SAVETI ZA PISANJE PERFORMANTNIH TRANSACT-SQL UPITA

MILOŠ RADIVOJEVIĆ, MICROSOFT DATA PLATFORM MVP

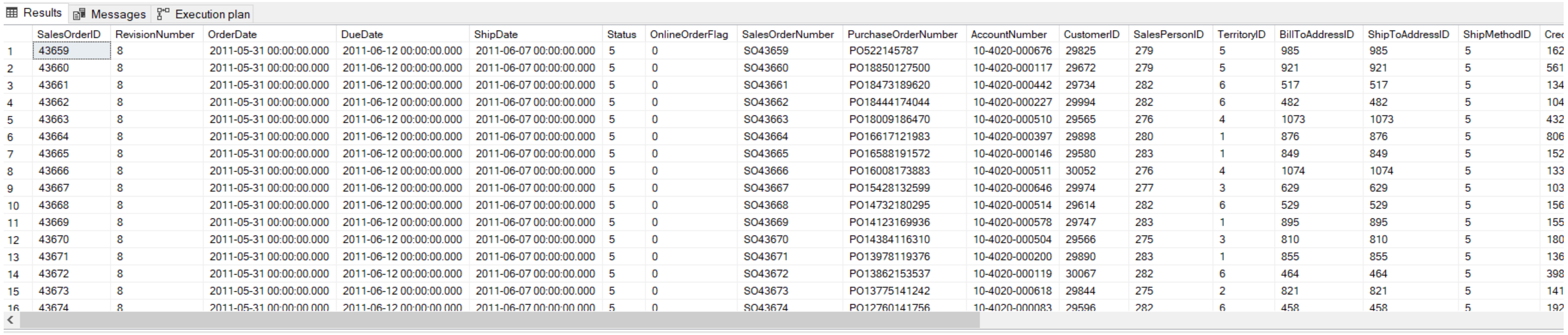
1 FUNKCIJE I ARITMETIČKE OPERACIJE

Testna tabela - Orders

```sql
USE AdventureWorks2019
GO
-- Create and populate the dbo.Orders table
DROP TABLE IF EXISTS dbo.Orders;
GO
SELECT * INTO dbo.Orders FROM Sales.SalesOrderHeader;

ALTER TABLE dbo.Orders ADD CONSTRAINT PK_Orders PRIMARY KEY (SalesOrderID);
GO
```

31 465 rows



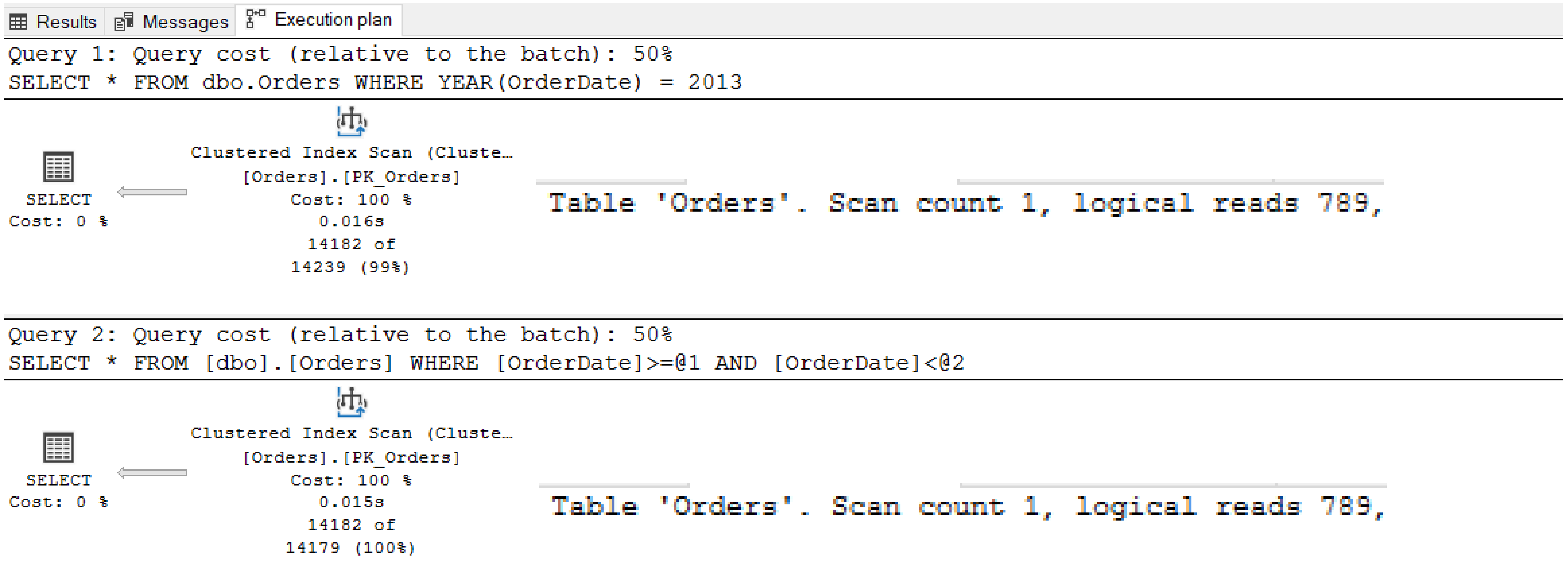
Results | Messages | Execution plan

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerID	SalesPersonID	TerritoryID	BillToAddressID	ShipToAddressID	ShipMethodID	Cre
1	43659	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43659	PO522145787	10-4020-000676	29825	279	5	985	985	5	162
2	43660	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43660	PO18850127500	10-4020-000117	29672	279	5	921	921	5	561
3	43661	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43661	PO18473189620	10-4020-000442	29734	282	6	517	517	5	134
4	43662	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43662	PO18444174044	10-4020-000227	29994	282	6	482	482	5	104
5	43663	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43663	PO18009186470	10-4020-000510	29565	276	4	1073	1073	5	432
6	43664	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43664	PO16617121983	10-4020-000397	29898	280	1	876	876	5	806
7	43665	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43665	PO16588191572	10-4020-000146	29580	283	1	849	849	5	152
8	43666	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43666	PO16008173883	10-4020-000511	30052	276	4	1074	1074	5	133
9	43667	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43667	PO15428132599	10-4020-000646	29974	277	3	629	629	5	103
10	43668	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43668	PO14732180295	10-4020-000514	29614	282	6	529	529	5	156
11	43669	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43669	PO14123169936	10-4020-000578	29747	283	1	895	895	5	155
12	43670	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43670	PO14384116310	10-4020-000504	29566	275	3	810	810	5	180
13	43671	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43671	PO13978119376	10-4020-000200	29890	283	1	855	855	5	136
14	43672	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43672	PO13862153537	10-4020-000119	30067	282	6	464	464	5	398
15	43673	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43673	PO13775141242	10-4020-000618	29844	275	2	821	821	5	141
16	43674	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43674	PO12760141756	10-4020-000083	29596	282	6	458	458	5	192

YEAR funkcija vs. >= & <

svejedno

```sql
SELECT * FROM dbo.Orders WHERE YEAR(OrderDate) = 2013;
SELECT * FROM dbo.Orders WHERE OrderDate >= '20130101' AND OrderDate < '20140101';
```



```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Orders WHERE YEAR(OrderDate) = 2013

SELECT        Clustered Index Scan (Cluste…
Cost: 0 %       [Orders].[PK_Orders]
                  Cost: 100 %
                    0.016s
                   14182 of
                 14239 (99%)

Table 'Orders'. Scan count 1, logical reads 789,

Query 2: Query cost (relative to the batch): 50%
SELECT * FROM [dbo].[Orders] WHERE [OrderDate]>=@1 AND [OrderDate]<@2

SELECT        Clustered Index Scan (Cluste…
Cost: 0 %       [Orders].[PK_Orders]
                  Cost: 100 %
                    0.015s
                   14182 of
                 14179 (100%)

Table 'Orders'. Scan count 1, logical reads 789,
```

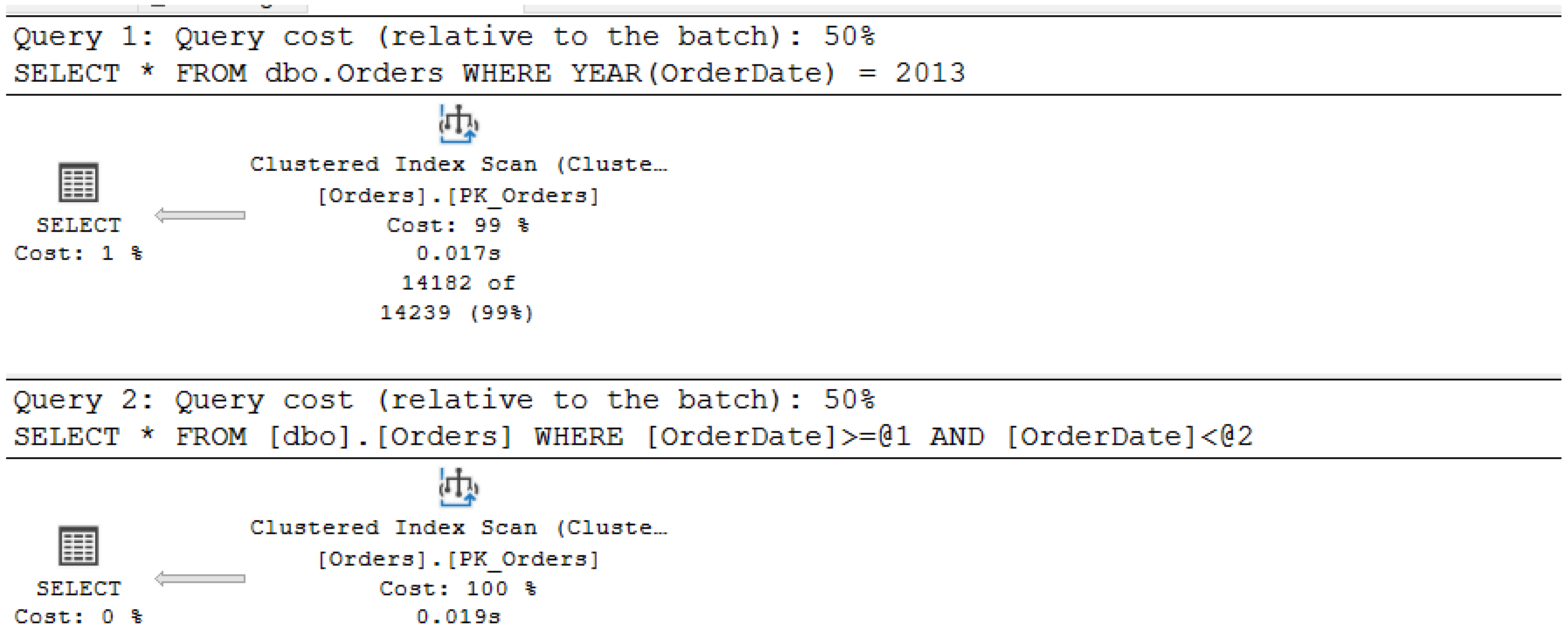
Zašto je svejedno?

- Tabela poseduje samo klaster indeks
- Kolona koja se nalazi u WHERE klauzuli nije indeksirana
- SQL Server ima samo jednu mogućnost da generiše rezultat upita, a to je skeniranje tabele
- Zbog svega toga, u ovom slučaju je nevažno kako je napisan upit

YEAR funkcija vs. >= & <

svejedno

```
CREATE INDEX ix1 ON dbo.Orders(OrderDate);
GO
SELECT * FROM dbo.Orders WHERE YEAR(OrderDate) = 2013;
SELECT * FROM dbo.Orders WHERE OrderDate >= '20130101' AND OrderDate < '20140101';
GO
```

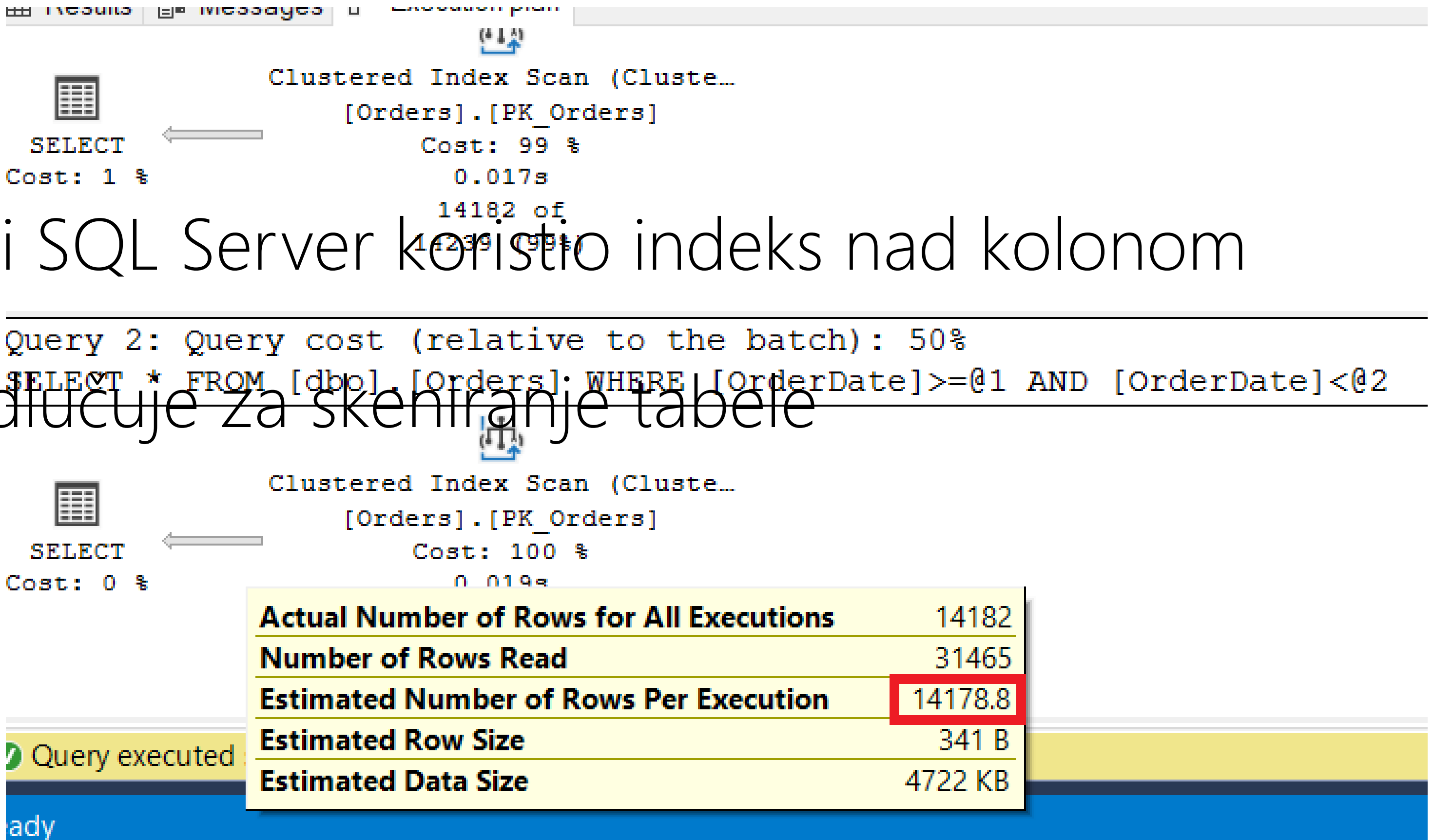


```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Orders WHERE YEAR(OrderDate) = 2013
```

```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM [dbo].[Orders] WHERE [OrderDate]>=@1 AND [OrderDate]<@2
```

Zašto je svejedno?

