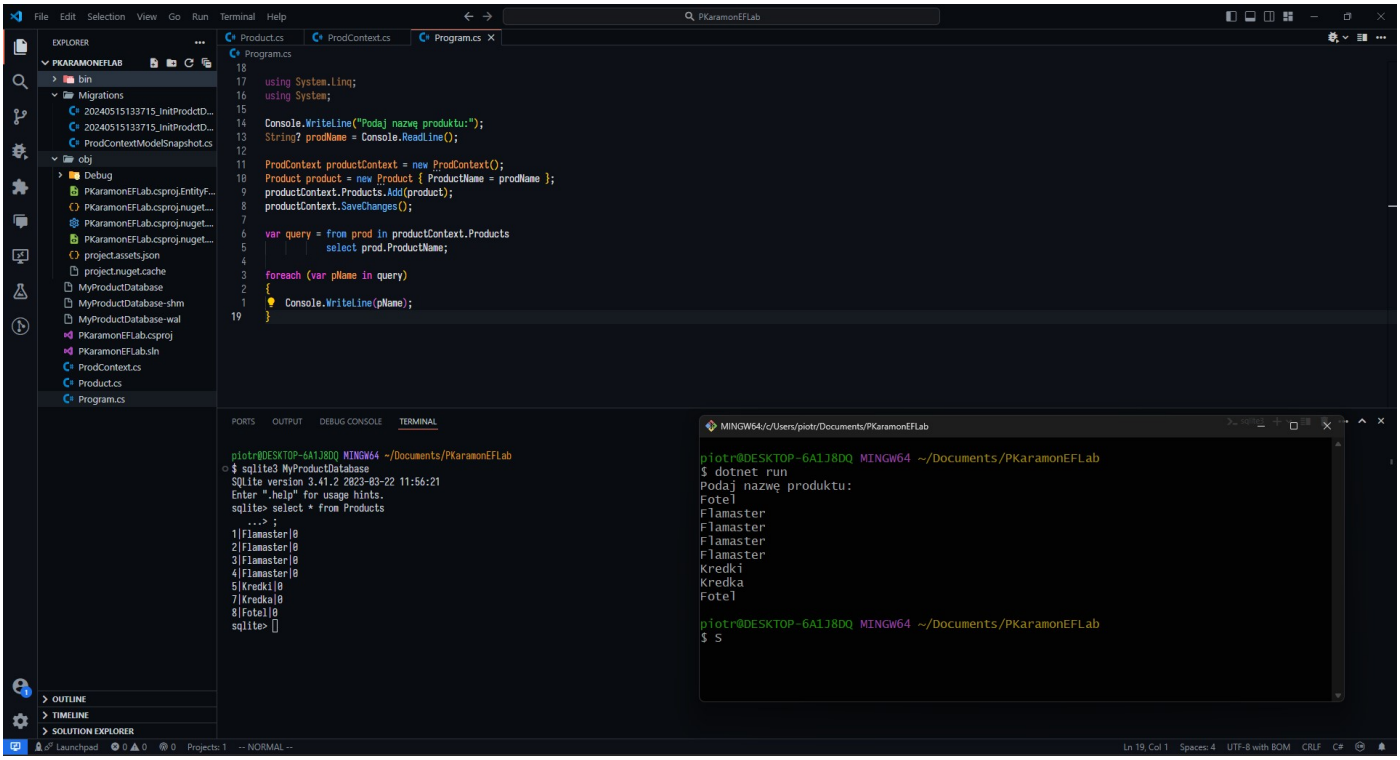
Entity framework - raport

W zespole:

- Bartłomiej Szubiak,
- Szymon Kubiczek,
- Konrad Armatys

Część I



Część II

Podpunkt a

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsOnStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NOT NULL
);
sqlite>
```

Podpunkt b

Program pokazujący działanie relacji klasy Supplier z klasą Product.

```
DataBaseContext prodContext = new DataBaseContext();
var supplier = prodContext.Suppliers.FirstOrDefault();

        if (supplier != null)
        {
            var supplierProducts = prodContext.Entry(supplier)
                                              .Collection(s => s.Products)
                                              .Query()
                                              .ToList();

            foreach (var product in supplierProducts)
            {
                Console.WriteLine($"Product ID: {product.ProductID}");
                Console.WriteLine($"Product Name: {product.ProductName}");
                Console.WriteLine($"Units On Stock: {product.UnitsOnStock}");
                Console.WriteLine();
            }
        }
```

Wynik tego programu.

