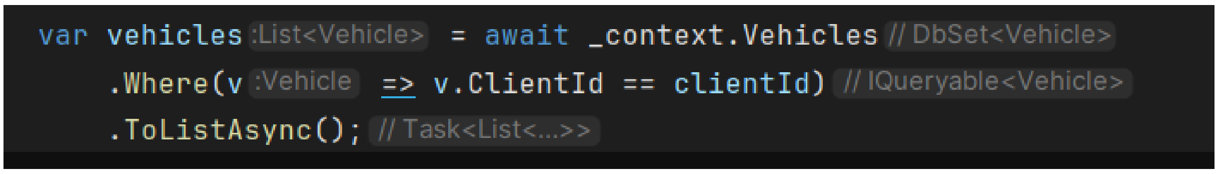
WorkshopManager 2.0

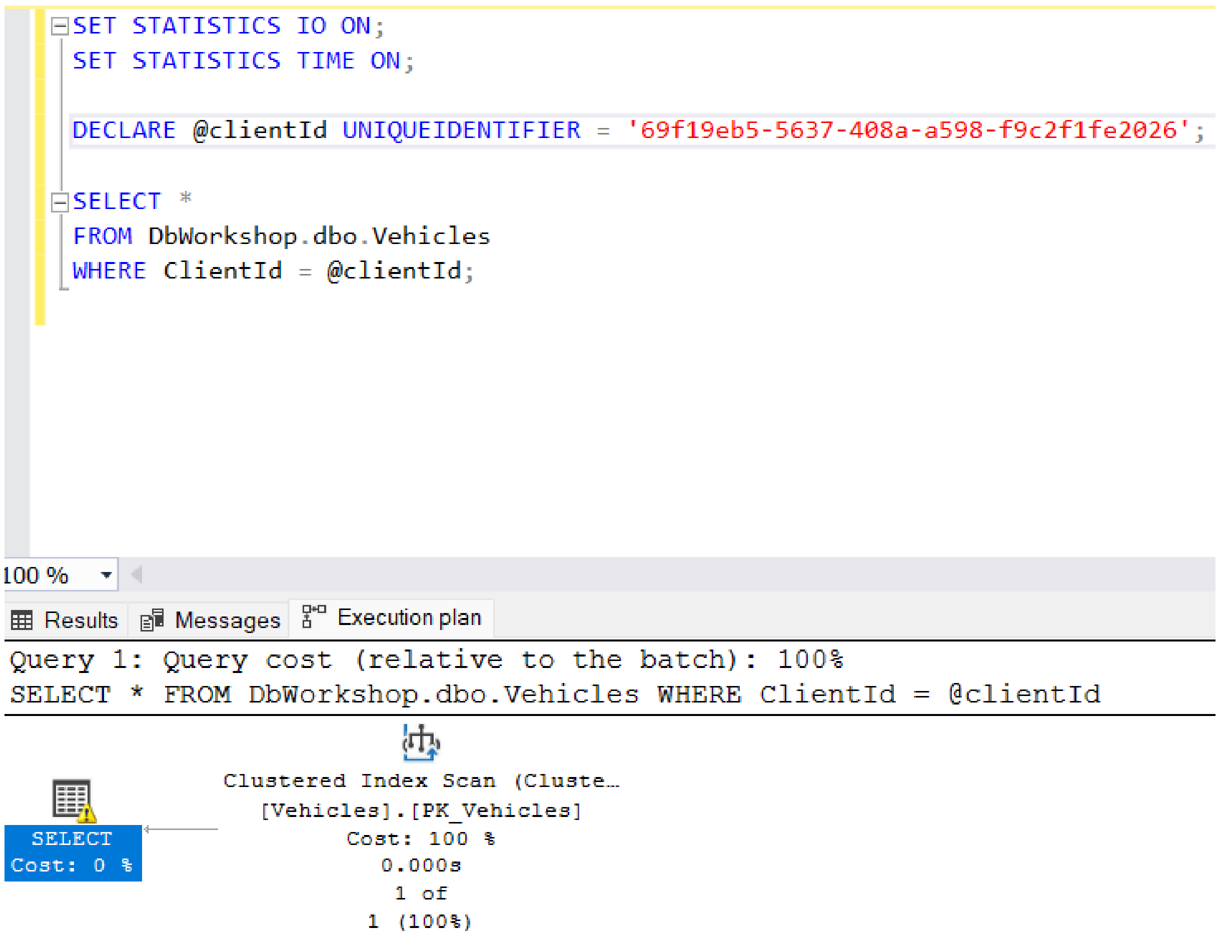
Jakub Nawrocki, Adam Kiraly

Indeksy optymalizacja zapytań

Przed dodaniem indeksu:

