	20321	DE02	2024	443584	2024-01-01	2024-05-03	1708.04	USD	KR	0	2025-05-26 15:17:57.933
10	18624	DE02	2023	443074	2024-01-01	2024-09-22	7843.80	PLN	KR	0	2025-05-26 15:17:57.933
11	23416	DE02	2024	070914	2024-01-01	2022-09-24	7271.87	EUR	AR	0	2025-05-26 15:17:57.933

	DefinedValues	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExec
1	NULL	176921.4	NULL	NULL	NULL	1,326784	NULL	NULL	SELECT	0	NULL
2	[LAB5].[dbo].[AccountingDocuments].[DocumentID]...	176921.4	1,132014	0,1947705	68	1,326784	[LAB5].[dbo].[AccountingDocuments].[DocumentID]...	NULL	PLAN_ROW	0	1

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:01176 907 rows

Pułapka

```
SELECT * FROM AccountingDocuments
WHERE YEAR(PostingDate) = 2024 AND MONTH(PostingDate) = 1;
```

ResultsMessagesLive Query StatisticsExecution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT \* FROM AccountingDocuments WHERE YEAR(PostingDate) = 2024 AND MONTH(PostingDate) = 1

SELECT

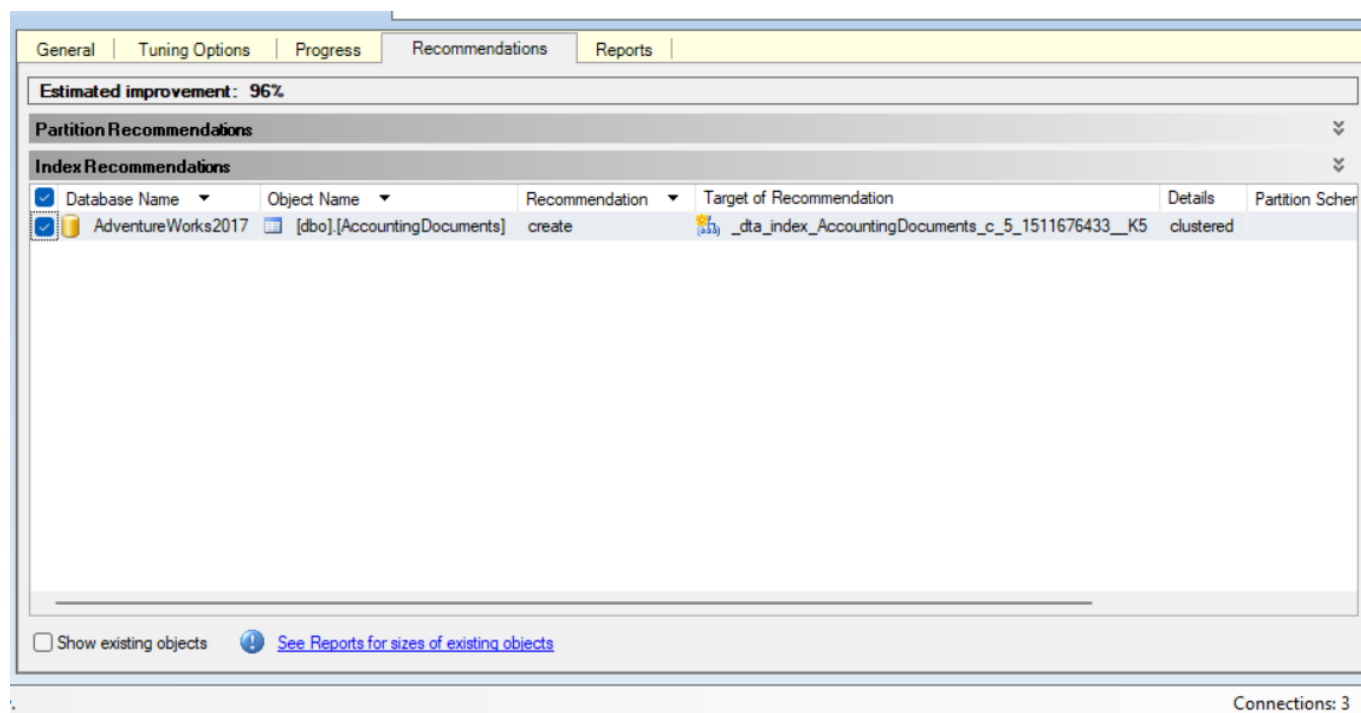
Cost: 0 %

Parallelism  
(Gather Streams)  
Cost: 5 %  
0.186s  
176907 of  
288577 (61%)

Clustered Index Scan (Cluste...  
[AccountingDocuments].[IX\_Ac...  
Cost: 95 %  
0.304s  
176907 of  
288577 (61%)

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:01176 910 rows

## Wyniki z Database Engine Tuning Advisor



### Opis i wnioski

Jak można zauważyć w przypadku, w którym indeks klastrowany nie był założony nastąpiła pełne i równoległe skanowanie całej tabeli (operacji **table scan** - skanowanie, **gathering stream** - zrównoleglenie), które z racji jej rozmiaru wygenerowało ok 50k operacji odczytu logicznego, miało bardzo wysoki koszt 39,8 oraz trwało 400ms, co jest bardzo dużą wartością dla takiego prostego zapytanie. Stało się tak dlatego, że aby znaleźć wszystkie rekordy spełniające warunek klauzuli **WHERE** trzeba było sprawdzić warunek dla każdego pojedynczego rekordu. Po założeniu indeksu klastrowego można zaobserwować znaczący spadek ilości logicznych odczytów (45k => 1.5k), kosztu zapytania (39.8 => 1.32) oraz czasu trwania (400ms => 147ms, ale tu nie było nic zrównoleglane CPU time ok 9x mniejszy). Operacje zrównoleglonego pełnego skanowania zastąpiono operacją wyszukiwania po indeksu (**INDEX SEEK**), z racji, że to jest indeks klastrowany pomimo wybierania wszystkich kolumn (indeks jest założony tylko na jednej) nie ma tu operacji **RID-lookup** ponieważ wszystkie dane fizycznie są indeksowane.

*Pułapka:* Indeks nie zadziałał, na funkcji z kolumny, pomimo takiego samego logicznego znaczenia zapytania. Analizując jego budowę jest to bardzo logiczne, metryka (a więc i jego budowa) indeksu może być całkowicie zmieniona poprzez funkcje.

