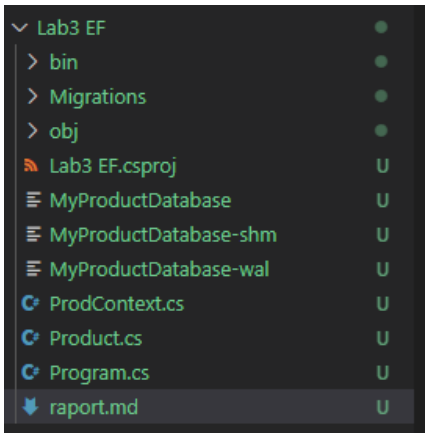
Entity Framework

Imiona i nazwiska autorów: **Tomasz Furgała**

Część I

Udało się zrealizować wszystko zgodnie z tym co było w instrukcji.

Widok projektu



DbContext

```
using Microsoft.EntityFrameworkCore;

4 references
public class ProdContext: DbContext {
    2 references
    public DbSet<Product> Products { get; set; }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("Datasource=MyProductDatabase");
    }
}
```

Product

```
3 references
public class Product {
    0 references
    public int ProductID { get; set; }
    2 references
    public String? ProductName { get; set; }
}
```

Program.cs

```
2
3   Console.WriteLine("Podaj nazwe produktu: ");
4   String? prodName = Console.ReadLine();
5
6   Product product = new Product { ProductName = prodName };
7
8   ProdContext prodContext = new ProdContext();
9   prodContext.Products.Add(product);
10  prodContext.SaveChanges();
11
12  var query = from prod in prodContext.Products
13      select prod.ProductName;
14
15  Console.WriteLine("Lista produktów w bazie: ");
16
17  foreach (var p in query) {
18      Console.WriteLine(p);
19  }
20
21  Console.WriteLine("Hello, World!");
22
```

Wykonanie Program.cs

```
PS C:\Users\tomci\Desktop\Bazy-danych-2024\Lab3 EF> dotnet run
Podaj nazwe produktu:
myszka
Klawiatura
Klawiatura
myszka
Hello, World!
PS C:\Users\tomci\Desktop\Bazy-danych-2024\Lab3 EF>
```

Wyświetlenie bazy przy pomocy sqlite3

```
PS C:\Users\tomci\Desktop\Bazy-danych-2024\Lab3 EF> sqlite3 .\MyProductDatabase
SQLite version 3.45.3 2024-04-15 13:34:05 (UTF-16 console I/O)
Enter ".help" for usage hints.
sqlite> select * from Product;
Parse error: no such table: Product
sqlite> select * from Products;
1|Klawiatura
sqlite> .tables
Products                __EFMigrationsHistory
sqlite> select * from Products;
1|Klawiatura
2|Klawiatura
sqlite> select * from Products;
1|Klawiatura
2|Klawiatura
3|myszka
sqlite>
```

Część II

a. Relacja Suppliers -> Products

Stworzenie nowej klasy Supplier

```
Lab3 EF > Supplier.cs > Supplier > SupplierID
    1 reference
1   public class Supplier {
        0 references
2       public int SupplierID { get; set; }
        0 references
3       public String? CompanyName { get; set; }
        0 references
4       public String? Street { get; set; }
        0 references
5       public String? City { get; set; }
6   }
```

Dodanie zbioru Suppliers do ProdContext

```
Lab3 EF > ProdContext.cs > ProdContext > OnConfiguring
1   using Microsoft.EntityFrameworkCore;
2
    4 references
3   public class ProdContext: DbContext {
        2 references
4       public DbSet<Product> Products { get; set; }
        1 reference
5       public DbSet<Supplier> Suppliers { get; set; }
6
        0 references
7       protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
8       {
9           base.OnConfiguring(optionsBuilder);
10          optionsBuilder.UseSqlite("Datasource=MyProductDatabase");
11      }
12  }
13
```