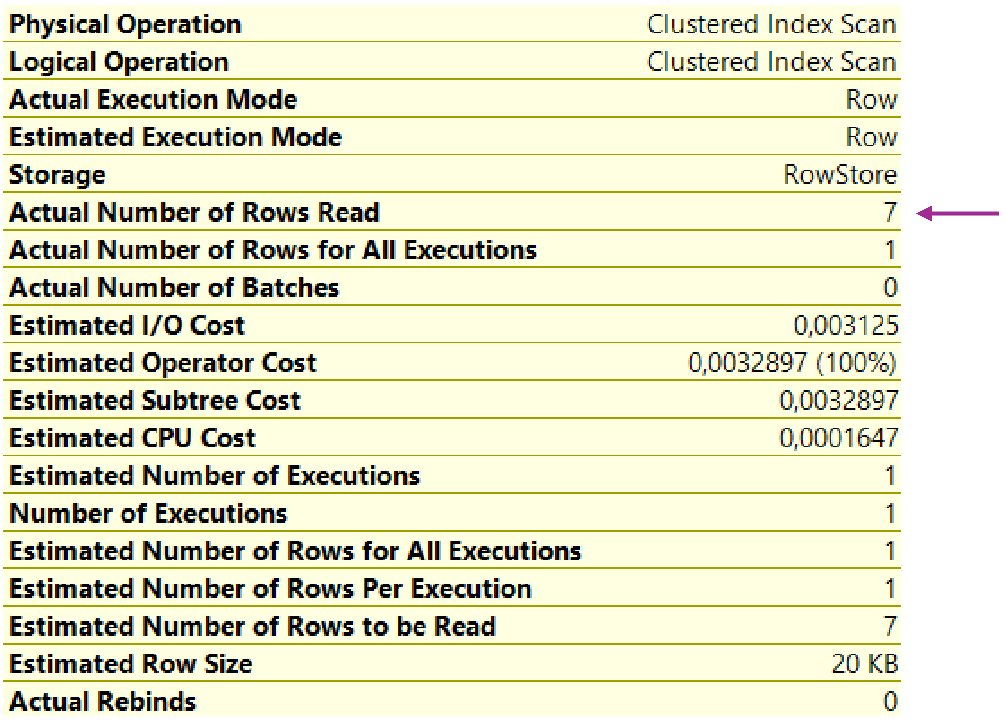


```
var vehicles :List<Vehicle> = await _context.Vehicles // DbSet<Vehicle>
    .Where(v :Vehicle => v.ClientId == clientId) // IQueryable<Vehicle>
    .ToListAsync(); // Task<List<...>>
```



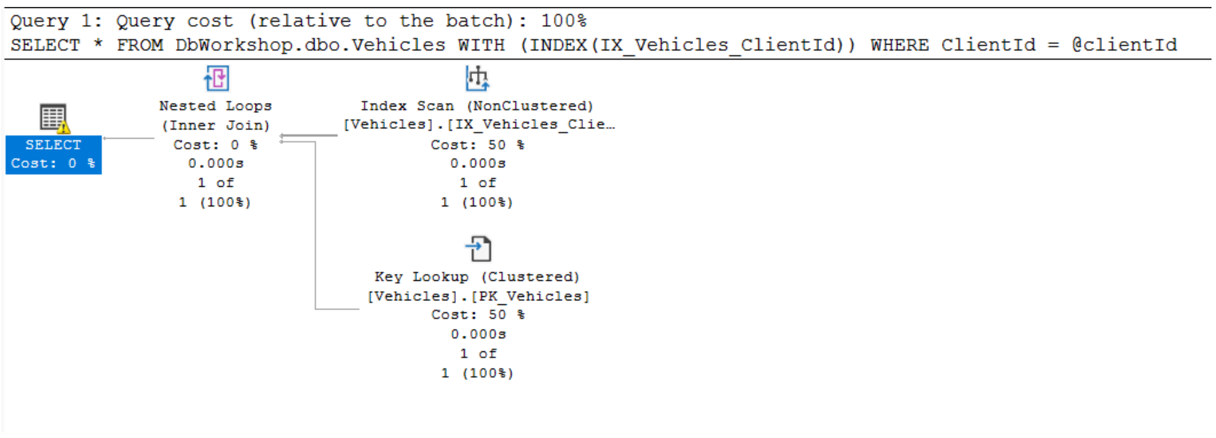
```
SET STATISTICS IO ON;
SET STATISTICS TIME ON;

DECLARE @clientId UNIQUEIDENTIFIER = '69f19eb5-5637-408a-a598-f9c2f1fe2026';

SELECT *
FROM DbWorkshop.dbo.Vehicles
WHERE ClientId = @clientId;
```