### Użycie indeksu założonego na dwóch kolumnach

```
SELECT CompanyCode, FiscalYear, Amount, Currency FROM AccountingDocuments
WHERE CompanyCode = 'PL01' AND FiscalYear = 2023;
```

```
CREATE NONCLUSTERED INDEX IX_AD_Company_Year -- 1
ON AccountingDocuments (CompanyCode, FiscalYear);
```

```
CREATE NONCLUSTERED INDEX IX_AD_Company_Year2 -- 2
ON AccountingDocuments(CompanyCode, FiscalYear)
INCLUDE (Amount, Currency);
```

Przed założeniem indeksu

110 %

Results Messages Live Query Statistics Execution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT \* FROM [AccountingDocuments] WHERE [CompanyCode]=@1 AND [FiscalYear]=@2  
Missing Index (Impact 76.0825): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[AccountingDocuments] ([CompanyCode],[FiscalY...

SELECT  
Cost: 0 %

Parallelism  
(Gather Streams)  
Cost: 7 %  
380243 of  
663132 (57%)

Table Scan  
[AccountingDocuments]  
Cost: 93 %  
380243 of  
663132 (57%)

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:03 380 246 rows

Results Messages Live Query Statistics Execution plan

CPU time = 0 ms, elapsed time = 0 ms.  
  
(380243 rows affected)  
Table 'AccountingDocuments'. Scan count 9, logical reads 49226, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0  
  
(3 rows affected)  
  
(1 row affected)  
  
SQL Server Execution Times:  
CPU time = 1420 ms, elapsed time = 841 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

110 %

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:03 380 246 rows

110 %

Results Messages Live Query Statistics Execution plan

	DocumentID	CompanyCode	FiscalYear	DocumentNumber	PostingDate	DocumentDate	Amount	Currency	DocumentType	IsDeleted	CreatedAt
1	800026	PL01	2023	483113	2023-10-18	2024-01-10	3320.52	NULL	SA	0	2025-05-26 15:17:57.933
2	963068	PL01	2023	403534	2023-10-18	2024-11-17	8591.95	PLN	DR	0	2025-05-26 15:17:57.933
3	972964	PL01	2023	559949	2023-10-18	2024-08-27	3412.17	PLN	KR	0	2025-05-26 15:17:57.933
4	1239301	PL01	2023	295201	2023-10-18	2025-03-01	8187.89	EUR	NULL	0	2025-05-26 15:17:57.933
5	1242929	PL01	2023	228902	2023-10-18	2023-05-27	52.42	PLN	KR	0	2025-05-26 15:17:57.933
6	1367459	PL01	2023	946041	2023-10-18	2023-03-07	6414.81	PLN	NULL	0	2025-05-26 15:17:57.933
7	1382360	PL01	2023	136269	2023-10-18	2024-11-25	2264.87	NULL	NULL	0	2025-05-26 15:17:57.933
8	1505182	PL01	2023	531637	2023-10-18	2024-01-06	1691.38	USD	SA	0	2025-05-26 15:17:57.933
9	1715973	PL01	2023	067997	2023-10-18	2024-10-18	2796.52	PLN	KR	0	2025-05-26 15:17:57.933
10	1723475	PL01	2023	543535	2023-10-18	2024-05-14	6712.50	EUR	KR	0	2025-05-26 15:17:57.933

	Values	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExecutions
1		663131.9	NULL	NULL	NULL	41,05121	NULL	NULL	SELECT	0	NULL
2		663131.9	0	1,757534	65	41,05121	[LAB5] [dbo].[AccountingDocuments].[DocumentID]...	NULL	PLAN_ROW	1	1
3	dbo].[AccountingDocuments].[DocumentID]...	663131.9	36,46617	1,570847	65	38,03701	[LAB5] [dbo].[AccountingDocuments].[DocumentID]...	NULL	PLAN_ROW	1	1

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:03 380 243 rows

Po założeniu indeksu 1

110 %

Results Messages Live Query Statistics Execution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT [CompanyCode],[FiscalYear],[Amount],[Currency] FROM [AccountingDocuments] WHERE [CompanyCode]=@1 AND [FiscalYear]=@2  
Missing Index (Impact 89.778): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[AccountingDocuments] ([CompanyCode],[FiscalYe...

SELECT  
Cost: 0 %

Parallelism  
(Gather Streams)  
Cost: 4 %  
0.117s  
380243 of  
382974 (99%)

Table Scan  
[AccountingDocuments]  
Cost: 96 %  
0.134s  
380243 of  
382974 (99%)

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:02 380 246 rows

110 %

Results Messages Live Query Statistics Execution plan

Table 'AccountingDocuments'. Scan count 9, logical reads 49226, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0  
(3 rows affected)  
(1 row affected)  
  
SQL Server Execution Times:  
CPU time = 1111 ms, elapsed time = 290 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
Completion time: 2025-05-28T20:07:49.8270036+02:00

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:02 380 246 rows

110 %

Results Messages Live Query Statistics Execution plan

	CompanyCode	FiscalYear	Amount	Currency
1	PL01	2023	7434.25	USD
2	PL01	2023	82.24	USD
3	PL01	2023	6425.69	NULL
4	PL01	2023	8436.05	PLN
5	PL01	2023	6632.78	EUR
6	PL01	2023	1795.25	NULL
7	PL01	2023	5814.59	PLN
8	PL01	2023	2606.22	PLN
9	PL01	2023	4965.83	NULL
10	PL01	2023	5755.86	EUR

	DefinedValues	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel
1	NULL	382973.6	NULL	NULL	NULL	39,7592	NULL	NULL	SELECT	0
2	NULL	382973.6	0	0.4655207	29	39,7592	[LAB5].[dbo].[AccountingDocuments].[CompanyCod...	NULL	PLAN_ROW	1
3	s). WHERE...	382973.6	36,46617	1,570847	29	38,03701	[LAB5].[dbo].[AccountingDocuments].[CompanyCod...	NULL	PLAN_ROW	1

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:02 380 246 rows

Po założeniu indeksu 2



ResultsMessagesLive Query StatisticsExecution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT [CompanyCode],[FiscalYear],[Amount],[Currency] FROM [AccountingDocuments] WHERE [CompanyCode]=@1 AND [FiscalYear]=@2

Index Seek (NonClustered)  
[AccountingDocuments].[IX\_AD...  
Cost: 100 %  
0.096s  
380243 of  
382974 (99%)

SELECT  
Cost: 0 %

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:02380 245 rows

110 %

ResultsMessagesLive Query StatisticsExecution plan

CPU time = 0 ms, elapsed time = 0 ms.  
  