- Sada tabele poseduje non-cluster indeks, ali
- SQL Server procenjuje da će upit vratiti preko 14 hiljada redova, to je 45% svih podataka



```
SELECT                Clustered Index Scan (Cluste…
Cost: 1 %                 [Orders].[PK_Orders]
                               Cost: 99 %
                                 0.017s
                                14182 of
                              14239 (99%)

Query 2: Query cost (relative to the batch): 50%
SELECT * FROM [dbo].[Orders] WHERE [OrderDate]>=@1 AND [OrderDate]<@2

SELECT                Clustered Index Scan (Cluste…
Cost: 0 %                 [Orders].[PK_Orders]
                               Cost: 100 %
                                 0.019s
```

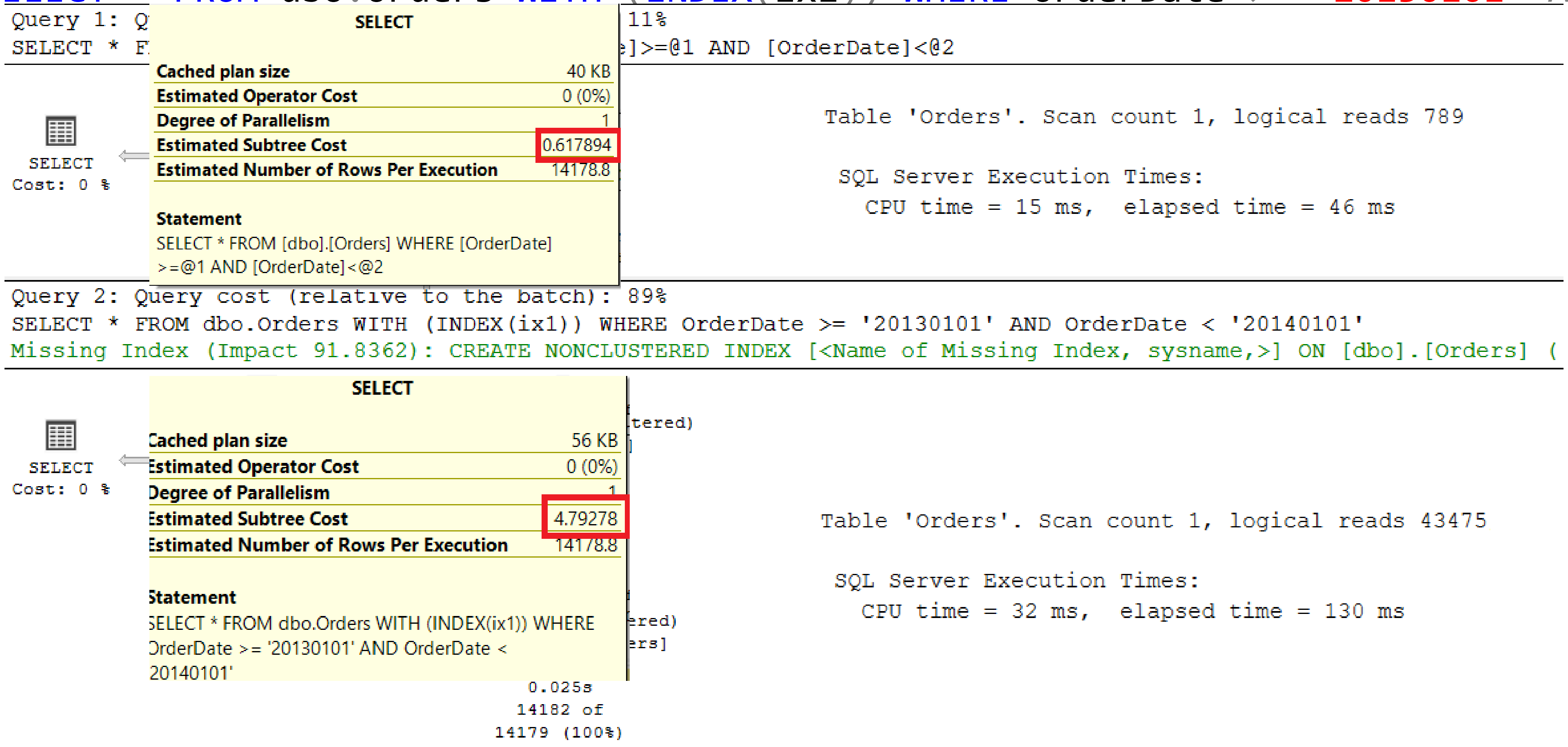
Actual Number of Rows for All Executions	14182
Number of Rows Read	31465
Estimated Number of Rows Per Execution	14178.8
Estimated Row Size	341 B
Estimated Data Size	4722 KB

Query executed

- To je previše redova da bi SQL Server koristio indeks nad kolonom OrderDate efikasno
- SQL Server se ponovo odlučuje za skeniranje tabele

Zašto SQL Server ignoriše indeks?

```
SELECT * FROM dbo.Orders WHERE OrderDate >= '20130101' AND OrderDate < '20140101';
SELECT * FROM dbo.Orders WITH (INDEX(ix1)) WHERE OrderDate >= '20130101' AND
O
```



```
Query 1: Q                                         11%
SELECT * F                                         ]>=@1 AND [OrderDate]<@2
```

SELECT

Cached plan size	40 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	1
Estimated Subtree Cost	0.617894
Estimated Number of Rows Per Execution	14178.8

Statement
SELECT * FROM [dbo].[Orders] WHERE [OrderDate] >=@1 AND [OrderDate]<@2

```
SELECT
Cost: 0 %
```

```
Table 'Orders'. Scan count 1, logical reads 789

 SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 46 ms
```

```
Query 2: Query cost (relative to the batch): 89%
SELECT * FROM dbo.Orders WITH (INDEX(ix1)) WHERE OrderDate >= '20130101' AND OrderDate < '20140101'
Missing Index (Impact 91.8362): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[Orders] (
```

SELECT

Cached plan size	56 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	1
Estimated Subtree Cost	4.79278
Estimated Number of Rows Per Execution	14178.8

Statement
SELECT * FROM dbo.Orders WITH (INDEX(ix1)) WHERE OrderDate >= '20130101' AND OrderDate < '20140101'

```
SELECT
Cost: 0 %
```

```
tered)
ered)
ers]
0.025s
14182 of
14179 (100%)
```

```
Table 'Orders'. Scan count 1, logical reads 43475

 SQL Server Execution Times:
   CPU time = 32 ms,  elapsed time = 130 ms
```

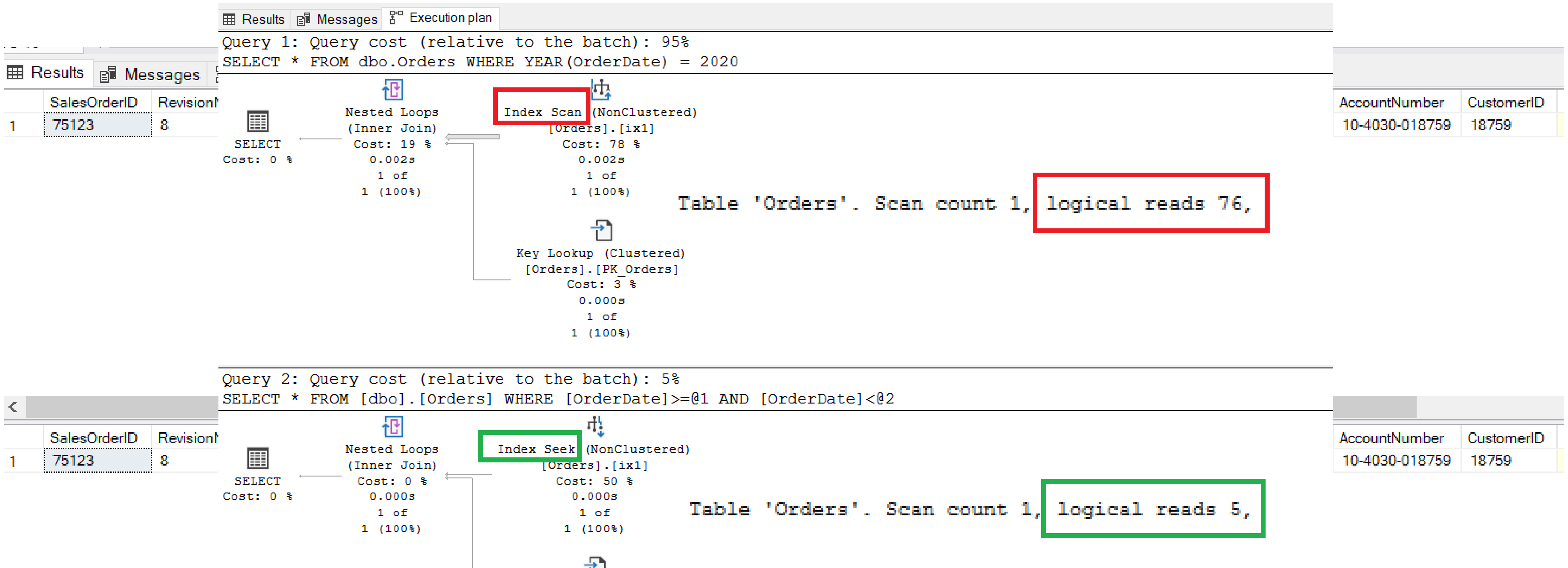
Manipulisanje testnom tabelom

```
UPDATE dbo.Orders SET OrderDate = '20200413', DueDate= '20200414',
      ShipDate= '20200415'
WHERE SalesOrderID = 75123;
```

- Sada za 2020. imamo tačno jedan red u tabeli Orders

YEAR funkcija vs. >= & <

```sql
SELECT * FROM dbo.Orders WHERE YEAR(OrderDate) = 2020;
SELECT * FROM dbo.Orders WHERE OrderDate >= '20200101' AND OrderDate < '20210101';
```



```
Query 1: Query cost (relative to the batch): 95%
SELECT * FROM dbo.Orders WHERE YEAR(OrderDate) = 2020
```

	SalesOrderID	Revision	...	AccountNumber	CustomerID
1	75123	8	...	10-4030-018759	18759

```
SELECT        Nested Loops       Index Scan (NonClustered)
Cost: 0 %     (Inner Join)       [Orders].[ix1]
              Cost: 19 %         Cost: 78 %
              0.002s             0.002s
              1 of               1 of
              1 (100%)           1 (100%)

                                 Key Lookup (Clustered)
                                 [Orders].[PK_Orders]
                                 Cost: 3 %
                                 0.000s
                                 1 of
                                 1 (100%)
```

```
Table 'Orders'. Scan count 1, logical reads 76,
```

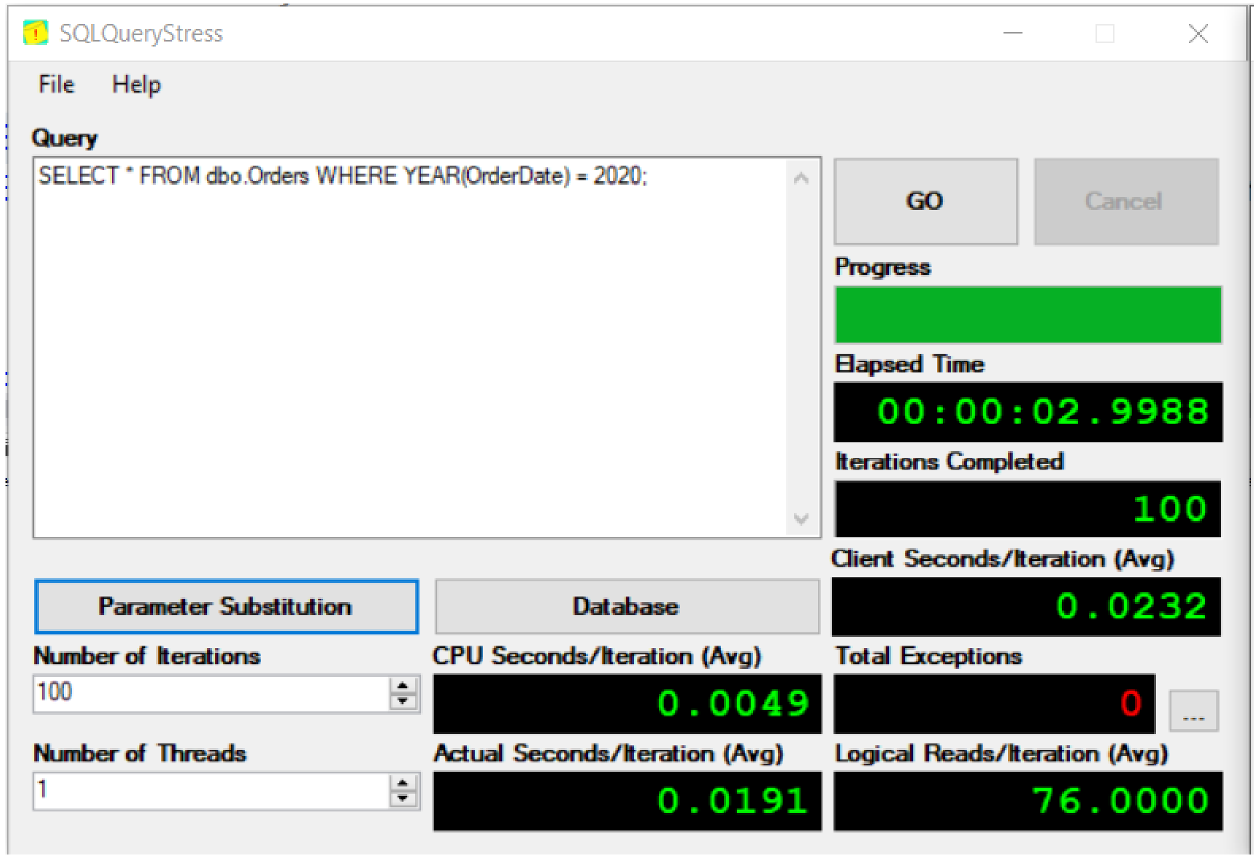
```
Query 2: Query cost (relative to the batch): 5%
SELECT * FROM [dbo].[Orders] WHERE [OrderDate]>=@1 AND [OrderDate]<@2
```

	SalesOrderID	Revision	...	AccountNumber	CustomerID
1	75123	8	...	10-4030-018759	18759

```
SELECT        Nested Loops       Index Seek (NonClustered)
Cost: 0 %     (Inner Join)       [Orders].[ix1]
              Cost: 0 %          Cost: 50 %
              0.000s             0.000s
              1 of               1 of
              1 (100%)           1 (100%)
```

```
Table 'Orders'. Scan count 1, logical reads 5,
```

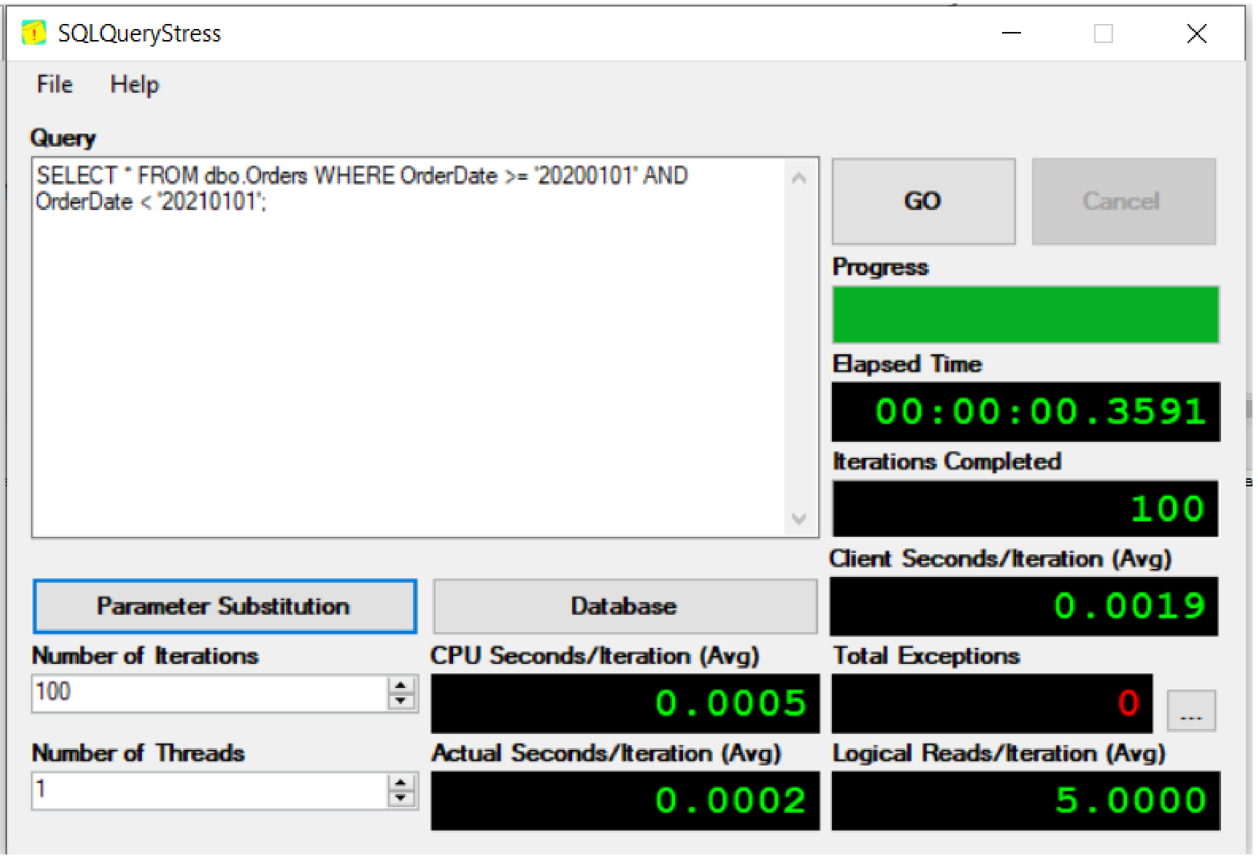
YEAR funkcija vs. >= & <



SQLQueryStress

```
SELECT * FROM dbo.Orders WHERE YEAR(OrderDate) = 2020;
```

Elapsed Time	00:00:02.9988
Iterations Completed	100
Client Seconds/Iteration (Avg)	0.0232
Number of Iterations	100
CPU Seconds/Iteration (Avg)	0.0049
Total Exceptions	0
Number of Threads	1
Actual Seconds/Iteration (Avg)	0.0191
Logical Reads/Iteration (Avg)	76.0000