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM DbWorkshop.dbo.Vehicles WHERE ClientId = @clientId
```

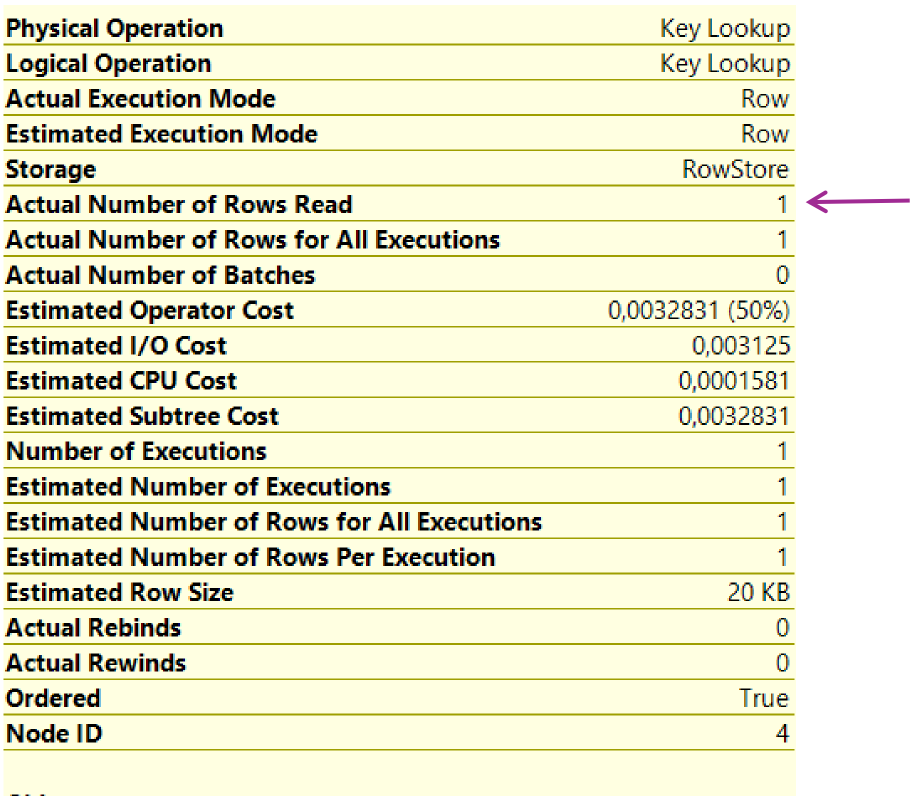


Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows Read	7
Actual Number of Rows for All Executions	1
Actual Number of Batches	0
Estimated I/O Cost	0,003125
Estimated Operator Cost	0,0032897 (100%)
Estimated Subtree Cost	0,0032897
Estimated CPU Cost	0,0001647
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Number of Rows to be Read	7
Estimated Row Size	20 KB
Actual Rebinds	0

Po dodaniu indeksu nieklastrowanego:

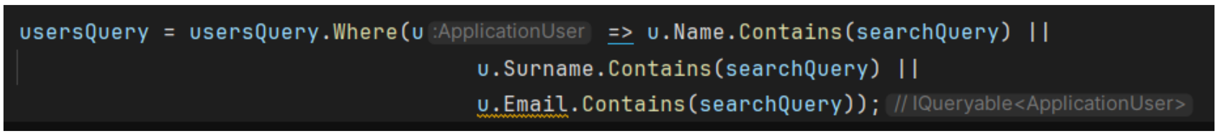