(380243 rows affected)  
Table 'AccountingDocuments'. Scan count 1, logical reads 1824, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical re  
  
(2 rows affected)  
  
(1 row affected)  
  
SQL Server Execution Times:  
CPU time = 112 ms, elapsed time = 180 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:02380 245 rows

110 %

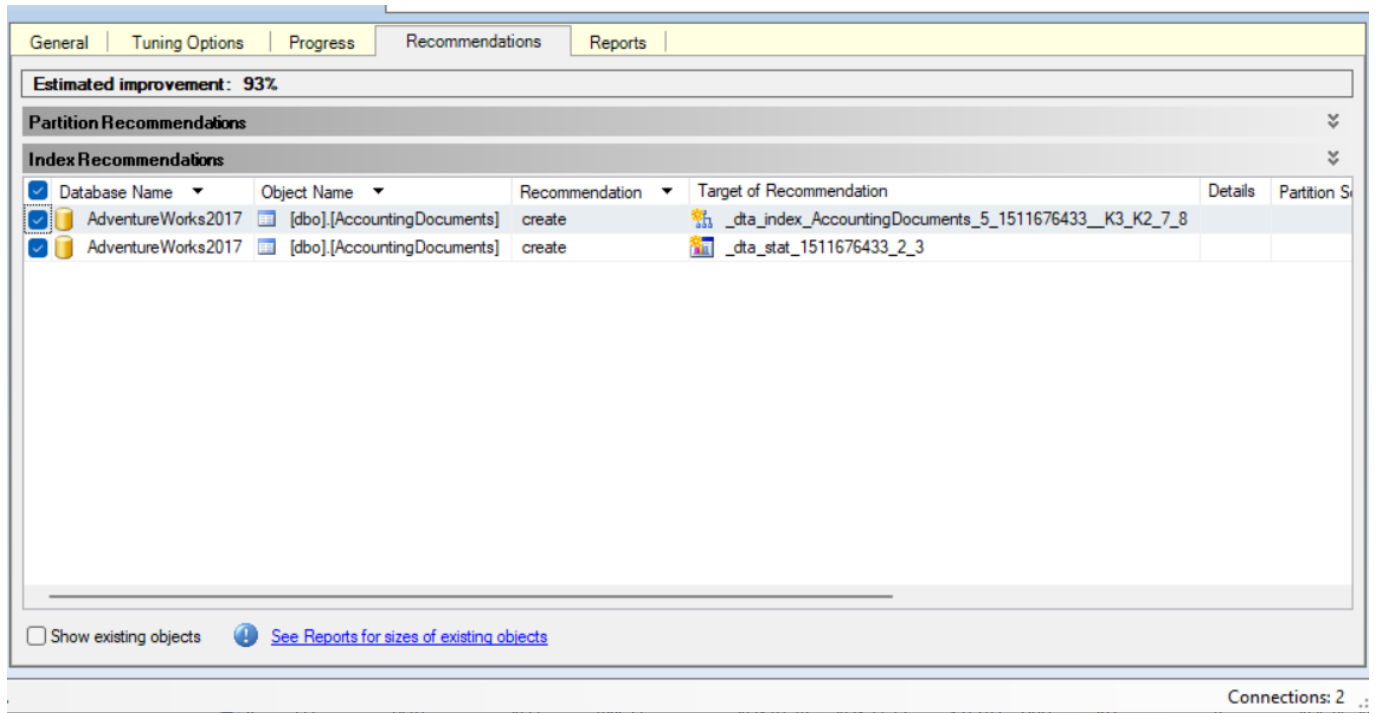
ResultsMessagesLive Query StatisticsExecution plan

	CompanyCode	FiscalYear	Amount	Currency
1	PL01	2023	1084.96	NULL
2	PL01	2023	6585.46	PLN
3	PL01	2023	865.03	PLN
4	PL01	2023	9472.63	PLN
5	PL01	2023	2176.48	NULL
6	PL01	2023	1085.31	EUR
7	PL01	2023	3985.95	USD
8	PL01	2023	3668.46	EUR
9	PL01	2023	4294.71	NULL
10	PL01	2023	629.33	EUR
11	PL01	2023	9498.16	NULL

	Index	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExecutions
1		382973.6	NULL	NULL	NULL	1,722568	NULL	NULL	SELECT	0	NULL
2	[dbo].[AccountingDocuments].[CompanyCod...	382973.6	1,30114	0,4214279	29	1,722568	[LAB5].[dbo].[AccountingDocuments].[CompanyCod...	NULL	PLAN_ROW	0	1

Query executed successfully.localhost (15.0 RTM)SA (57)LAB500:00:02380 243 rows

Wyniki z Database Engine Tuning Advisor



## Opis i wnioski

Analizując plany poszczególnych zapytań jako pierwszy można wysnuć wniosek, że po wdrożeniu indeksu nr1 plan zapytania pozostał dokładnie taki sam jak w przypadku, kiedy nie było żadnego indeksu. Stało się tak dlatego, że pomimo użycie w tej sytuacji może się wydawać na pierwszy rzut oka logiczne, ponieważ indeks jest założony dokładnie na tych kolumnach, które występują w klauzuli **WHERE**, ale jednak jako, że wybieramy dodatkowe 2 kolumny, których indeks nie obejmuje to po operacji **INDEX SEEK** pomimo, że jest bardzo efektywna musiałaby nastąpić operacja **RID-lookup**, która jest bardzo kosztowna, kiedy jest bardzo dużo rekordów w wynikach (a tak jest w tym wypadku, duża tabela i dosyć ogólny warunek), w skutek czego kompilator zdecydował, żeby nie używać tego indeksu w tym wypadku. Tak, więc w przypadku 2 pierwszych zapytań nastąpiło pełne skanowanie tabeli w celu określenie warunku klauzuli **WHERE**, co spowodowało 49k operacji logicznego odczytu, wygenerowało koszt 39,8 oraz trwało 290ms. W trzecim zapytaniu w odróżnieniu od poprzednich został użyty indeks, ponieważ klauzula **INCLUDE** rozwiązała problem z operacją **RID-lookup** (nie jest ona już potrzebna, ponieważ wszystkie kolumny które wybieramy fizycznie znajdują się w indeksie), więc nastąpiła efektywna operacja **INDEX SEEK**, która pozwoliła zredukować ilość odczytów logicznych (50k => 1.5k), koszt (39,8 => 1.7) oraz nieznacznie czas (290ms => 180ms, ale tu znowu pełne skanowanie było zrównoległone, 9 krotna różnica w CPU time).