SQLQueryStress

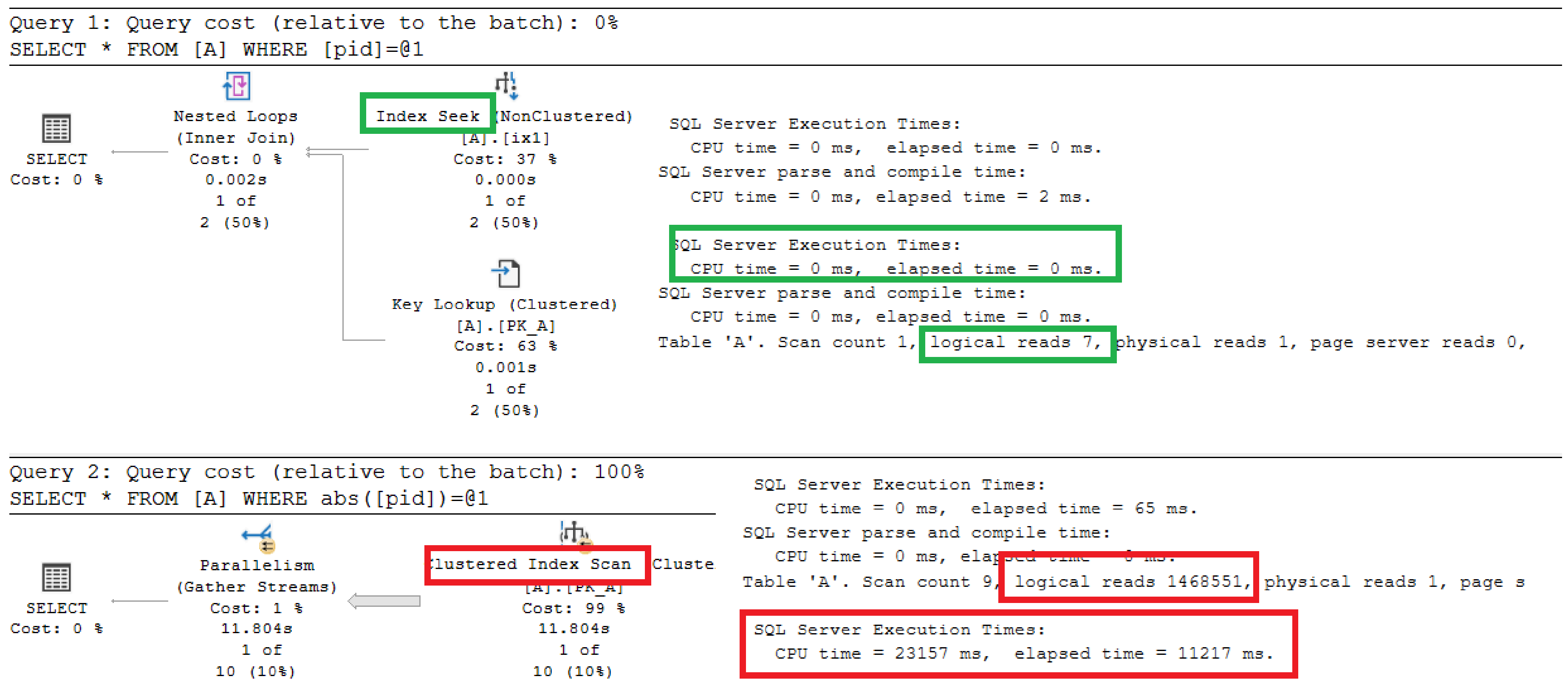
```
SELECT * FROM dbo.Orders WHERE OrderDate >= '20200101' AND
OrderDate < '20210101';
```

Elapsed Time	00:00:00.3591
Iterations Completed	100
Client Seconds/Iteration (Avg)	0.0019
Number of Iterations	100
CPU Seconds/Iteration (Avg)	0.0005
Total Exceptions	0
Number of Threads	1
Actual Seconds/Iteration (Avg)	0.0002
Logical Reads/Iteration (Avg)	5.0000

Velika tabela (100 miliona redova)

nije svejedno

```
USE Statistik;
SELECT * FROM A WHERE pid = 77765;
SELECT * FROM A WHERE ABS(pid) = 77765;
GO
```



```
Query 1: Query cost (relative to the batch): 0%
SELECT * FROM [A] WHERE [pid]=@1
```

```
SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 2 ms.

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.
Table 'A'. Scan count 1, logical reads 7, physical reads 1, page server reads 0,
```

```
Query 2: Query cost (relative to the batch): 100%
SELECT * FROM [A] WHERE abs([pid])=@1
```

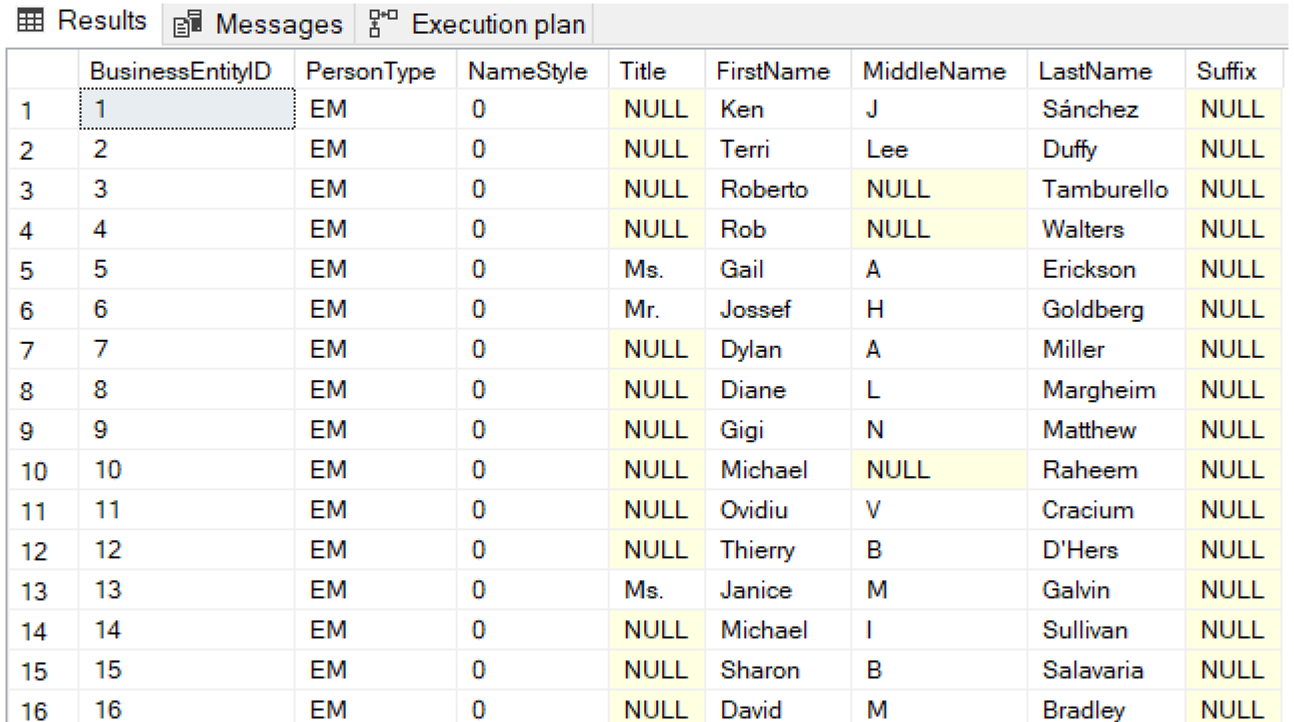
```
SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 65 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'A'. Scan count 9, logical reads 1468551, physical reads 1, page s

SQL Server Execution Times:
   CPU time = 23157 ms,  elapsed time = 11217 ms.
```

Nova testna tabela - Contacts

```
USE AdventureWorks2019;
DROP TABLE IF EXISTS dbo.Contacts;
GO
SELECT BusinessEntityID, PersonType, NameStyle, Title, FirstName, MiddleName, LastName,
Suffix INTO dbo.Contacts FROM Person.Person;

ALTER TABLE dbo.Contacts ADD CONSTRAINT PK_Contacts PRIMARY KEY (BusinessEntityID);
GO
```

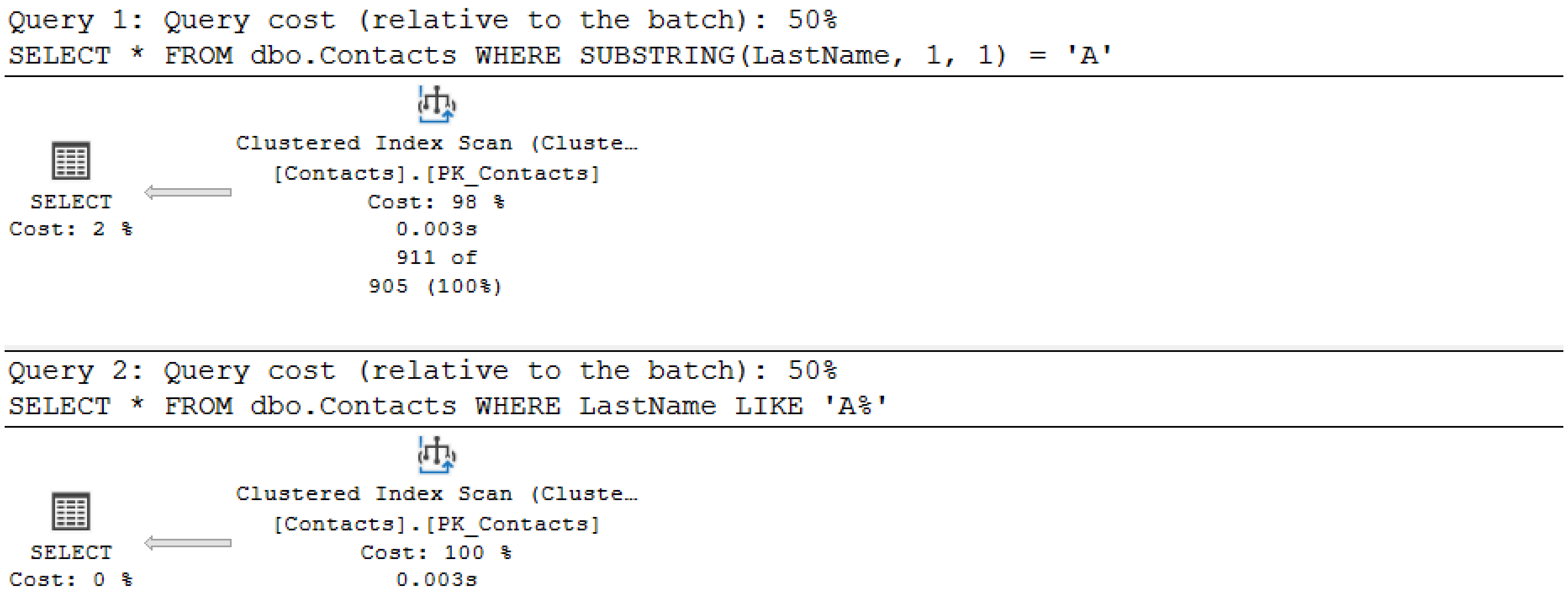


	BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName	Suffix
1	1	EM	0	NULL	Ken	J	Sánchez	NULL
2	2	EM	0	NULL	Terri	Lee	Duffy	NULL
3	3	EM	0	NULL	Roberto	NULL	Tamburello	NULL
4	4	EM	0	NULL	Rob	NULL	Walters	NULL
5	5	EM	0	Ms.	Gail	A	Erickson	NULL
6	6	EM	0	Mr.	Jossef	H	Goldberg	NULL
7	7	EM	0	NULL	Dylan	A	Miller	NULL
8	8	EM	0	NULL	Diane	L	Margheim	NULL
9	9	EM	0	NULL	Gigi	N	Matthew	NULL
10	10	EM	0	NULL	Michael	NULL	Raheem	NULL
11	11	EM	0	NULL	Ovidiu	V	Cracium	NULL
12	12	EM	0	NULL	Thierry	B	D'Hers	NULL
13	13	EM	0	Ms.	Janice	M	Galvin	NULL
14	14	EM	0	NULL	Michael	I	Sullivan	NULL
15	15	EM	0	NULL	Sharon	B	Salavaria	NULL
16	16	EM	0	NULL	David	M	Bradley	NULL

19 972 rows

SUBSTRING vs. LIKE

```
CREATE INDEX ix1 ON dbo.Contacts(LastName);
GO
SELECT * FROM dbo.Contacts WHERE SUBSTRING(LastName, 1, 1) = 'A';
SELECT * FROM dbo.Contacts WHERE LastName LIKE 'A%';
```

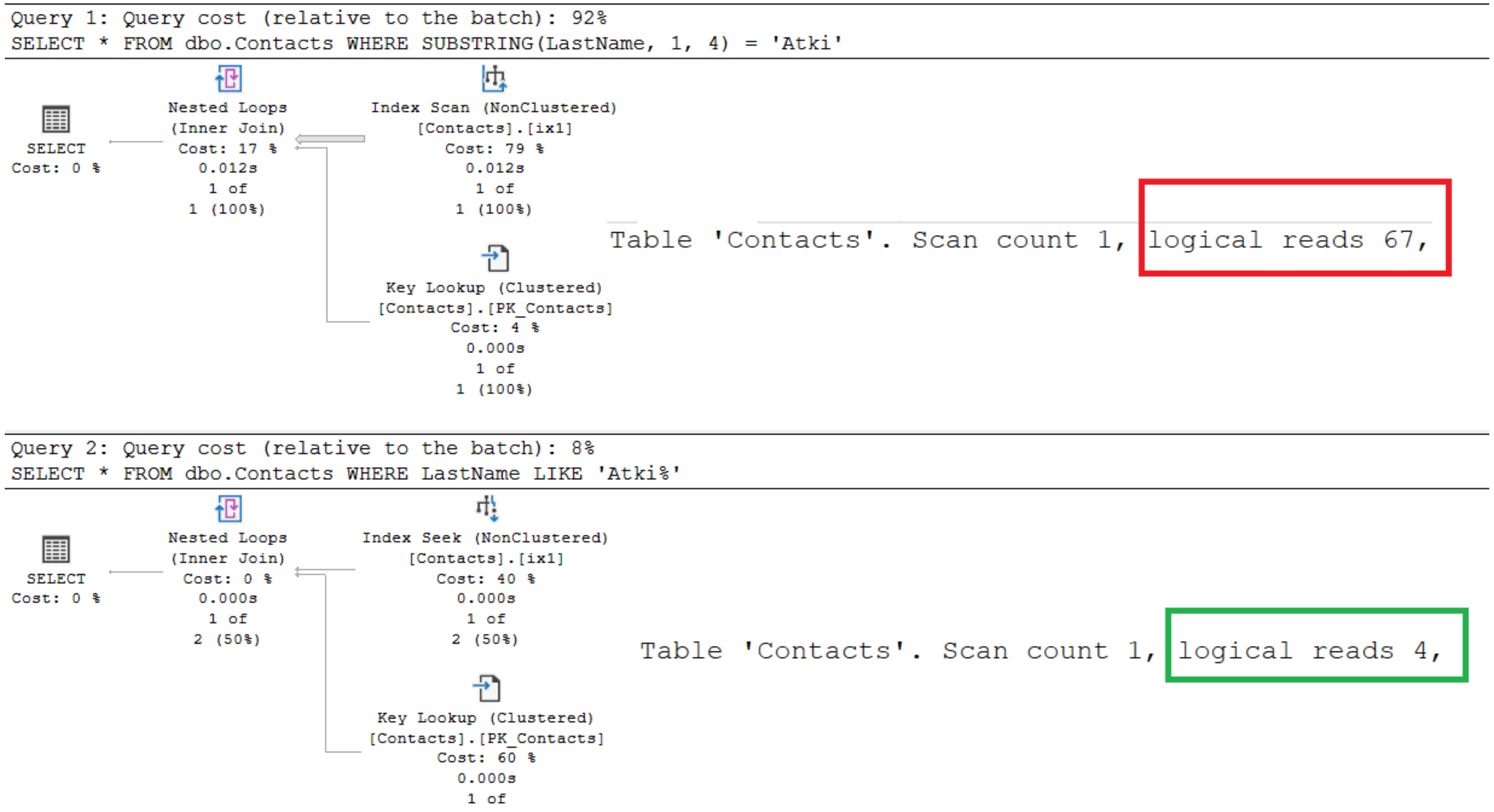


```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Contacts WHERE SUBSTRING(LastName, 1, 1) = 'A'
```

```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Contacts WHERE LastName LIKE 'A%'
```