```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM DbWorkshop.dbo.Vehicles WITH (INDEX(IX_Vehicles_ClientId)) WHERE ClientId = @clientId
```

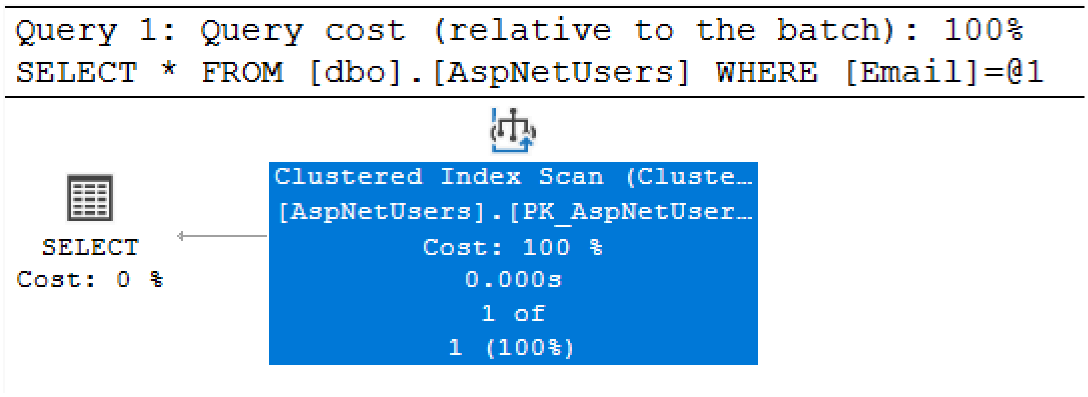


Physical Operation	Key Lookup
Logical Operation	Key Lookup
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows Read	1
Actual Number of Rows for All Executions	1
Actual Number of Batches	0
Estimated Operator Cost	0,0032831 (50%)
Estimated I/O Cost	0,003125
Estimated CPU Cost	0,0001581
Estimated Subtree Cost	0,0032831
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	20 KB
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	4

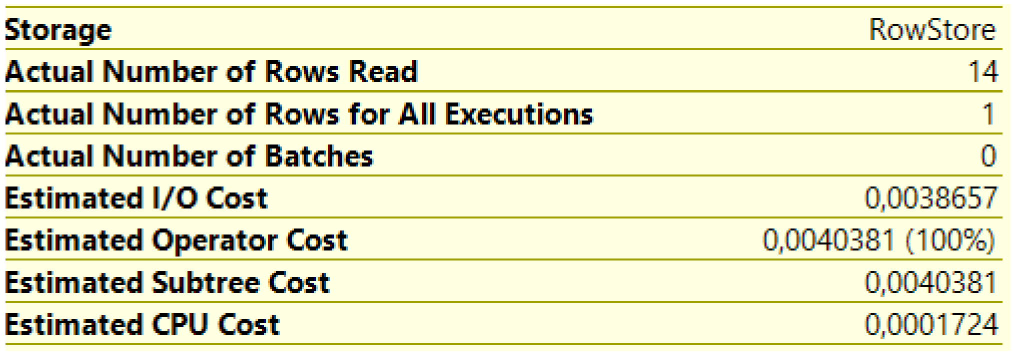
2 Przypadek