## Użycie indeksu warunkowego

```
SELECT CompanyCode, PostingDate, Amount, Currency FROM AccountingDocuments -- 1
WHERE IsDeleted = 0 AND CompanyCode = 'PL01' AND PostingDate > '2024-01-01';

SELECT CompanyCode, PostingDate, Amount, Currency FROM AccountingDocuments -- 2
WHERE CompanyCode = 'PL01' AND PostingDate > '2024-01-01';

CREATE NONCLUSTERED INDEX IX_AD_NotDeleted
ON AccountingDocuments(CompanyCode, PostingDate)
INCLUDE (Amount, Currency)
WHERE IsDeleted = 0;
```

Bad practise:

```
SELECT CompanyCode, PostingDate, Amount, Currency FROM AccountingDocuments WITH
(INDEX(IX_AD_NotDeleted))
WHERE PostingDate > '2024-01-01';
```

SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
Msg 8622, Level 16, State 1, Line 78  
Query processor could not produce a query plan because of the hints defined in this query. Resubmit the query without specifying any hints and without using SET FORCEPI  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms

Przed założeniem indeksu

110 %

Results Messages Live Query Statistics Execution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT [CompanyCode],[PostingDate],[Amount],[Currency] FROM [AccountingDocuments] WHERE ([IsDeleted]=@1 AND [CompanyCode]=@2 AND [PostingDate]>@3  
Missing Index (Impact 85.3698): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[AccountingDocuments] ([CompanyCode],[IsDelet...

SELECT

Cost: 0 %

Parallelism  
(Gather Streams)

Cost: 9 %  
0.443s  
923093 of  
1344390 (68%)

Table Scan  
[AccountingDocuments]  
Cost: 92 %  
0.129s  
923093 of  
1344390 (68%)

Query executed successfully. localhost (15.0 RTM) SA (57) LABS 00:00:05 923 096 rows

110 %

Results Messages Live Query Statistics Execution plan

(923093 rows affected)  
Table 'AccountingDocuments'. Scan count 9, logical reads 49226, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical  
  
(3 rows affected)  
  
(1 row affected)  
  
SQL Server Execution Times:  
CPU time = 1664 ms, elapsed time = 818 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
Completion time: 2025-05-28T20:29:57.4860356+02:00

Query executed successfully. localhost (15.0 RTM) SA (57) LABS 00:00:05 923 096 rows

110 %

Results Messages Live Query Statistics Execution plan

	CompanyCode	PostingDate	Amount	Currency
1	PL01	2024-10-24	8524.66	USD
2	PL01	2024-10-24	8813.37	EUR
3	PL01	2024-10-24	9673.33	NULL
4	PL01	2024-10-24	7434.25	USD
5	PL01	2024-10-24	901.60	PLN
6	PL01	2024-10-24	506.51	NULL
7	PL01	2024-10-24	82.24	USD
8	PL01	2024-10-24	4691.87	USD
9	PL01	2024-10-24	9712.21	NULL
10	PL01	2024-10-24	575.81	EUR

	Indexes	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExecutions
1		1344392	NULL	NULL	NULL	41,26151	NULL	NULL	SELECT	0	NULL
2		1344392	0	1,539428	28	41,26151	[LAB5].[dbo].[AccountingDocuments].[CompanyCod...	NULL	PLAN_ROW	1	1
3		1344392	36,46617	1,570847	29	38,03701	[LAB5].[dbo].[AccountingDocuments].[CompanyCod...	NULL	PLAN_ROW	1	1

Query executed successfully. localhost (15.0 RTM) SA (57) LABS 00:00:05 923 093 rows

Po założeniu indeksu dla 1

110 %

Results Messages Live Query Statistics Execution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT [CompanyCode],[PostingDate],[Amount],[Currency] FROM [AccountingDocuments] WHERE [IsDeleted]=@1 AND [CompanyCode]=@2 AND [PostingDate]>@3

Index Seek (NonClustered)  
([AccountingDocuments].[IX\_AD...]  
Cost: 100 %  
0.293s  
923093 of  
1361880 (67%)

SELECT  
Cost: 0 %

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:06 923 095 rows

110 %

Results Messages Live Query Statistics Execution plan

Table 'AccountingDocuments'. Scan count 1, logical reads 4310, physical reads 0, page server reads 0, read-ahead reads 6, page server read-ahead reads 0, lob logical re  
(2 rows affected)  
(1 row affected)  
  
SQL Server Execution Times:  
CPU time = 424 ms, elapsed time = 1073 ms.  
SQL Server parse and compile time:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
SQL Server Execution Times:  
CPU time = 0 ms, elapsed time = 0 ms.  
  