SUBSTRING vs. LIKE

```
SELECT * FROM dbo.Contacts WHERE SUBSTRING(LastName, 1, 4) = 'Atki';
SELECT * FROM dbo.Contacts WHERE LastName LIKE 'Atki%';
```



```
Query 1: Query cost (relative to the batch): 92%
SELECT * FROM dbo.Contacts WHERE SUBSTRING(LastName, 1, 4) = 'Atki'
```

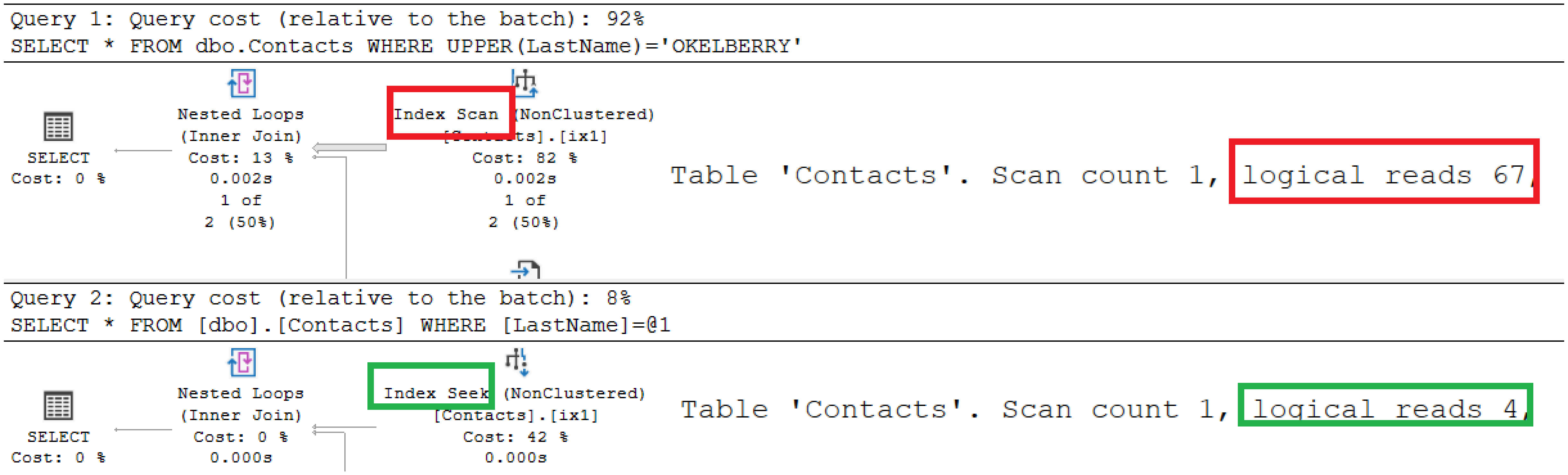
```
Table 'Contacts'. Scan count 1, logical reads 67,
```

```
Query 2: Query cost (relative to the batch): 8%
SELECT * FROM dbo.Contacts WHERE LastName LIKE 'Atki%'
```

```
Table 'Contacts'. Scan count 1, logical reads 4,
```

UPPER funkcija

```
SELECT * FROM dbo.Contacts WHERE UPPER(LastName)='OKELBERRY';
SELECT * FROM dbo.Contacts WHERE LastName ='oKELbERrY';
```



```
Query 1: Query cost (relative to the batch): 92%
SELECT * FROM dbo.Contacts WHERE UPPER(LastName)='OKELBERRY'
```

```
Table 'Contacts'. Scan count 1, logical reads 67,
```

```
Query 2: Query cost (relative to the batch): 8%
SELECT * FROM [dbo].[Contacts] WHERE [LastName]=@1
```

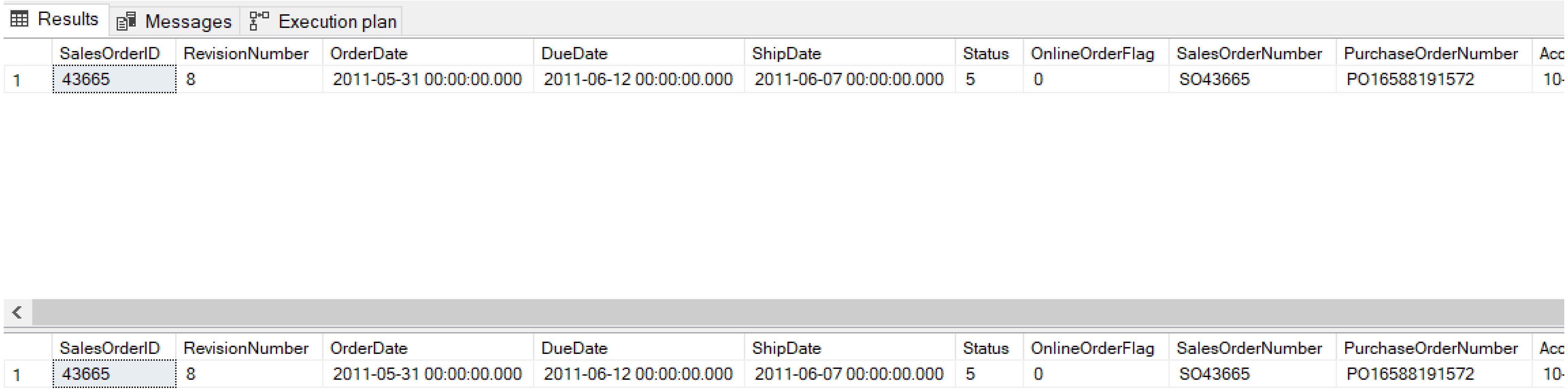
```
Table 'Contacts'. Scan count 1, logical reads 4,
```

Da rezimiramo

- Dva upita: jedan sa funkcijom u WHERE klauzuli, drugi bez
- Performanse:
  - Nema indeksa - ISTE
  - Ima indeksa, ali je upit neselektivan – ISTE
  - Ima indeksa i upit je dovoljno selektivan – Upit bez funkcije je brži
- Zato, kad god je moguće, izbegnite funkciju u WHERE klauzuli
- U slučaju velikih tabela razlika može da bude dramatična

Aritmetičke operacije

```sql
USE AdventureWorks2019;
DROP TABLE IF EXISTS dbo.Orders;
GO
SELECT * INTO dbo.Orders FROM Sales.SalesOrderHeader;
ALTER TABLE dbo.Orders ADD CONSTRAINT PK_Orders PRIMARY KEY (SalesOrderID);
GO
SELECT * FROM dbo.Orders WHERE SalesOrderID = 43665;
SELECT * FROM dbo.Orders WHERE SalesOrderID + 1 = 43666;
```

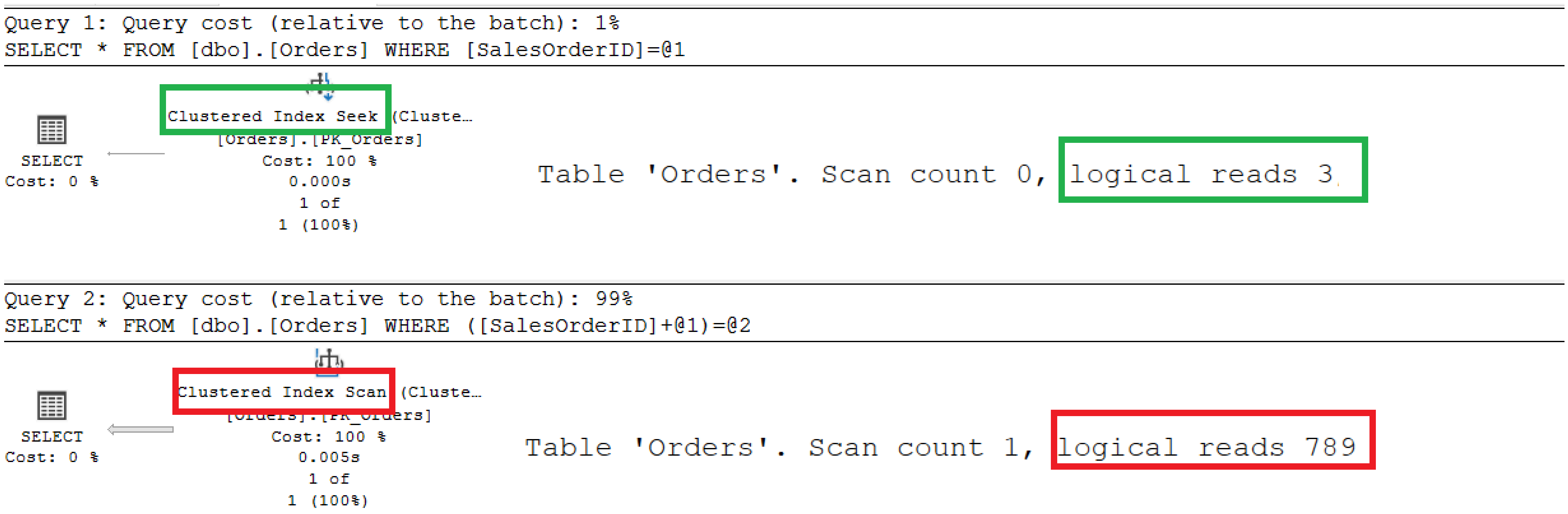


	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	Acc
1	43665	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43665	PO16588191572	10-

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	Acc
1	43665	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43665	PO16588191572	10-

Aritmetičke operacije

```
SELECT * FROM dbo.Orders WHERE SalesOrderID = 43665;
SELECT * FROM dbo.Orders WHERE SalesOrderID + 1 = 43666;
```



```
Query 1: Query cost (relative to the batch): 1%
SELECT * FROM [dbo].[Orders] WHERE [SalesOrderID]=@1
```

```
Table 'Orders'. Scan count 0, logical reads 3
```

```
Query 2: Query cost (relative to the batch): 99%
SELECT * FROM [dbo].[Orders] WHERE ([SalesOrderID]+@1)=@2
```

```
Table 'Orders'. Scan count 1, logical reads 789
```

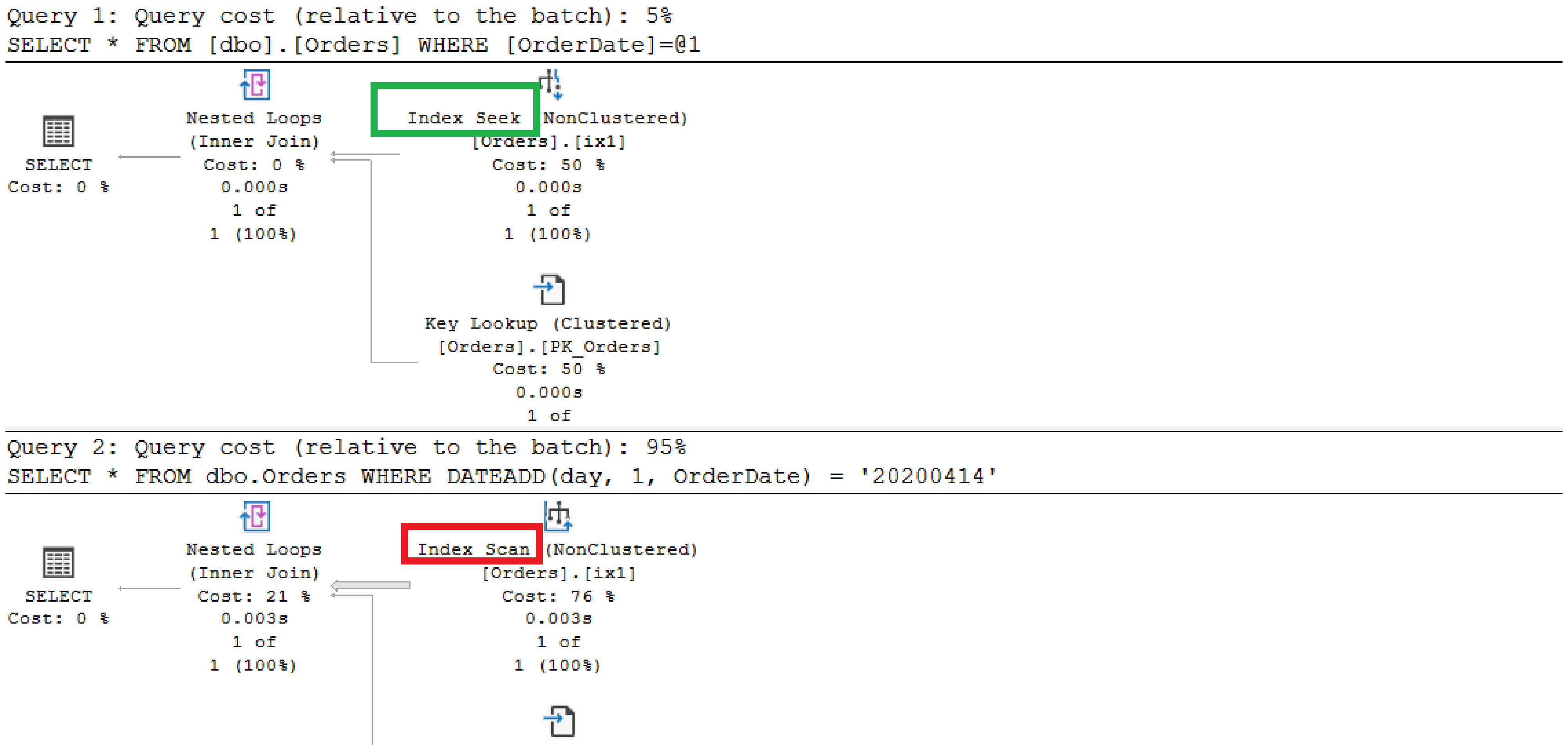
Zaključak

- Izbegavajte aritmetičke operacije u WHERE klauzuli sa kolonom kao operandom

JOŠ MALO DETALJA

Funkcija u WHERE klauzuli

```
SELECT * FROM dbo.Orders WHERE OrderDate = '20200413';
SELECT * FROM dbo.Orders WHERE DATEADD(day, 1, OrderDate) = '20200414';
```

