Optymalizacja indeksów w WorkshopManager

ZAPYTANIE 1

```
var orders = _context.ServiceOrders
    .Include(o => o.ServiceTasks
    .Include(o => o.Comments)
    .Select(o => new ServiceOrderDto
    {
        Id = o.Id,
        Status = o.Status,
        VehicleId = o.VehicleId,
        AssignedMechanicId = o.AssignedMechanicId,
        ServiceTaskIds = o.ServiceTasks.Select(t => t.Id).ToList(),
        CommentIds = o.Comments.Select(c => c.Id).ToList()
    }).ToList();
```

SQL generowany przez EF:

```
SELECT [s].[Id], [s].[Status], [s].[VehicleId],
[s].[AssignedMechanicId],
       [t].[ServiceOrderId], [t].[Id] AS [TaskId],
       [c].[ServiceOrderId], [c].[Id] AS [CommentId]
FROM [ServiceOrders] AS [s]
LEFT JOIN [ServiceTasks] AS [t] ON [s].[Id] = [t].[ServiceOrderId]
LEFT JOIN [Comments] AS [c] ON [s].[Id] = [c].[ServiceOrderId]
ORDER BY [s].[Id], [t].[Id], [c].[Id]
```

Testowanie przed utworzeniem indeksów:

```
(24 rows affected)
Table 'Comments'. Scan count 1, logical reads 51, physic
Table 'ServiceTasks'. Scan count 1, logical reads 45, ph
Table 'ServiceOrders'. Scan count 1, logical reads 3, ph

Completion time: 2025-06-03T17:00:49.5303707+02:00
```

Tworzenie indeksów:

```
CREATE NONCLUSTERED INDEX IX_ServiceTasks_ServiceOrderId
ON ServiceTasks (ServiceOrderId)
INCLUDE (Id, Description);

CREATE NONCLUSTERED INDEX IX_Comments_ServiceOrderId
ON Comments (ServiceOrderId)
INCLUDE (Id);
```

wyniki po dodaniu indeksów:

```
(24 rows affected)
Table 'Comments'. Scan count 17, logical reads 34, physical reads
Table 'ServiceTasks'. Scan count 15, logical reads 30, physical r
Table 'ServiceOrders'. Scan count 1, logical reads 3, physical re

Completion time: 2025-06-03T17:00:03.2179124+02:00
```

Analiza korzyści

Tabela	PRZED	PO	Poprawa	Scan Count
Comments	51 reads	34 reads	-33%	1 -> 17
ServiceTasks	45 reads	30 reads	-33%	1 -> 15
ServiceOrders	3 reads	3 reads	0%	1 -> 1
TOTAL	**99 reads**	**67 reads**	**-33%**	

Wnioski:
Dodanie odpowiednich indeksów na kolumnach wykorzystywanych w JOIN i ORDER BY znacząco poprawiło wydajność zapytania – liczba fizycznych odczytów spadła o 33%.

Zapytanie 2

```
var customers = await _context.Customers .Include(c =>
c.Vehicles).ToListAsync();
```

SQL generowany przez EF:

```
SELECT [c].[Id], [c].[FirstName], [c].[LastName], [c].[Email],
[c].[PhoneNumber],
       [v].[Id], [v].[VIN], [v].[RegistrationNumber], [v].[ImageUrl],
[v].[CustomerId]
FROM [Customers] AS [c]
LEFT JOIN [Vehicles] AS [v] ON [c].[Id] = [v].[CustomerId]
ORDER BY [c].[Id], [v].[Id]
```

Testowanie przed utworzeniem indeksów:

```
(45 rows affected)
Table 'Vehicles'. Scan count 1, logical reads 84, physical r
Table 'Customers'. Scan count 1, logical reads 3, physical r

Completion time: 2025-06-03T16:58:45.8391845+02:00
```

Tworzenie indeksów:

```
CREATE NONCLUSTERED INDEX IX_Vehicles_CustomerId_Enhanced
ON Vehicles (CustomerId)
INCLUDE (Id, VIN, RegistrationNumber, ImageUrl);
```

Testowanie po utworzeniu indeksów:

```
(45 rows affected)
Table 'Vehicles'. Scan count 28, logical reads 56, phy
Table 'Customers'. Scan count 1, logical reads 3, phys

Completion time: 2025-06-03T16:44:29.3032328+02:00
```

Analiza korzyści

Tabela	**PRZED**	**PO**	**Poprawa**	**Scan Count**
Vehicles	84 reads	56 reads	-33%	1 -> 28
Customers	3 reads	3 reads		
TOTAL	**99 reads**	**67 reads**	**-33%**	

Wnioski:

- Indeks pozwolił zastąpić pełne skanowanie tabeli punktowym wyszukiwaniem. Każde z 28 operacji odpowiada wyszukiwaniu pojazdów dla konkretnego klienta, co jest bardziej wydajne niż przeglądanie całej tabeli.
- Indeks zawiera wszystkie kolumny wymagane w zapytaniu, co eliminuje konieczność odwoływania się do tabeli bazowej. To główny powód redukcji odczytów.