Completion time: 2025-05-28T20:31:07.7389955+02:00

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:06 923 095 rows

110 %

Results Messages Live Query Statistics Execution plan

Click to select the whole row

	CompanyCode	PostingDate	Amount	Currency
1	PL01	2024-01-02	1840.65	NULL
2	PL01	2024-01-02	39.73	NULL
3	PL01	2024-01-02	3983.87	EUR
4	PL01	2024-01-02	2574.52	USD
5	PL01	2024-01-02	678.52	NULL
6	PL01	2024-01-02	5468.92	PLN
7	PL01	2024-01-02	5354.53	NULL
8	PL01	2024-01-02	9625.14	EUR
9	PL01	2024-01-02	8315.32	EUR
10	PL01	2024-01-02	9570.35	PLN
11	PL01	2024-01-02	7475.09	USD

	DefinedValues	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type
1	NULL	1361875	NULL	NULL	NULL	5,769493	NULL	NULL	SELECT
2	ocuments].[IX_A... [LAB5].[dbo].[AccountingDocuments].[CompanyCod...	1361875	4,271273	1,49822	28	5,769493	[LAB5].[dbo].[AccountingDocuments].[CompanyCod...	NULL	PLAN_ROW

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:06 923 093 rows

Po założeniu indeksu dla 2

Results Messages Live Query Statistics Execution plan

Query 1: Query cost (relative to the batch): 100%  
SELECT [CompanyCode],[PostingDate],[Amount],[Currency] FROM [AccountingDocuments] WHERE [CompanyCode]=@1 AND [PostingDate]>@2

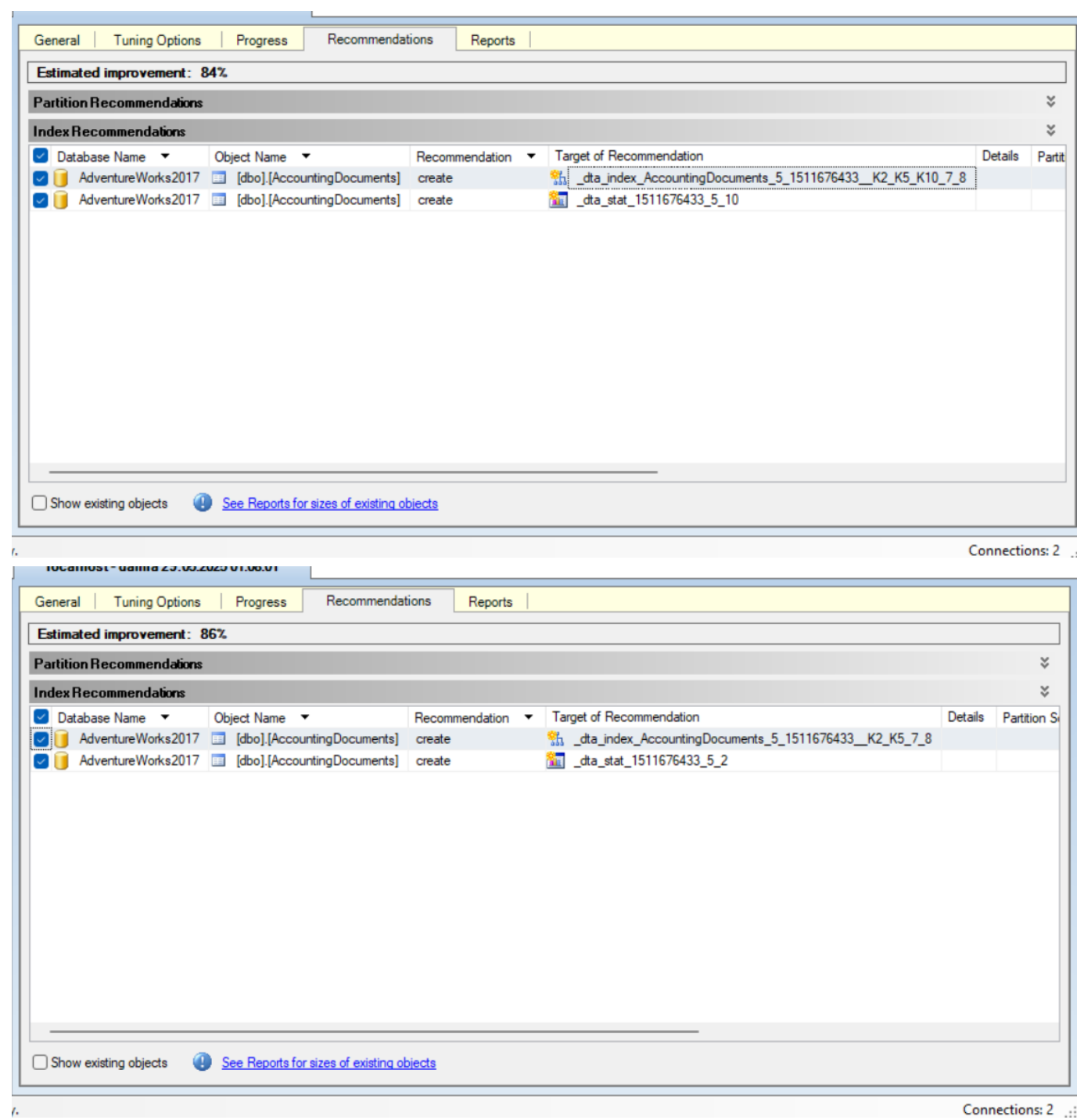
Parallelism  
(Gather Streams)  
Cost: 7 %  
0.491s  
971930 of  
1361880 (71%)

Table Scan  
([AccountingDocuments])  
Cost: 93 %  
0.108s  
971930 of  
1361880 (71%)

SELECT  
Cost: 0 %

Query executed successfully. localhost (15.0 RTM) SA (57) LAB5 00:00:06 971 933 rows

Wyniki z Database Engine Tuning Advisor



Opis i wnioski

Pierwszym zaobserwowanym wnioskiem w przypadku indeksów warunkowych, powinien być fakt, że w przypadku braku takiego samego warunku w klauzuli **WHERE** nie jest on używane. Wynika to bezpośrednio z jego budowy (w jego skład nie wchodzi wiersze wykluczone przez **WHERE**). Wtedy zarówno z jak i bez indeksu zapytanie zachowuje się tak samo, następuje pełne skanowanie tabeli z danymi, co skutkuje 50k odczytów logicznych, kosztem 41, oraz czasem wykonania 820ms. Z kolei kiedy warunek, który został użyty znajduje się w (może być też jakiś inny, jeśli jest stworzony indeks to raczej logicznie powinien) w klauzuli **WHERE** to indeks jest użyty dokładnie w ten sam sposób, jak taki bez warunków. Następuje operacja **INDEX SEEK**, co pozwala zredukować odczyty logiczne 50k => 4.3, czas się zwiększył 820ms => 1040ms (pewnie to przez ilość wątków), oraz koszt 41 => 4.7. Trzeba też również zwrócić uwagę, na fakt, że budowa i utrzymanie takiego indeksu (zwłaszcza utrzymanie) jest tańsze niż takiego zwykłego. W tej sytuacji jest to ciężkie do zauważenia (jest ok 5% usuniętych), ale zdarzają się sytuację, gdzie jest ok20% usuniętych.

```
SELECT CompanyCode, DocumentDate, SUM(Amount) AS Total
FROM AccountingDocuments
GROUP BY CompanyCode, DocumentDate;
```

```
CREATE CLUSTERED COLUMNSTORE INDEX IX_AD_Columnstore
ON AccountingDocuments;
```