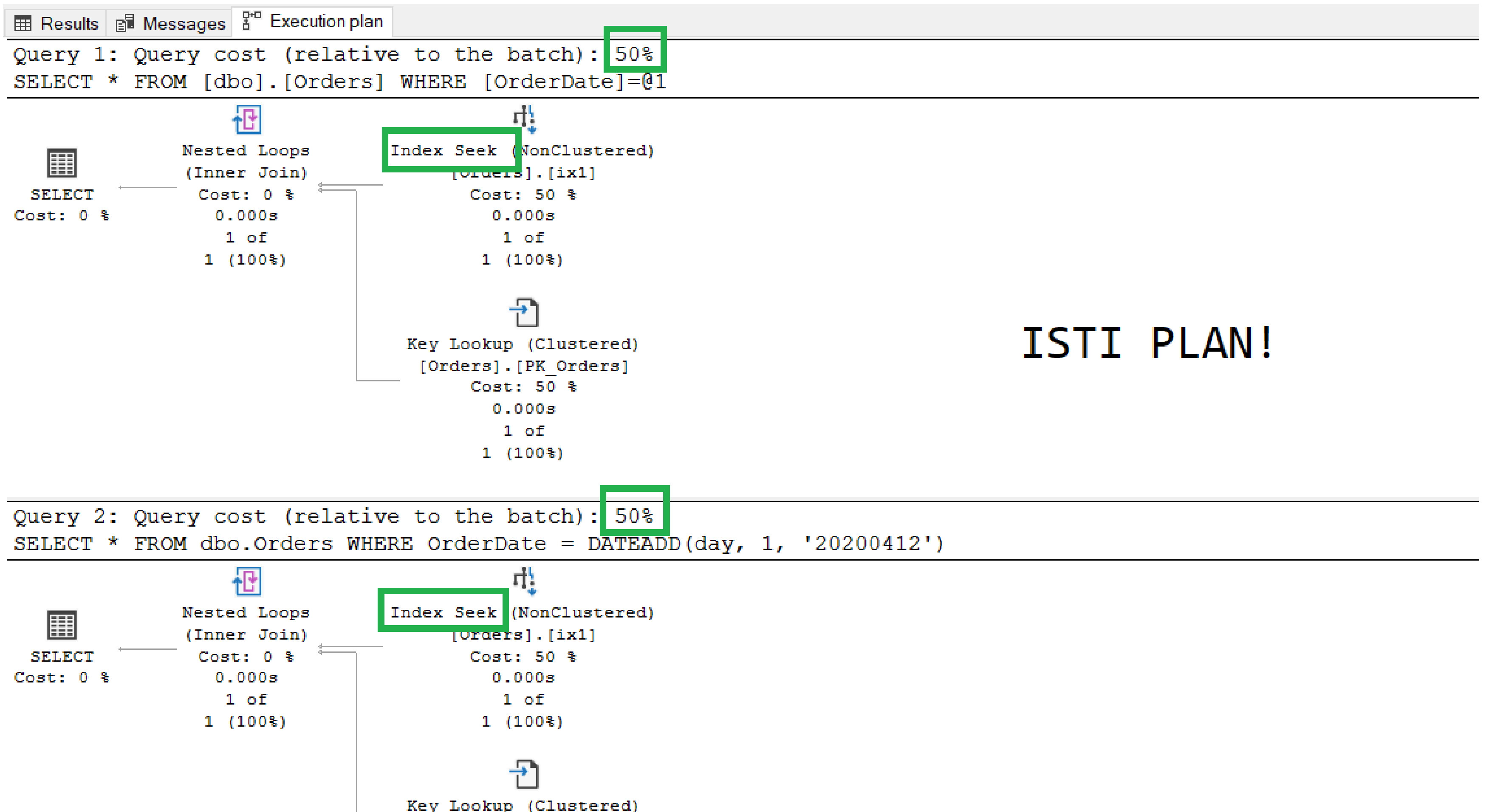


```
Query 1: Query cost (relative to the batch): 5%
SELECT * FROM [dbo].[Orders] WHERE [OrderDate]=@1
```

```
Query 2: Query cost (relative to the batch): 95%
SELECT * FROM dbo.Orders WHERE DATEADD(day, 1, OrderDate) = '20200414'
```

Funkcija u WHERE klauzuli

```sql
SELECT * FROM dbo.Orders WHERE OrderDate = '20200413';
SELECT * FROM dbo.Orders WHERE OrderDate = DATEADD(day, -1, '20200414');
```



```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM [dbo].[Orders] WHERE [OrderDate]=@1
```

```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM dbo.Orders WHERE OrderDate = DATEADD(day, 1, '20200412')
```

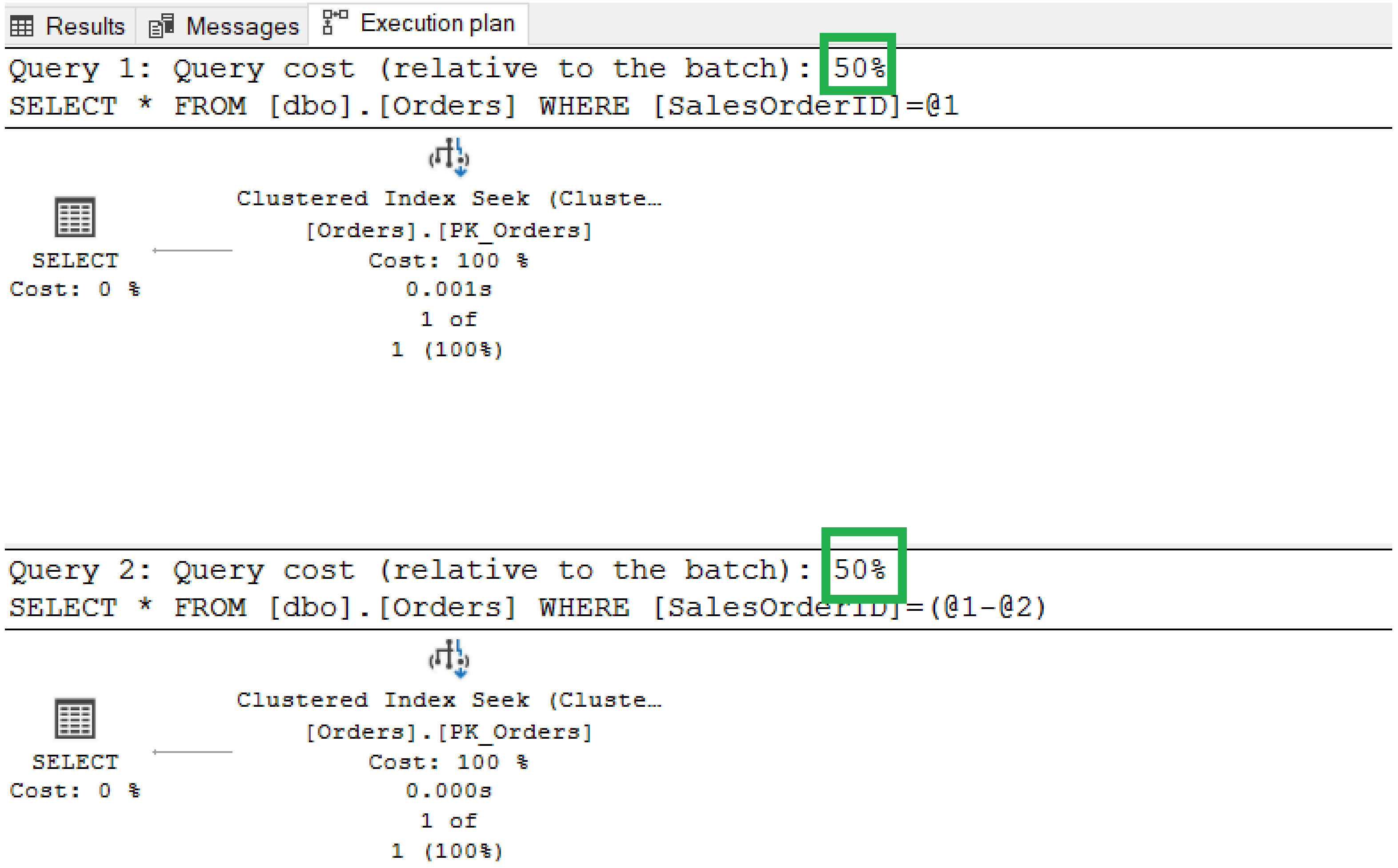
ISTI PLAN!

Kako isti plan kad imamo funkciju?

- Nije problem prisustvo funkcije u WHERE klauzuli, problem je kad je jedan or agumenata kolona tabele
- Funkcija se evaluira za sve vrednost kolone u svim redovima koji su se kvalifikovali u upitu

Aritmetičke operacije

```
SELECT * FROM dbo.Orders WHERE SalesOrderID = 43665;
SELECT * FROM dbo.Orders WHERE SalesOrderID = 43666 - 1;
```



```
Query 1: Query cost (relative to the batch): 50%
SELECT * FROM [dbo].[Orders] WHERE [SalesOrderID]=@1
```

SELECT
Cost: 0 %

Clustered Index Seek (Cluste…
[Orders].[PK_Orders]
Cost: 100 %
0.001s
1 of
1 (100%)

```
Query 2: Query cost (relative to the batch): 50%
SELECT * FROM [dbo].[Orders] WHERE [SalesOrderID]=(@1-@2)
```

SELECT
Cost: 0 %

Clustered Index Seek (Cluste…
[Orders].[PK_Orders]
Cost: 100 %
0.000s
1 of
1 (100%)

Šta je brže?

```
YEAR(OrderDate) = 2013
```

```
OrderDate >= '20130101' AND
OrderDate < '20140101';
```

Šta je brže?

```
SUBSTRING(LastName, 1, 4) = 'Atki';
```

```
LastName LIKE 'Atki%';
```

Šta je brže?

```
UPPER(LastName) = 'OKELBERRY';
```

```
LastName = 'OKELBERRY';
```

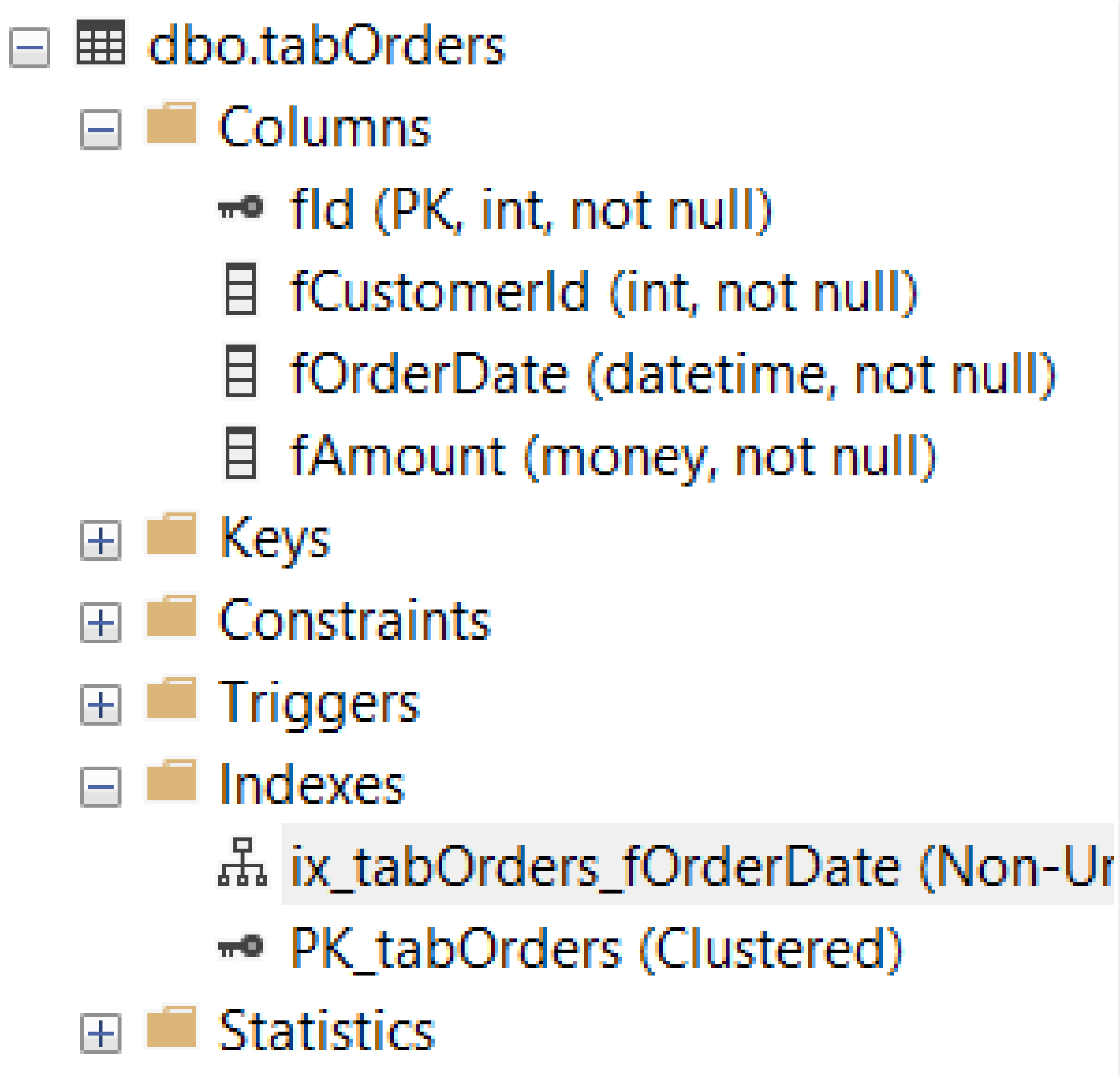
Šta je brže?

```
SalesOrderID + 1 = 43666;
```

```
SalesOrderID = 43665;
```

2 LOKALNE PROMENLJIVE U SQL SERVERU

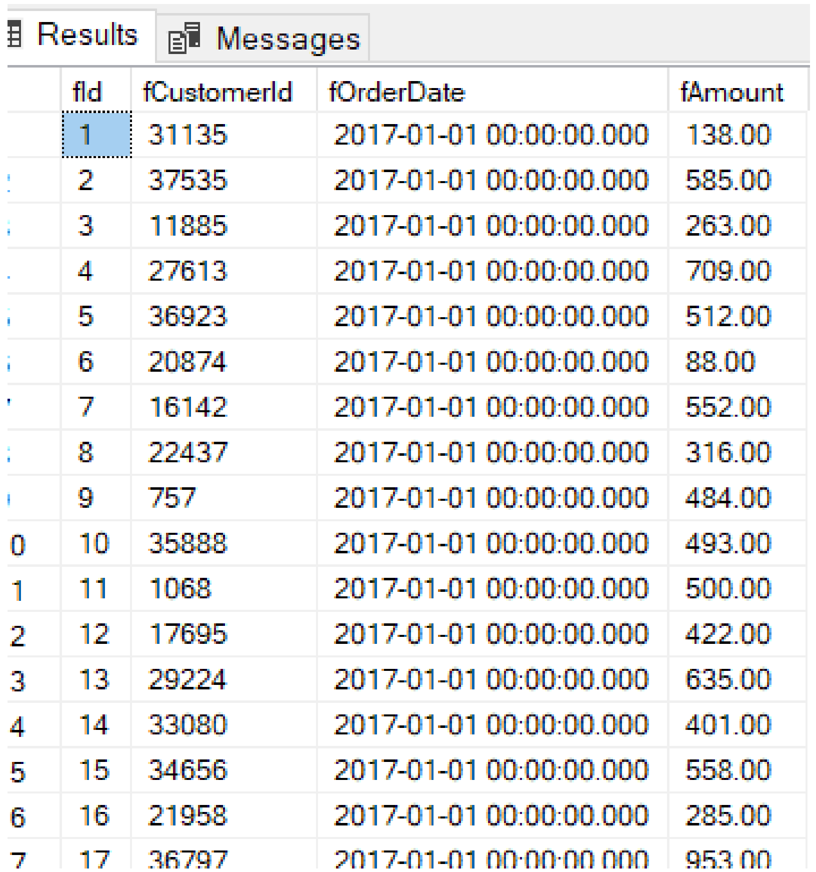
Testna tabela - Orders



dbo.tabOrders
- Columns
  - fId (PK, int, not null)
  - fCustomerId (int, not null)
  - fOrderDate (datetime, not null)
  - fAmount (money, not null)
- Keys
- Constraints
- Triggers
- Indexes
  - ix_tabOrders_fOrderDate (Non-Ur
  - PK_tabOrders (Clustered)
- Statistics