```
Time Elapsed 00:00:02.49
MacBook-Pro-Szymon:entityFrameworkLab szymonkubiczek$ dotnet run
Product ID: 1
Product Name: flamaster bialy
Units On Stock: 0

Product ID: 2
Product Name: flamaster rozowy
Units On Stock: 0

Product ID: 3
Product Name: flamaster zielony
Units On Stock: 0
```

Wyżej wymienione klasy

```
3 references
public class Supplier
{
    0 references
    public int SupplierID { get; set; }
    1 reference
    public String? CompanyName { get; set; }
    1 reference
    public String? Street { get; set; }
    1 reference
    public String? City { get; set; }
    3 references
    public ICollection<Product> Products { get; set; } = new List<Product>();
}
```

```
public class Product
{
    1 reference
    public int ProductID { get; set;}
    1 reference
    public String? ProductName { get; set;}
    1 reference
    public int UnitsOnStock { get; set;}

}
```

Podpunkt c

```
sqlite> SELECT s.SupplierID, s.CompanyName, p.ProductID, p.ProductName, p.UnitsOnStock
   ...> FROM Suppliers s
   ...> LEFT JOIN Products p ON s.SupplierID = p.SupplierID
   ...> ORDER BY s.SupplierID;
1|Supplier1|1|flamaster bialy|0
1|Supplier1|2|flamaster rozowy|0
1|Supplier1|3|flamaster zielony|0
sqlite>
```

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    // Konfiguracja one to many
    modelBuilder.Entity<Supplier>()
        .HasMany(s => s.Products)
        .WithOne(p => p.Supplier)
        .HasForeignKey(p => p.SupplierID);
}
```

Podpunkt d

Wyświetlanie produktów należących do poszczególnej faktury.

Dzieje się to za pośrednictwem tabeli pośredniej którą Entity Framework tworzy samo.

```
sqlite> SELECT * FROM Products p
   ...> JOIN InvoiceProduct ip ON p.ProductID = ip.ProductsProductID
   ...> JOIN Invoices i ON i.InvoiceID = ip.InvoicesInvoiceID
   ...> WHERE i.InvoiceNumber = 2;
6|flamaster chabrowy|0|2|2|6|2|2|0
7|flamaster seledynowy|0|2|2|7|2|2|0
sqlite>
```

Wyświetlenie faktur do których należy dany produkt.

```
sqlite> SELECT * FROM Invoices i
   ...> JOIN InvoiceProduct ip ON i.InvoiceID = ip.InvoicesInvoiceID
   ...> JOIN Products p ON p.ProductID = ip.ProductsProductID
   ...> WHERE ProductName = "flamaster turkusowy"
   ...> ;
1|1|0|1|4|4|flamaster turkusowy|0|2
sqlite>
```

Podpunkt e

```
sqlite> sqlite> SELECT * FROM Company;
1|customerCompany3|0|Street3|City1|12-345|Customer|0|5.0||
2|customerCompany4|0|Street3|City1|12-345|Customer|0|5.0||
3|supplierCompany1|0|Street1|City1|12-345|Supplier|||0|bankAccountNumber1
4|supplierCompany2|0|Street2|City1|12-345|Supplier|||0|bankAccountNumber2
sqlite>
```

Podpunkt f

Tabela po której dziedziczymy

```
sqlite> SELECT * FROM Companies;
1|customerCompany3|0|Street3|City1|12-345
2|customerCompany4|0|Street3|City1|12-345
3|supplierCompany1|0|Street1|City1|12-345
4|supplierCompany2|0|Street2|City1|12-345
sqlite>
```

Tabele dziedziczące

```
sqlite> SELECT * FROM Suppliers;
3|0|bankAccountNumber1
4|0|bankAccountNumber2
sqlite>
```

```
sqlite> SELECT * FROM Customers;
1|0|5.0
2|0|5.0
sqlite>
```

Podpunkt g

W podpunkcie e została użyta strategia Table=Per-Hierarchy (TPH) , natomiast w podpunkcie f strategia Table-Per-Type (TPT).

Różnice między strategiami:

- Liczba tabel:
  - TPH: Wymaga jednej tabeli dla całej hierarchii klas.
  - TPT: Wymaga osobnej tabeli dla każdej klasy w hierarchii.
- Złożoność relacji:
  - TPH: Mniej relacji między tabelami, co upraszcza zapytania.
  - TPT: Więcej relacji między tabelami, co może skomplikować zapytania.
- Kolumny z wartościami null:
  - TPH: Posiada wiele kolumn z wartościami null, co prowadzi do marnowania przestrzeni.
  - TPT: Nie ma problemu z wartościami null, ponieważ każda tabela przechowuje tylko dane specyficzne dla swojej klasy.
- Tworzenie nowych tabel:
  - TPH: Rozbudowa hierarchii wymaga modyfikacji wszystkich danych w hierarchii.
  - TPT: Tworzenie nowej tabeli nie wpływa na dane w istniejących tabelach.
- Skalowalność:
  - TPH: Przy dużej liczbie dziedziczących tabel schemat może stać się bardzo skomplikowany.
  - TPT: Każda nowa klasa dodaje nową tabelę, co ułatwia rozbudowę i zarządzanie schematem.