Results

Messages

Live Query Statistics

Execution plan

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

SELECT CompanyCode, DocumentDate, SUM(Amount) AS Total FROM AccountingDocuments GROUP BY CompanyCode, DocumentDate

SELECT

Cost: 0 %

Parallelism  
(Gather Streams)

Cost: 0 %

0.189s

4000 of 4012 (99%)

Compute Scalar  
(Aggregate)

Cost: 0 %

0.000s

4000 of 4012 (99%)

Hash Match  
(Aggregate)

Cost: 3 %

0.042s

4000 of 4012 (99%)

Table Scan (Heap)  
[AccountingDocuments]

Cost: 97 %

0.153s

5712100 of 5712100 (100%)

Query executed successfully.

localhost (15.0 RTM) | SA (57) | LAB5 | 00:00:01 | 4 005 rows

110 %

Results

Messages

Live Query Statistics

Execution plan

CPU time = 0 ms, elapsed time = 0 ms.

(4000 rows affected)

Table 'AccountingDocuments'. Scan count 9, logical reads 49226, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0

Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0

(5 rows affected)

(1 row affected)

SQL Server Execution Times:

CPU time = 1431 ms, elapsed time = 193 ms.

SQL Server parse and compile time:

CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:

CPU time = 0 ms, elapsed time = 0 ms.

Query executed successfully.

localhost (15.0 RTM) | SA (57) | LAB5 | 00:00:01 | 4 005 rows

110 %

Results

Messages

Live Query Statistics

Execution plan

	CompanyCode	DocumentDate	Total
1	PL01	2023-05-08	9493077.29
2	PL01	2023-08-10	9570735.31
3	US03	2024-11-13	4244753.09
4	PL01	2023-03-09	9499756.27
5	DE02	2025-02-05	6349608.92
6	DE02	2025-03-11	6419224.59
7	US03	2024-09-23	4467296.13
8	PL01	2023-03-26	9561497.75
9	DE02	2024-01-10	6356869.45

Query executed successfully.

localhost (15.0 RTM) | SA (57) | LAB5 | 00:00:01 | 4 005 rows

110 %

Results

Messages

Live Query Statistics

Execution plan

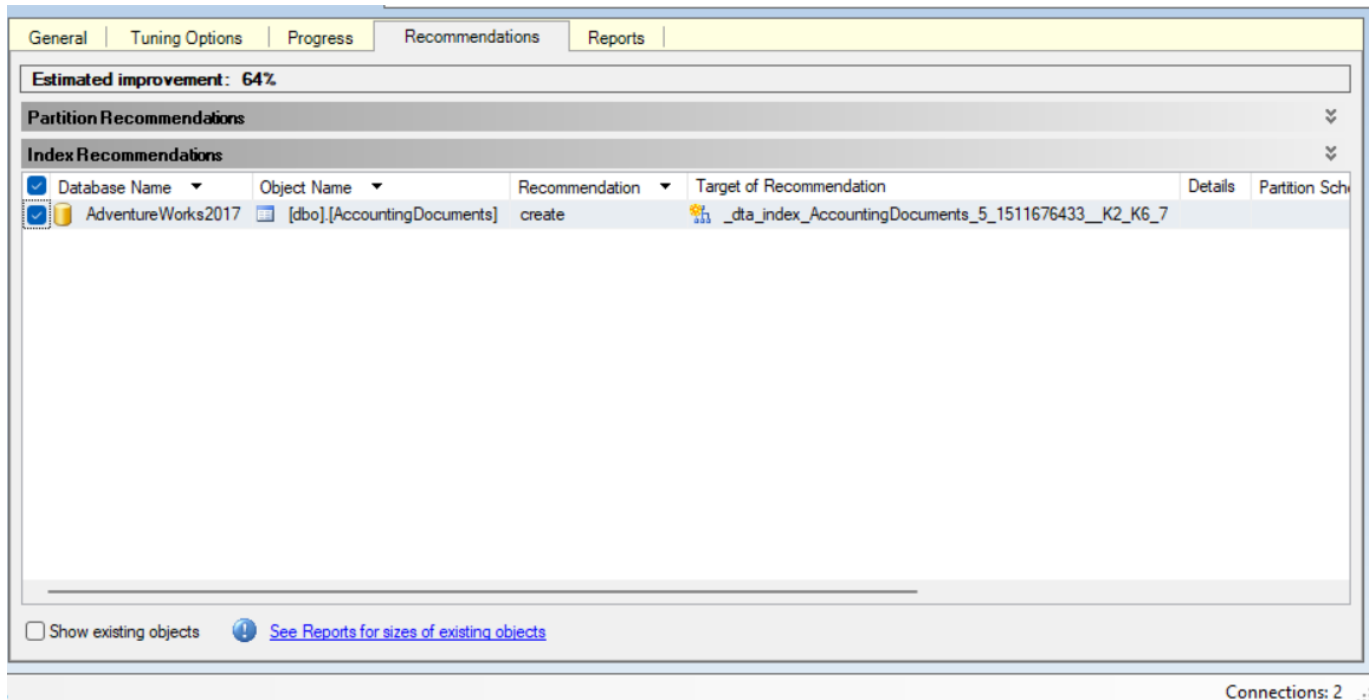
	DefinedValues	EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings
1	NULL	4012	NULL	NULL	NULL	39,08966	NULL	Warning: Null value is being produced by this query. The null value will not be included in any aggregate result.
2	NULL	4012	0	0.03287057	33	39,08966	[LAB5].[dbo].[AccountingDocuments].[CompanyCode]	Warning: Null value is being produced by this query. The null value will not be included in any aggregate result.
3	SE WHEN [Expr1008]=(0) THEN...	4012	0	0	33	39,05679	[LAB5].[dbo].[AccountingDocuments].[CompanyCode]	Warning: Null value is being produced by this query. The null value will not be included in any aggregate result.
4	countingDocuments].[CompanyC...	4012	0	1.019837	33	39,05679	[LAB5].[dbo].[AccountingDocuments].[CompanyCode]	Warning: Null value is being produced by this query. The null value will not be included in any aggregate result.
5	AccountingDocuments))	5712100	36,46609	1,570867	25	38,03695	[LAB5].[dbo].[AccountingDocuments].[CompanyCode]	Warning: Null value is being produced by this query. The null value will not be included in any aggregate result.