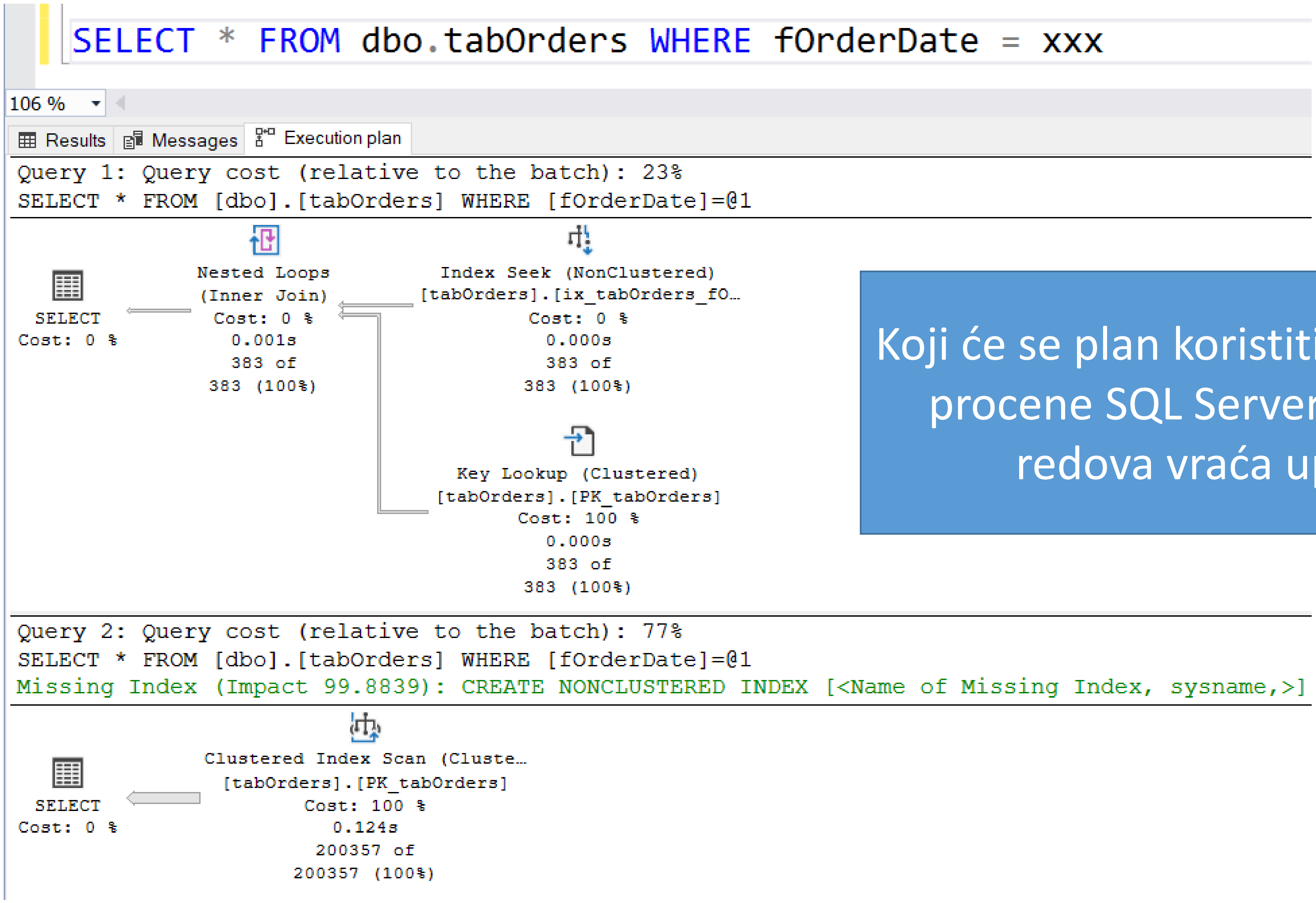
Milion redova

Indeks na koloni fOrderDate



Results | Messages

	fId	fCustomerId	fOrderDate	fAmount
	1	31135	2017-01-01 00:00:00.000	138.00
	2	37535	2017-01-01 00:00:00.000	585.00
	3	11885	2017-01-01 00:00:00.000	263.00
	4	27613	2017-01-01 00:00:00.000	709.00
	5	36923	2017-01-01 00:00:00.000	512.00
	6	20874	2017-01-01 00:00:00.000	88.00
	7	16142	2017-01-01 00:00:00.000	552.00
	8	22437	2017-01-01 00:00:00.000	316.00
	9	757	2017-01-01 00:00:00.000	484.00
0	10	35888	2017-01-01 00:00:00.000	493.00
1	11	1068	2017-01-01 00:00:00.000	500.00
2	12	17695	2017-01-01 00:00:00.000	422.00
3	13	29224	2017-01-01 00:00:00.000	635.00
4	14	33080	2017-01-01 00:00:00.000	401.00
5	15	34656	2017-01-01 00:00:00.000	558.00
6	16	21958	2017-01-01 00:00:00.000	285.00
7	17	36797	2017-01-01 00:00:00.000	953.00



```
SELECT * FROM dbo.tabOrders WHERE fOrderDate = xxx
```

106 %

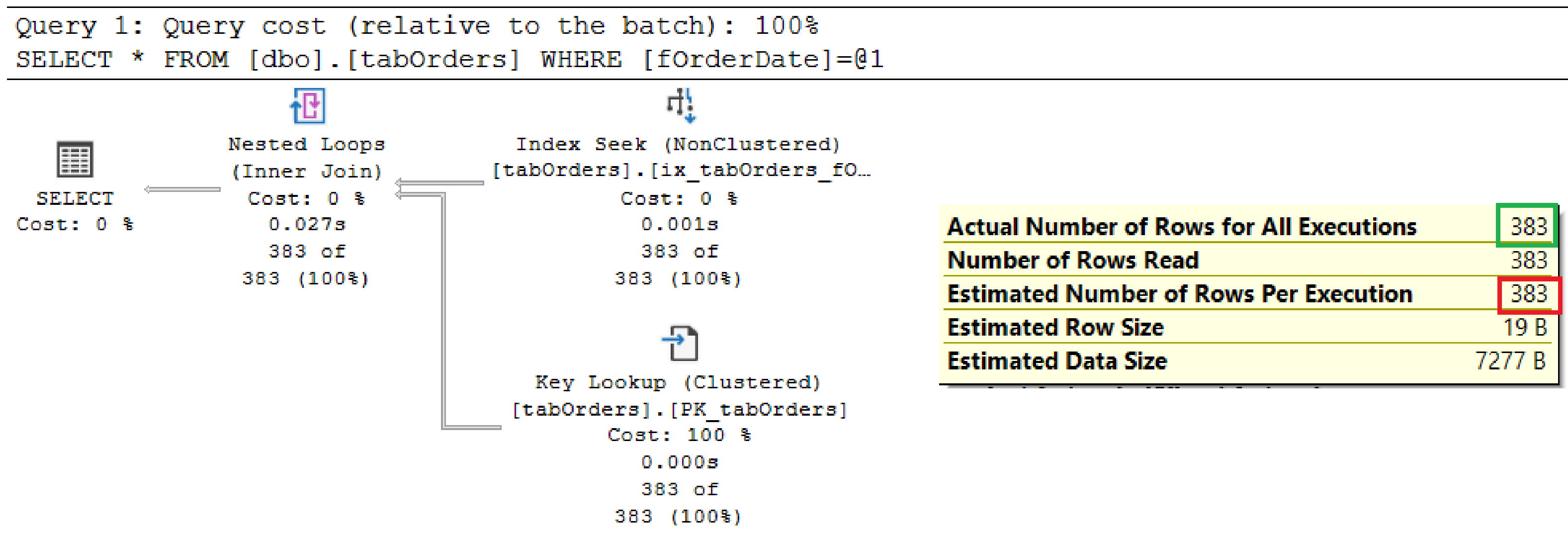
```
Query 1: Query cost (relative to the batch): 23%
SELECT * FROM [dbo].[tabOrders] WHERE [fOrderDate]=@1
```

```
Query 2: Query cost (relative to the batch): 77%
SELECT * FROM [dbo].[tabOrders] WHERE [fOrderDate]=@1
Missing Index (Impact 99.8839): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>]
```

Koji će se plan koristiti zavisi od procene SQL Server koliko redova vraća upit

Testni upiti

```sql
SELECT * FROM dbo.tabOrders WHERE fOrderDate = '20140330';
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[tabOrders] WHERE [fOrderDate]=@1
```

SELECT
Cost: 0 %

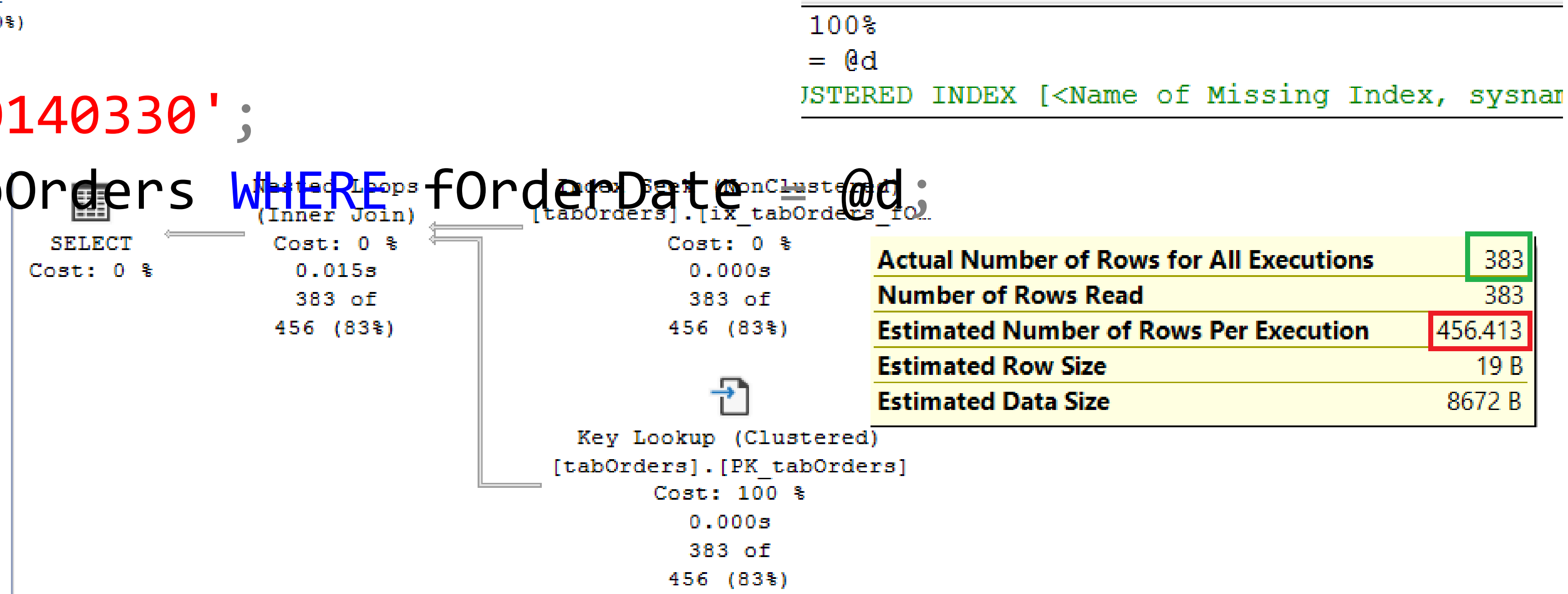
Nested Loops
(Inner Join)
Cost: 0 %
0.027s
383 of
383 (100%)

Index Seek (NonClustered)
[tabOrders].[ix_tabOrders_fO…
Cost: 0 %
0.001s
383 of
383 (100%)

Key Lookup (Clustered)
[tabOrders].[PK_tabOrders]
Cost: 100 %
0.000s
383 of
383 (100%)

Actual Number of Rows for All Executions	383
Number of Rows Read	383
Estimated Number of Rows Per Execution	383
Estimated Row Size	19 B
Estimated Data Size	7277 B

```sql
DECLARE @d DATE = '20140330';
SELECT * FROM dbo.tabOrders WHERE fOrderDate = @d;
```



```
100%
= @d
JSTERED INDEX [<Name of Missing Index, sysnar
```

SELECT
Cost: 0 %

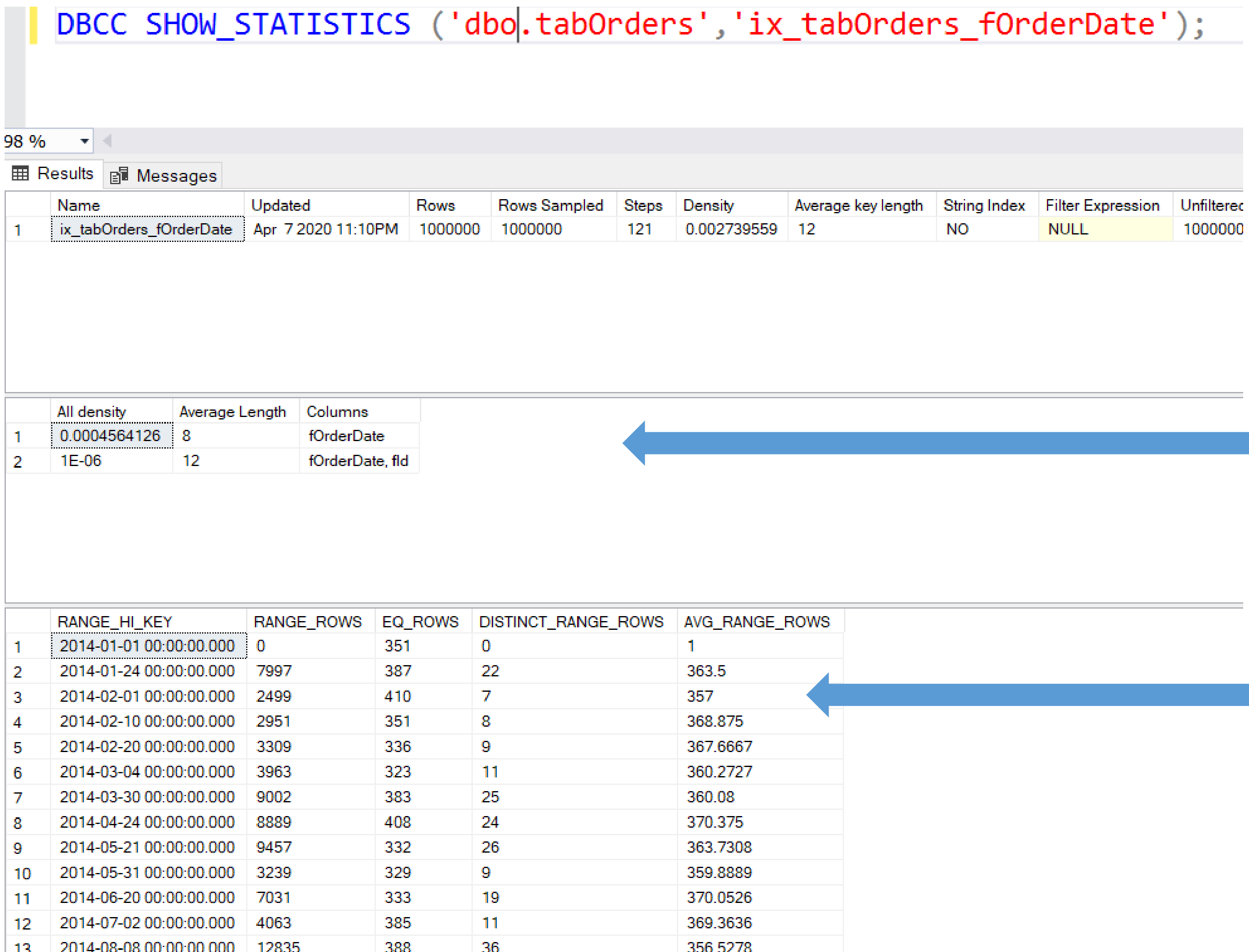
Nested Loops
(Inner Join)
Cost: 0 %
0.015s
383 of
456 (83%)

Index Seek (NonClustered)
[tabOrders].[ix_tabOrders_fO…
Cost: 0 %
0.000s
383 of
456 (83%)

Key Lookup (Clustered)
[tabOrders].[PK_tabOrders]
Cost: 100 %
0.000s
383 of
456 (83%)

Actual Number of Rows for All Executions	383
Number of Rows Read	383
Estimated Number of Rows Per Execution	456.413
Estimated Row Size	19 B
Estimated Data Size	8672 B

Objekat statistike



```
DBCC SHOW_STATISTICS ('dbo.tabOrders','ix_tabOrders_fOrderDate');
```

	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltere
1	ix_tabOrders_fOrderDate	Apr 7 2020 11:10PM	1000000	1000000	121	0.002739559	12	NO	NULL	1000000