```
usersQuery = usersQuery.Where(u :ApplicationUser  => u.Name.Contains(searchQuery) ||
                                              u.Surname.Contains(searchQuery) ||
                                              u.Email.Contains(searchQuery)); // IQueryable<ApplicationUser>
```

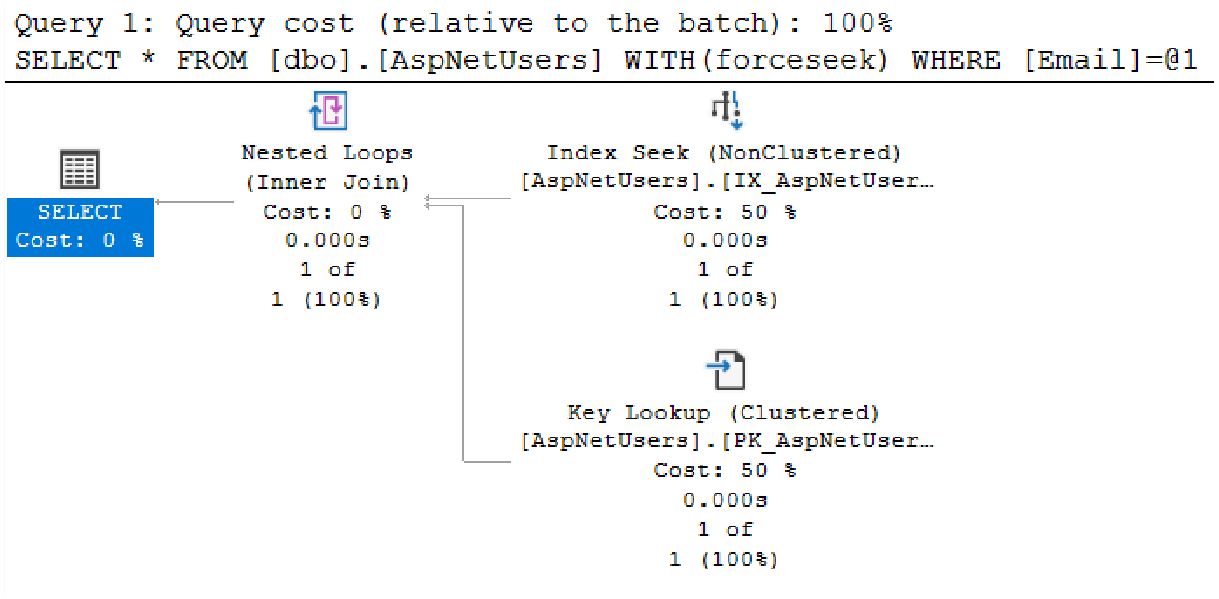


```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[AspNetUsers] WHERE [Email]=@1
```

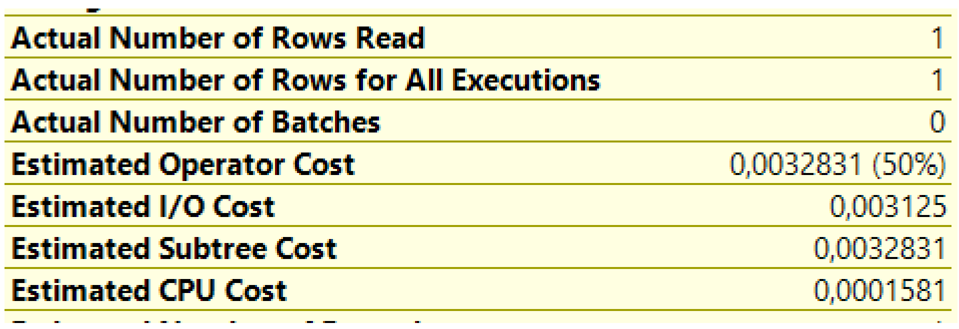


Storage	RowStore
Actual Number of Rows Read	14
Actual Number of Rows for All Executions	1
Actual Number of Batches	0
Estimated I/O Cost	0,0038657
Estimated Operator Cost	0,0040381 (100%)
Estimated Subtree Cost	0,0040381
Estimated CPU Cost	0,0001724

Po dodaniu indeksu nieklastrowanego:



```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [dbo].[AspNetUsers] WITH(forceseek) WHERE [Email]=@1
```



Actual Number of Rows Read	1
Actual Number of Rows for All Executions	1
Actual Number of Batches	0
Estimated Operator Cost	0,0032831 (50%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0032831
Estimated CPU Cost	0,0001581

Wnioski:

Indeks nieklastrowany znacznie zwiększa wydajność zapytania, ponieważ pozwala SQL Server na bezpośrednie dotarcie do interesujących danych, zamiast przeszukiwania całej tabeli, jednak nie zawsze jest opłacalne w przypadku małych baz danych.