Query executed successfully.

localhost (15.0 RTM) | SA (57) | LAB5 | 00:00:01 | 4 005 rows

## 30 / 34





### Opis i wnioski

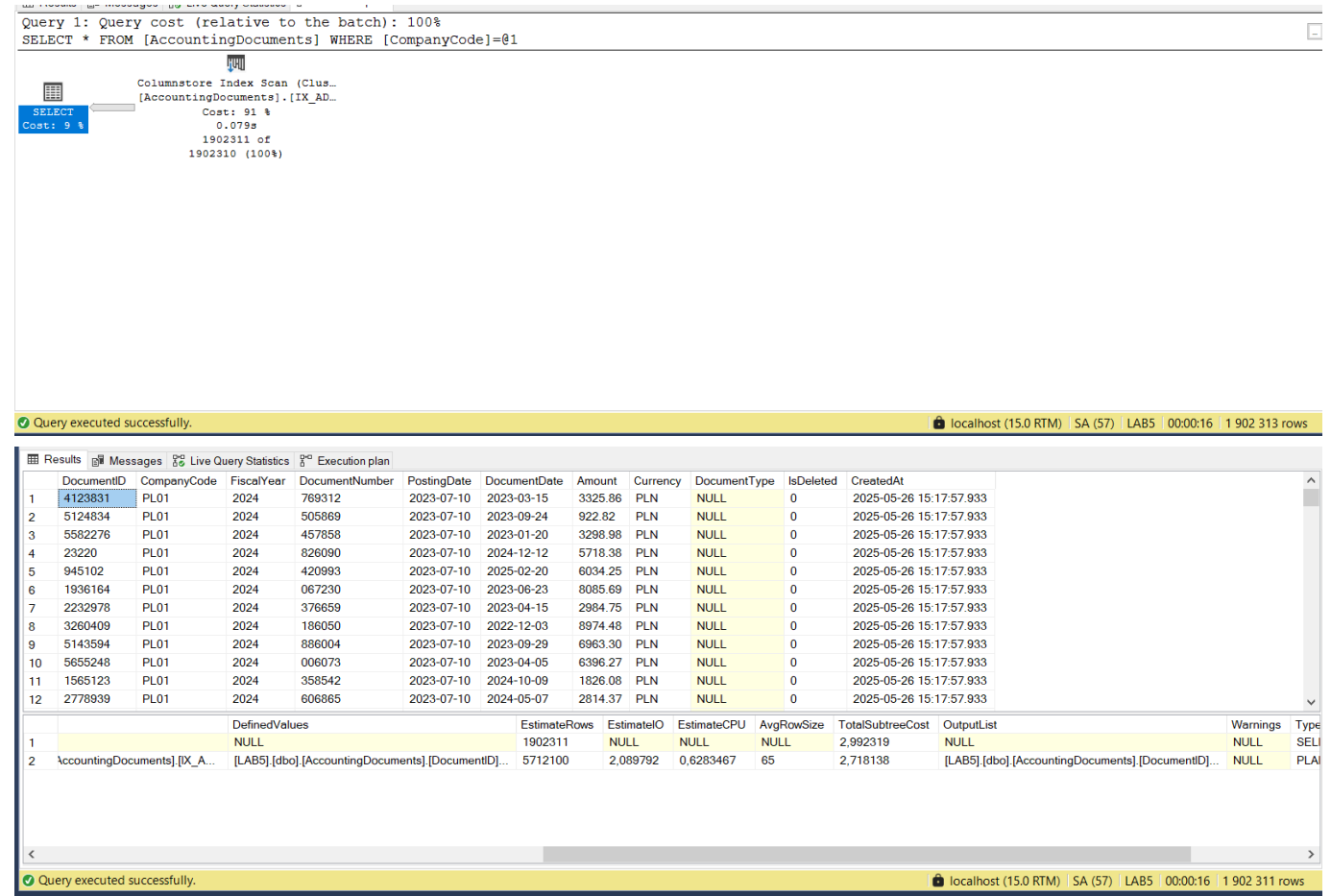
Indeks **columnstore** z racji na dość mocną ingerencję samej struktury tabeli pozwolił na znaczące przyśpieszenie i zmniejszenie kosztów w porównaniu do jego braku (typowa sytuacja - pełne równoległe skanowanie), kosztu z 39 => 1.7, czasu wykonania z 193ms => 42ms (tutaj zapytanie zarówno z użyciem indeksu jak i bez jest obliczane równoległe) oraz liczby logicznych odczytów z 49k do 8sztuk. Obserwujemy tak spektakularną poprawę z powodu tego, że indeksy columnstore są zoorientowane na kolumnach, przez co są niezwykle efektywne w operacjach agregujących (group by).