Stats Header

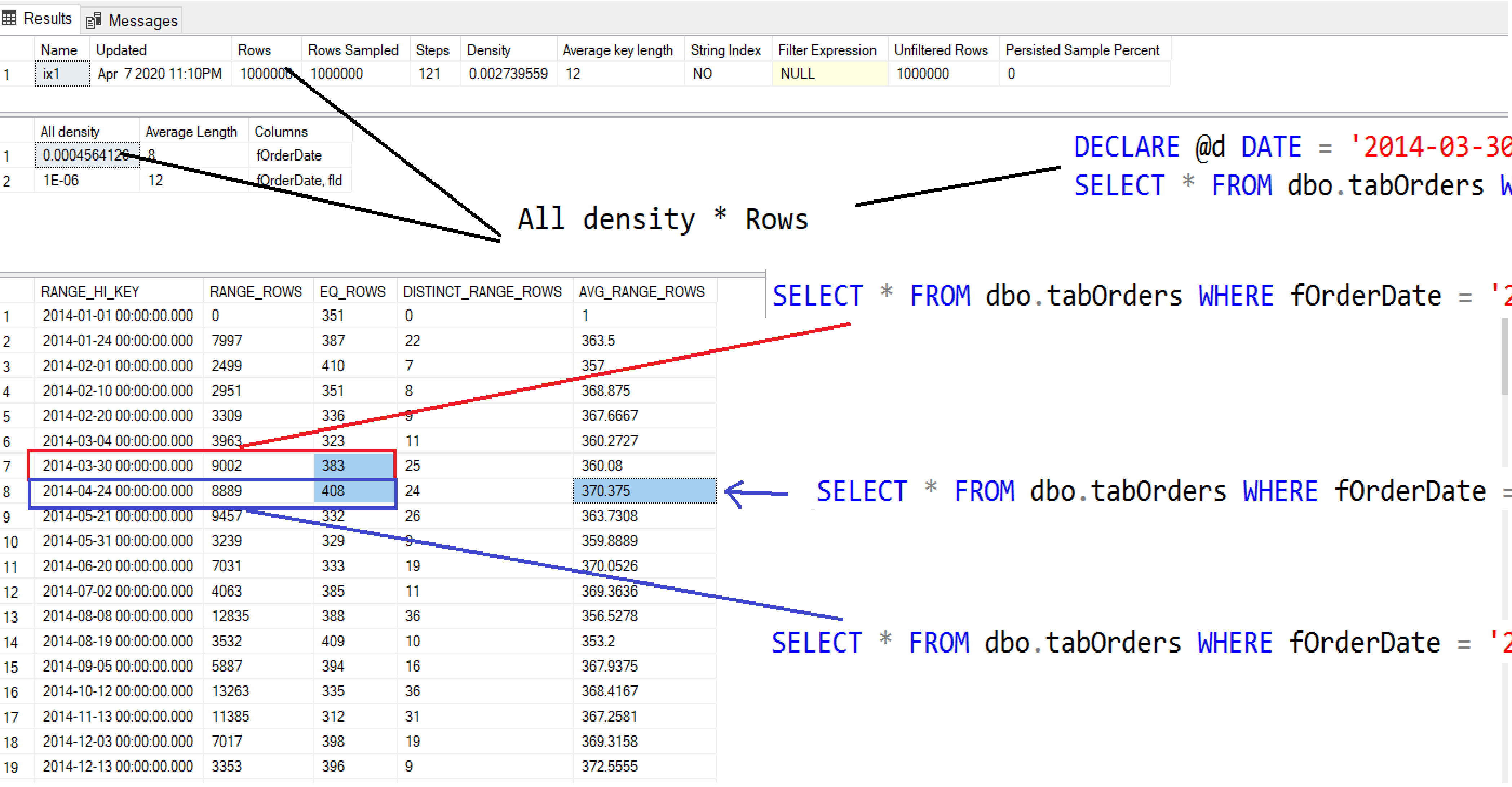
	All density	Average Length	Columns
1	0.0004564126	8	fOrderDate
2	1E-06	12	fOrderDate, fId

Density Vector

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	2014-01-01 00:00:00.000	0	351	0	1
2	2014-01-24 00:00:00.000	7997	387	22	363.5
3	2014-02-01 00:00:00.000	2499	410	7	357
4	2014-02-10 00:00:00.000	2951	351	8	368.875
5	2014-02-20 00:00:00.000	3309	336	9	367.6667
6	2014-03-04 00:00:00.000	3963	323	11	360.2727
7	2014-03-30 00:00:00.000	9002	383	25	360.08
8	2014-04-24 00:00:00.000	8889	408	24	370.375
9	2014-05-21 00:00:00.000	9457	332	26	363.7308
10	2014-05-31 00:00:00.000	3239	329	9	359.8889
11	2014-06-20 00:00:00.000	7031	333	19	370.0526
12	2014-07-02 00:00:00.000	4063	385	11	369.3636
13	2014-08-08 00:00:00.000	12835	388	36	356.5278

Stats Histogram

Objekat statistike



	Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows	Persisted Sample Percent
1	ix1	Apr 7 2020 11:10PM	1000000	1000000	121	0.002739559	12	NO	NULL	1000000	0

	All density	Average Length	Columns
1	0.0004564126	8	fOrderDate
2	1E-06	12	fOrderDate, fld

All density * Rows

```
DECLARE @d DATE = '2014-03-30'
SELECT * FROM dbo.tabOrders WHERE fOrderDate = @d
```

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	2014-01-01 00:00:00.000	0	351	0	1
2	2014-01-24 00:00:00.000	7997	387	22	363.5
3	2014-02-01 00:00:00.000	2499	410	7	357
4	2014-02-10 00:00:00.000	2951	351	8	368.875
5	2014-02-20 00:00:00.000	3309	336	9	367.6667
6	2014-03-04 00:00:00.000	3963	323	11	360.2727
7	2014-03-30 00:00:00.000	9002	383	25	360.08
8	2014-04-24 00:00:00.000	8889	408	24	370.375
9	2014-05-21 00:00:00.000	9457	332	26	363.7308
10	2014-05-31 00:00:00.000	3239	329	9	359.8889
11	2014-06-20 00:00:00.000	7031	333	19	370.0526
12	2014-07-02 00:00:00.000	4063	385	11	369.3636
13	2014-08-08 00:00:00.000	12835	388	36	356.5278
14	2014-08-19 00:00:00.000	3532	409	10	353.2
15	2014-09-05 00:00:00.000	5887	394	16	367.9375
16	2014-10-12 00:00:00.000	13263	335	36	368.4167
17	2014-11-13 00:00:00.000	11385	312	31	367.2581
18	2014-12-03 00:00:00.000	7017	398	19	369.3158
19	2014-12-13 00:00:00.000	3353	396	9	372.5555

```
SELECT * FROM dbo.tabOrders WHERE fOrderDate = '2014-03-30'
```

```
SELECT * FROM dbo.tabOrders WHERE fOrderDate = '2014-04-15'
```

```
SELECT * FROM dbo.tabOrders WHERE fOrderDate = '2014-04-24'
```

```sql
DECLARE @d DATE = '20191231';
SELECT * FROM dbo.tabOrders WHERE fOrderDate >= @d;
```

Estimated Number of rows = 30% redova iz tabele

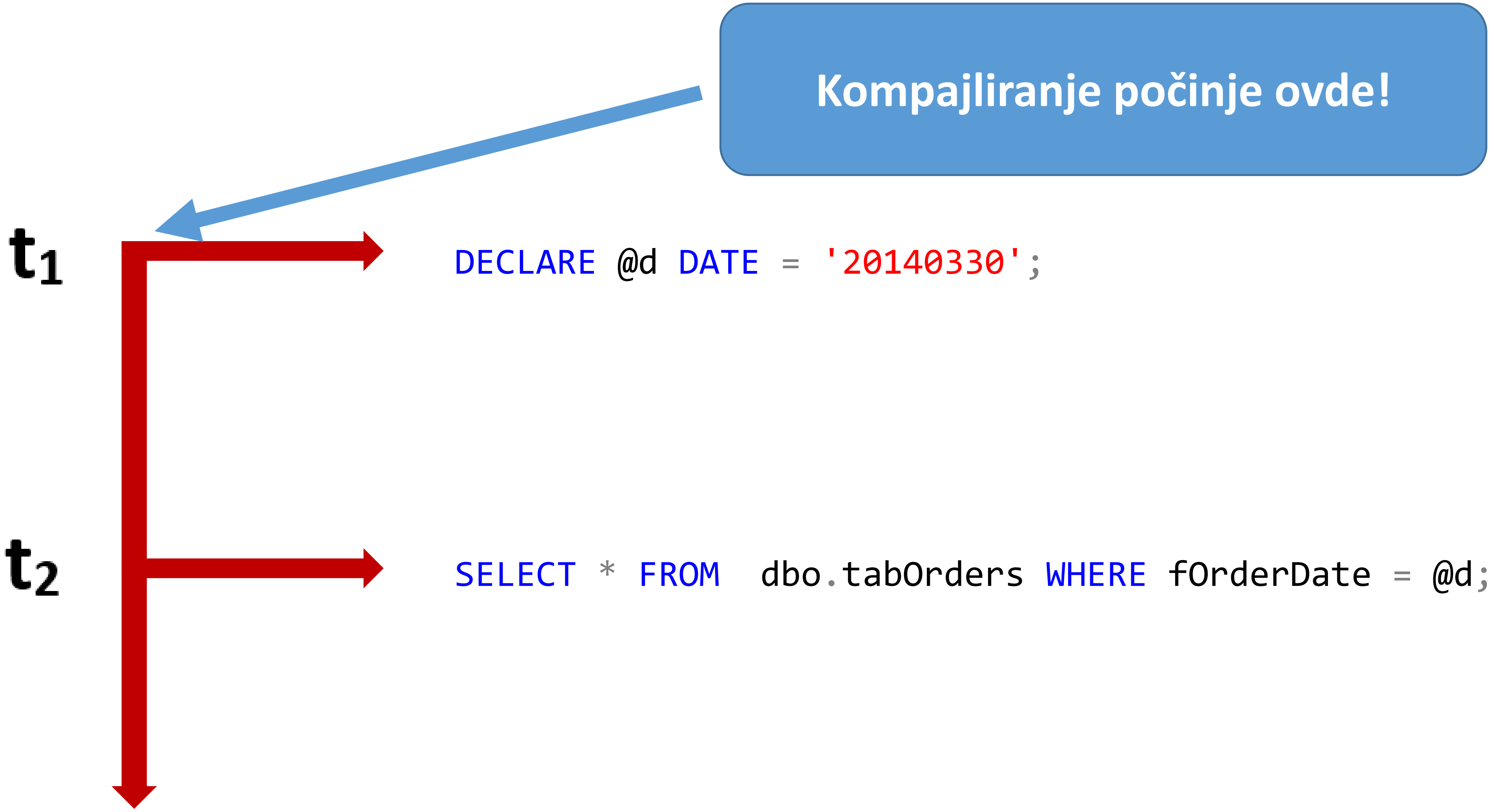
```sql
SELECT * FROM dbo.tabOrders WHERE fOrderDate BETWEEN
@d1 AND @d2;
```

Estimated Number of rows =

Pre SQL Servera 2014: 0.3*0.3 = 0.09 - 9%

Od SQL Servera 2014: 0.3*SQRT(0.3) = 0.1643 – 16,43%

Kompajliranje plana izvršenja

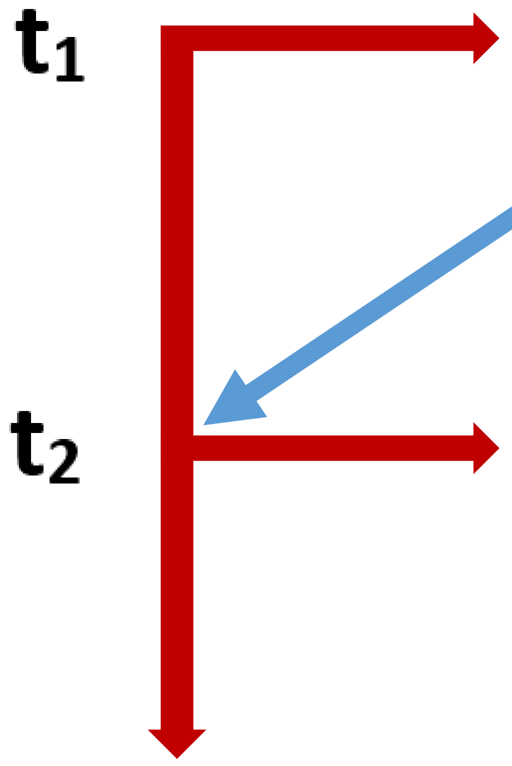


```
DECLARE @d DATE = '20140330';
```

```
SELECT * FROM  dbo.tabOrders WHERE fOrderDate = @d;
```

OPTION (RECOMPILE)

Kompajliranje počinje ovde!



```
DECLARE @d DATE = '20140330';

SELECT * FROM  dbo.tabOrders WHERE fOrderDate = @d
OPTION (RECOMPILE);
```

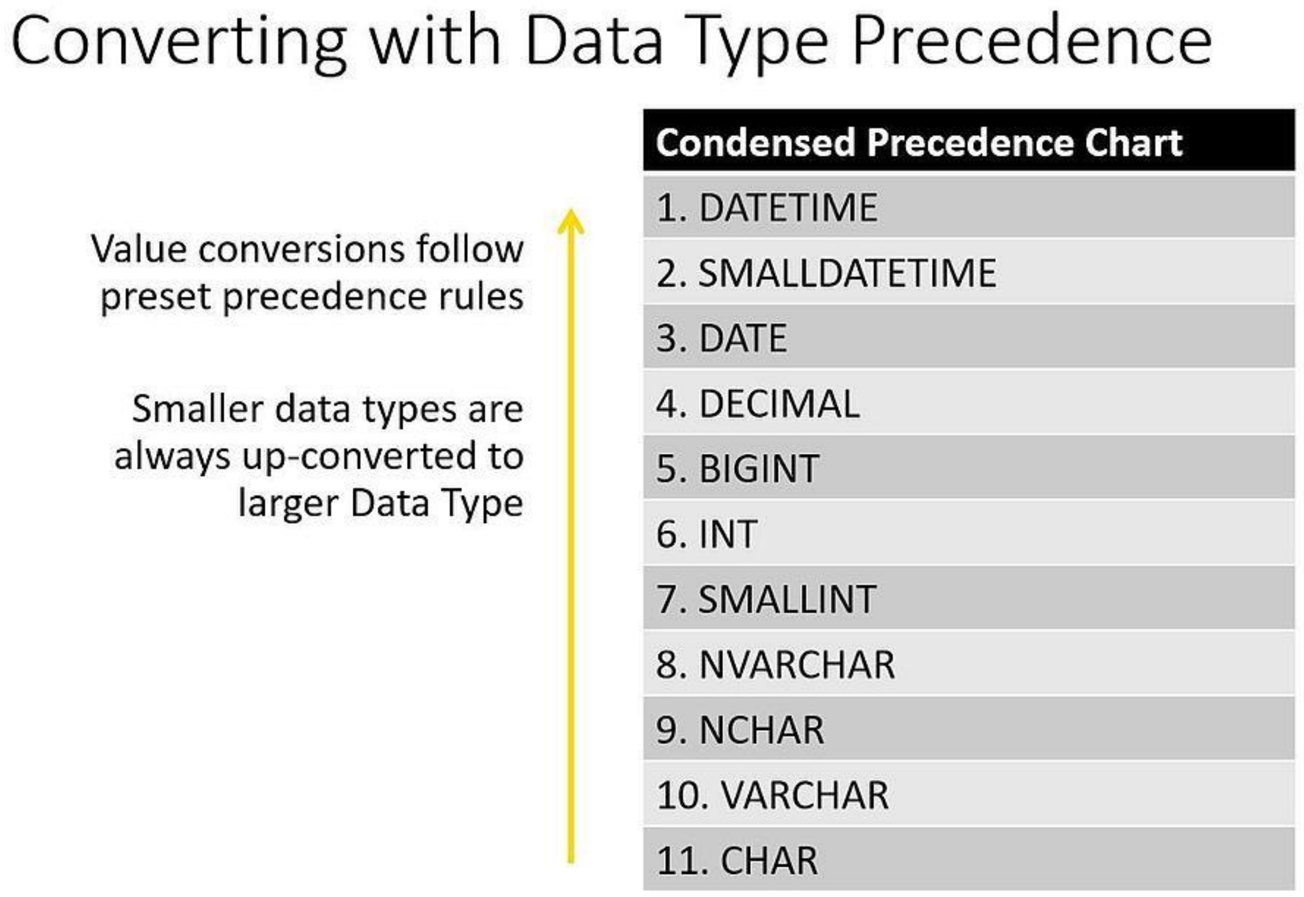
Zaključak

- Treba da razumete kako lokalne promenljive mogu da utiču na plan izvršenja
- U nekim slučajevima lokalne promenljive mogu drastično da pogoršaju performanse sistema
  - Operator = i neuniformna distribucija podataka
  - Tzv. range operatori (>, <, >=, <=, BETWEEN)
- Lokalne promenljive se ponašaju drugačije od SP parametara
- Ne zloupotrebljavajte i ne preterujte sa OPTION (RECOMPILE)
  - Ako se upit poziva 100x u sekundi, penali koje donosiOPTION (RECOMPILE) veći su od koristi!