### Wymuszenie użycia indeksu, kiedy nie jest to opłacalne

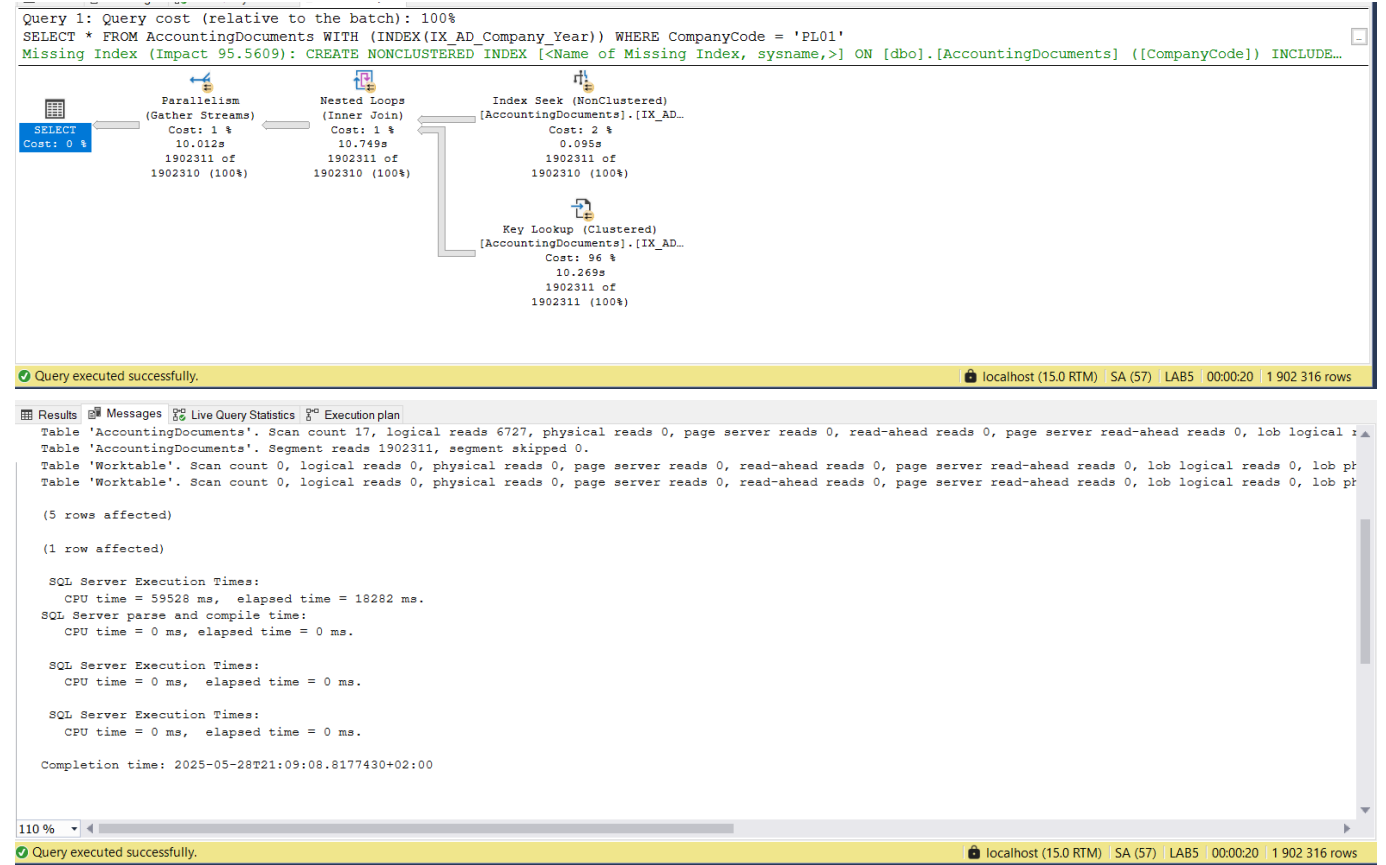
```
SELECT * FROM AccountingDocuments WITH (INDEX(IX_AD_Company_Year))
WHERE CompanyCode = 'PL01';
```

### Bez użycia indeksu





Z użyciem indeksu



Results												Messages												Live Query Statistics												Execution plan											
	DocumentID	CompanyCode	FiscalYear	DocumentNumber	PostingDate	DocumentDate	Amount	Currency	DocumentType	IsDeleted	CreatedAt																																				
1	1857541	PL01	2021	909371	2024-08-25	2023-11-16	8638.26	EUR	SA	0	2025-05-26 15:17:57.933																																				
2	4210482	PL01	2021	153830	2024-08-26	2024-05-18	4954.88	EUR	SA	0	2025-05-26 15:17:57.933																																				
3	4170904	PL01	2021	921710	2024-08-26	2022-10-09	6523.89	EUR	SA	0	2025-05-26 15:17:57.933																																				
4	4381404	PL01	2021	468519	2024-08-26	2023-01-10	2102.86	EUR	SA	0	2025-05-26 15:17:57.933																																				
5	539157	PL01	2021	722496	2024-04-23	2023-01-23	7469.65	EUR	SA	0	2025-05-26 15:17:57.933																																				
6	5711091	PL01	2021	179361	2024-04-23	2024-01-03	7548.09	EUR	SA	0	2025-05-26 15:17:57.933																																				
7	3212165	PL01	2021	227751	2024-04-23	2024-01-18	3461.02	EUR	SA	0	2025-05-26 15:17:57.933																																				
8	2058651	PL01	2021	014839	2024-04-23	2025-04-30	6248.49	EUR	SA	0	2025-05-26 15:17:57.933																																				
9	1982539	PL01	2021	653574	2024-04-23	2023-07-05	4966.08	EUR	SA	0	2025-05-26 15:17:57.933																																				
												EstimateRows	EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExecutions																										
1												1902311	NULL	NULL	NULL	339,3342	NULL	NULL	SELECT	0	NULL																										
2												1902311	0	4,988538	65	339,3342	[LAB5] [dbo].[AccountingDocuments].[DocumentID], ...	NULL	PLAN_ROW	1	1																										
3												1902311	0	1,987915	65	334,3457	[LAB5] [dbo].[AccountingDocuments].[DocumentID], ...	NULL	PLAN_ROW	1	1																										
4	000, [LAB5] [dbo].[AccountingDocum...											1902311	4,67297	0,5231748	25	5,196145	[ColStoreLoc1000], [LAB5] [dbo].[AccountingDocum...	NULL	PLAN_ROW	1	1																										
5	AccountingDocuments].[DocumentID], ...											1	0,003125	0,0001581	57	327,1616	[LAB5] [dbo].[AccountingDocuments].[DocumentID], ...	NULL	PLAN_ROW	1	1902311																										
Query executed successfully.																																															
localhost (15.0 RTM)   SA (57)   LAB5   00:00:20   1 902 311 rows																																															

W tej sytuacji można zaobserwować dlaczego optymalizator zdecydował się nie użyć 1 indeksu w drugim podpunkcie. Operacja **KEY-Lookup** (przyłączenia do wyników kolumn, które nie znajdują się w indeksie) jest tak kosztowna, że gdy w wyniku zapytania jest wiele rekordów to może bardzo podnieść koszt zapytania 2.7 => 340, co z tym idzie czas również.

zadanie	pkt
1	2
2	2
3	2
4	2
5	5
razem	15