3 KONVERZIJA TIPA PODATAKA

Implicitna konverzija

- Non-Unicode to Unicode
- Manji tip podatka u veći



Converting with Data Type Precedence

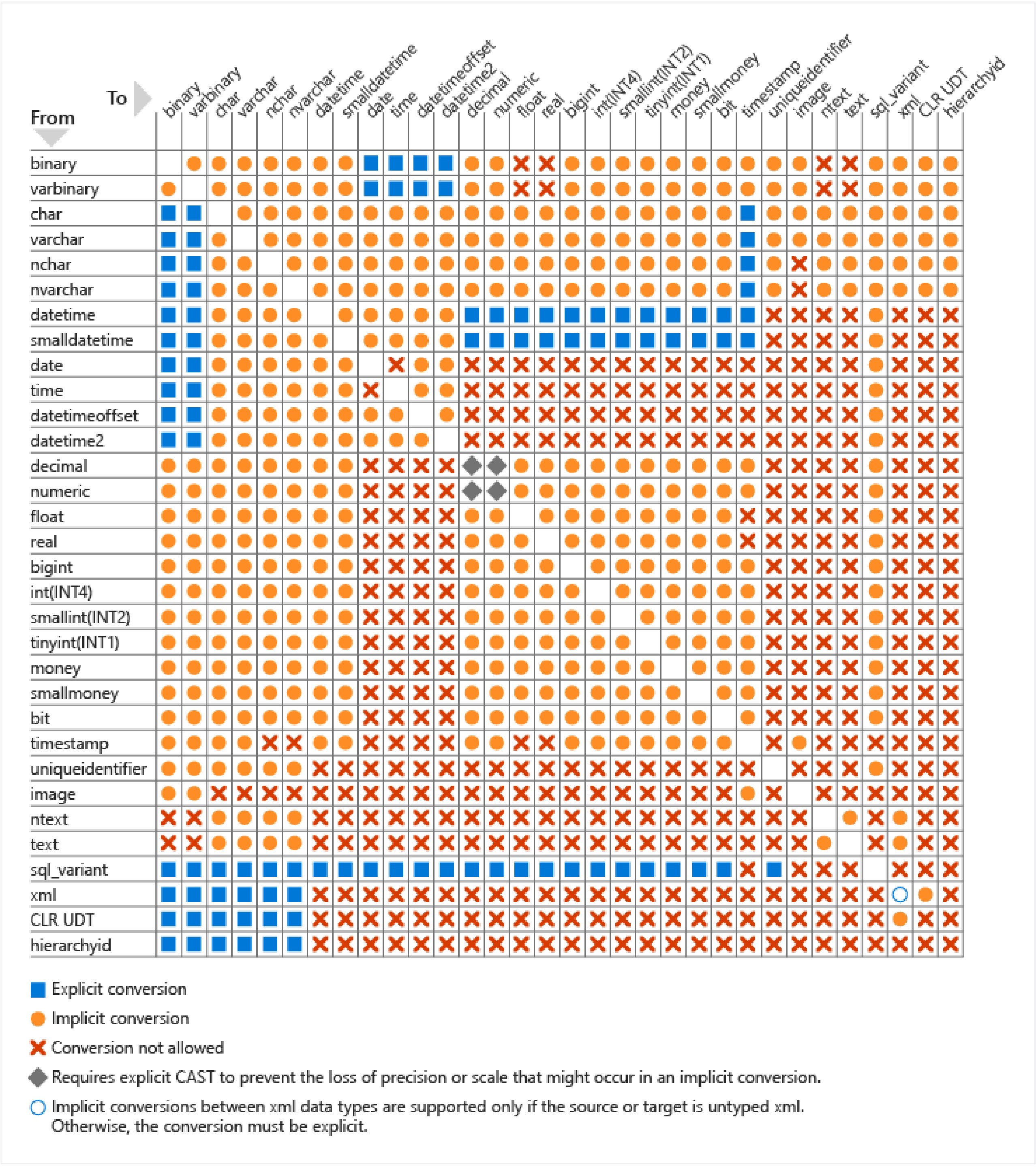
Value conversions follow preset precedence rules

Smaller data types are always up-converted to larger Data Type

Condensed Precedence Chart
1. DATETIME
2. SMALLDATETIME
3. DATE
4. DECIMAL
5. BIGINT
6. INT
7. SMALLINT
8. NVARCHAR
9. NCHAR
10. VARCHAR
11. CHAR

Moguće konverzije

- **Data type conversion (Database Engine)**
- https://bit.ly/3ciY9Ai



From / To	binary	varbinary	char	varchar	nchar	nvarchar	datetime	smalldatetime	date	time	datetimeoffset	datetime2	decimal	numeric	float	real	bigint	int(INT4)	smallint(INT2)	tinyint(INT1)	money	smallmoney	bit	timestamp	uniqueidentifier	image	ntext	text	sql_variant	xml	CLR UDT	hierarchyid
binary		●	●	●	●	●	●	●	■	■	■	■	●	●	✗	✗	●	●	●	●	●	●	●	●	●	●	✗	✗	●	●	●	●
varbinary	●		●	●	●	●	●	●	■	■	■	■	●	●	✗	✗	●	●	●	●	●	●	●	●	●	●	✗	✗	●	●	●	●
char	■	■		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	■	●	●	●	●	●	●	●	●
varchar	■	■	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	■	●	●	●	●	●	●	●	●
nchar	■	■	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	■	●	✗	●	●	●	●	●	●
nvarchar	■	■	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	■	●	✗	●	●	●	●	●	●
datetime	■	■	●	●	●	●		●	●	●	●	●	■	■	■	■	■	■	■	■	■	■	■	■	✗	✗	✗	✗	●	✗	✗	✗
smalldatetime	■	■	●	●	●	●	●		●	●	●	●	■	■	■	■	■	■	■	■	■	■	■	■	✗	✗	✗	✗	●	✗	✗	✗
date	■	■	●	●	●	●	●	●		✗	●	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	●	✗	✗	✗
time	■	■	●	●	●	●	●	●	✗		●	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	●	✗	✗	✗
datetimeoffset	■	■	●	●	●	●	●	●	●	●		●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	●	✗	✗	✗
datetime2	■	■	●	●	●	●	●	●	●	●	●		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	●	✗	✗	✗
decimal	●	●	●	●	●	●	●	●	✗	✗	✗	✗	◆	◆	●	●	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	✗	✗	✗
numeric	●	●	●	●	●	●	●	●	✗	✗	✗	✗	◆	◆	●	●	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	✗	✗	✗
float	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●		●	●	●	●	●	●	●	●	✗	✗	✗	✗	✗	●	✗	✗	✗
real	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●		●	●	●	●	●	●	●	✗	✗	✗	✗	✗	●	✗	✗	✗
bigint	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●	●		●	●	●	●	●	●	●	✗	✗	✗	✗	●	✗	✗	✗
int(INT4)	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●	●	●		●	●	●	●	●	●	✗	✗	✗	✗	●	✗	✗	✗
smallint(INT2)	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●	●	●	●		●	●	●	●	●	✗	✗	✗	✗	●	✗	✗	✗
tinyint(INT1)	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●	●	●	●	●		●	●	●	●	✗	✗	✗	✗	●	✗	✗	✗
money	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●	●	●	●	●	●		●	●	●	✗	✗	✗	✗	●	✗	✗	✗
smallmoney	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●	●	●	●	●	●	●		●	●	✗	✗	✗	✗	●	✗	✗	✗
bit	●	●	●	●	●	●	●	●	✗	✗	✗	✗	●	●	●	●	●	●	●	●	●	●		●	✗	✗	✗	✗	●	✗	✗	✗
timestamp	●	●	●	●	✗	✗	●	●	✗	✗	✗	✗	●	●	✗	✗	●	●	●	●	●	●	●		✗	●	✗	✗	✗	✗	✗	✗
uniqueidentifier	●	●	●	●	●	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		✗	✗	✗	●	✗	✗	✗
image	●	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	●	✗		✗	✗	✗	✗	✗	✗
ntext	✗	✗	●	●	●	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		●	✗	●	✗	✗
text	✗	✗	●	●	●	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	●		✗	●	✗	✗
sql_variant	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	✗	■	✗	✗	✗		✗	✗	✗
xml	■	■	■	■	■	■	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	○	●	✗
CLR UDT	■	■	■	■	■	■	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	●	✗	✗
hierarchyid	■	■	■	■	■	■	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

■ Explicit conversion

● Implicit conversion

✗ Conversion not allowed

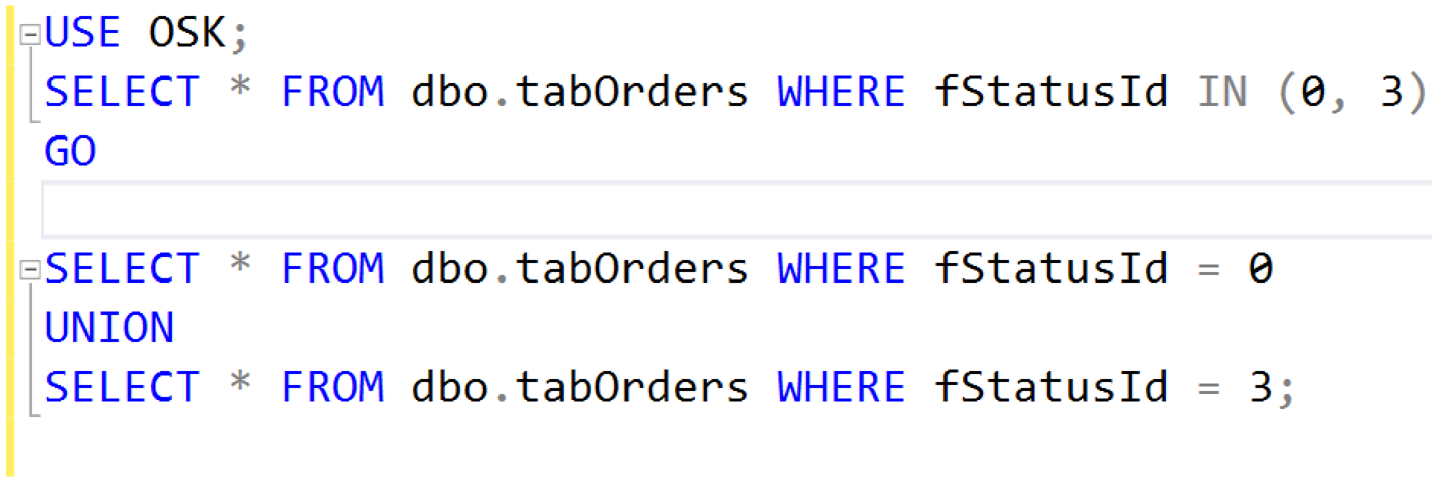
◆ Requires explicit CAST to prevent the loss of precision or scale that might occur in an implicit conversion.

○ Implicit conversions between xml data types are supported only if the source or target is untyped xml. Otherwise, the conversion must be explicit.

4 PODMUKLI OR STEJTMENT

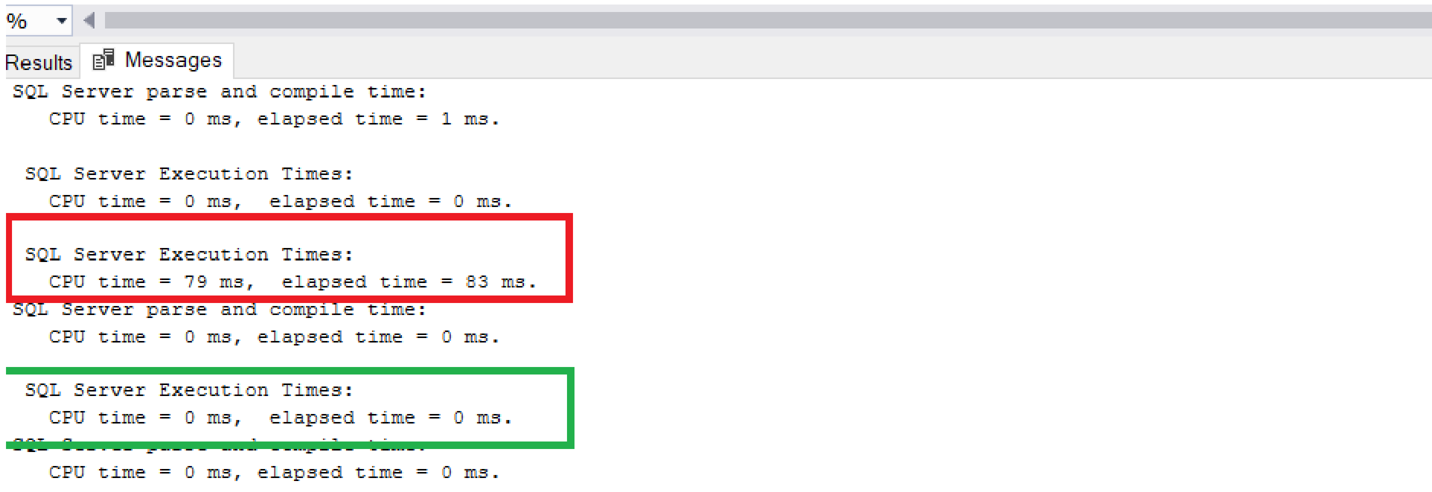
OR => UNION

- Ako je upit sa OR spor probajte da ga napišete pomoću UNION stejtmenta



```sql
USE OSK;
SELECT * FROM dbo.tabOrders WHERE fStatusId IN (0, 3)
GO

SELECT * FROM dbo.tabOrders WHERE fStatusId = 0
UNION
SELECT * FROM dbo.tabOrders WHERE fStatusId = 3;
```



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

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

 SQL Server Execution Times:
   CPU time = 79 ms,  elapsed time = 83 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 parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
```

5 KORISNIČKE FUNKCIJE I PERFORMANSE

Funkcije (UDF) u SQL Serveru

- Code reuse, encapsulation and modularity
- Complex business rules or computations
- Single place change
- Written once, invoke from many modules
- Reduce network traffic

Funkcije (UDF) u SQL Serveru

- Samo SELECT, ne može da se menja stanje baze funkcijama
- Ne može da se koristi Dynamic SQL
- Tipovi funkcija
  - Skalarne
  - Linijske (inline table-valued functions)
  - MSTVF (multi-statement table-valued functions)

Skalarne funkcije u SQL Serveru

Why do SQL Server Scalar-valued functions get slower?

Refactor SQL Server scalar UDF to inline TVF to improve performance

Why SQL Server scalar functions are bad?

T-SQL Best Practices - Don't Use Scalar Value Functions in Column .

Are SQL Server Functions Dragging Your Query Down?

SQL functions rarely perform well.

Skalarne funkcije u SQL Serveru

- Skalarne funkcije mogu da budu veoma spore
  - Iterativno pozivanje
  - Ekstra troškovi prilikom svakog pojedinačnog pozivanja
- Nemoguća optimizacija pozivajućeg upita
- Samo serijski planovi su mogući

Preporuka u vezi sa funkcijama (UDF)

- < SQL Server 2019
- Samo INLINE funkcije i bleya

- SQL Server 2019
- Preporuka je i dalje da koristite inline funkcije, ali je zbog fičera Scalar UDF Inlining dolaze u obzir i skalarane funkcije
  - Ne radi za sve skalarne funkcije
  - Novi fičer, pa su mogući bagovi

- Stoga, da biste bili na sigurnoj strani – birajte INLINE funkciju

6 DATABASE CONSTRAINTS AND PERFORMANCE

Database Constraints i performanse

- Tip: Konstrejnti pomažu SQL Serveru da napravi bolji plan

- Glavna svrha ograničenja (constraints) je integritet podataka, ali svi oni (Unique Constraints, Check Constraints i Foreign Keys) doprinose boljim performansama, tako da imate dva veoma bitna razloga da ih koristite

SUMMARY

- Izbegavajte funkcije i aritmetičke operacije u WHERE klauzuli kada je neka kolona argument funkcije odnosno jedan od operanada
- Budite obazrivi kada koristite lokalne promenljive, usporavaju sistem
  - Kad je neravnom. distribucija ili se koriste operatori nejednakosti
  - može da se ispegla sa OPTION(RECOMPILE), ali ne preterujte, to nije besplatno
- Upodobite tip podataka parametra ili promenljive sa tipom podataka kolone kako biste izbegli penale prilikom konverzije
- Upiti sa OR mogu da se ubrzaju ako se napišu pomoću operatora UNION
- Koristite inlajn funkcije umesto skalarnih ili MSTVF
- CHECK i UNIQUE konstrejnti pobljšavaju performanse sistema