Dodanie pola Supplier do klasy Product

```
Lab3 EF > Product.cs > Product
    3 references
1   public class Product {
        0 references
2       public int ProductID { get; set; }
        2 references
3       public String? ProductName { get; set; }
        0 references
4       public Supplier? Supplier { get; set; }
5   }
```

Stworzenie nowego dostawcy - Program.cs

```
ProdContext prodContext = new ProdContext();
Supplier supplier = new Supplier { CompanyName = "NowaFirma", City = "Krakow", Street = "Krakowska"};
prodContext.Suppliers.Add(supplier);

prodContext.SaveChanges();
```

udało się dodać nowego dostawcę

```
sqlite> select * from Suppliers;
1|NowaFirma|Krakowska|Krakow
sqlite>
```

Dodanie do istniejącego już produktu stworzonego przed chwilą dostawcę:

```
ProdContext prodContext = new ProdContext();

var product = prodContext.Products.SingleOrDefault(p => p.ProductID == 1);


var supplier = prodContext.Suppliers.SingleOrDefault(p => p.SupplierID == 1);

if (product != null)
{
    product.Supplier = supplier;
    prodContext.SaveChanges();
}
```

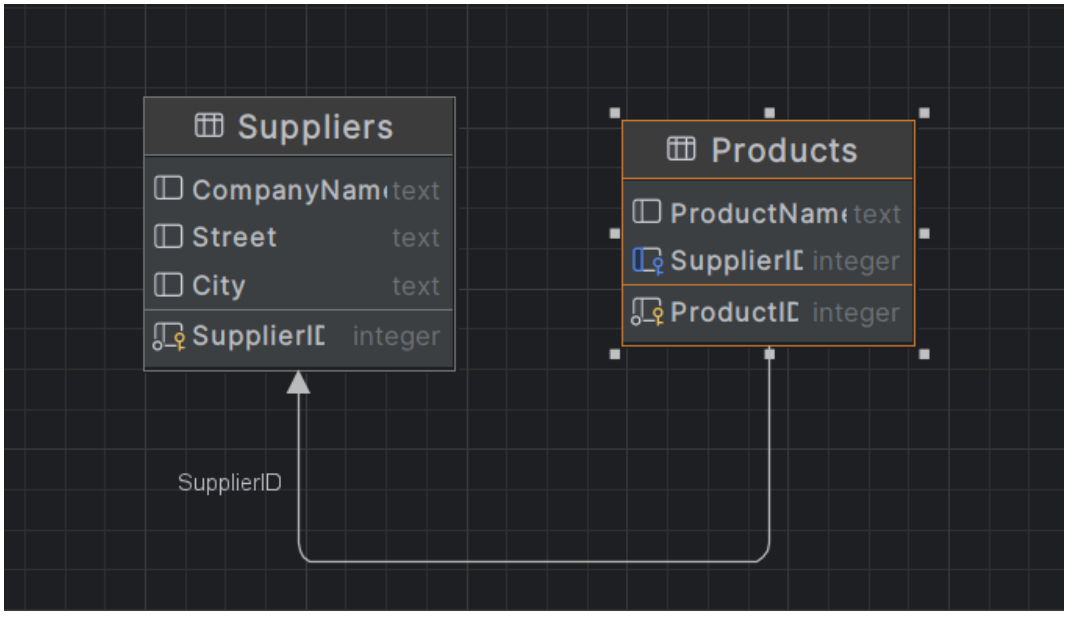
I teraz produkt z ProduktID == 1 ma przypisanego dostawce. Dostawca jest przypisany przez SupplierID.

```
sqlite> select * from Products;
1|Klawiatura|1
2|Klawiatura|
3|myszka|
sqlite>
```

```
ProdContext prodContext = new ProdContext();

var product = prodContext.Products.SingleOrDefault(p => p.ProductID == 1);

Supplier supplier = new Supplier { CompanyName = "Firma2", City = "Warszawa", Street = "Krakowska"};
prodContext.Suppliers.Add(supplier);

if (product != null)
{
    product.Supplier = supplier;
}
prodContext.SaveChanges();
```

```
sqlite> select * from Products;
1|Klawiatura|2
2|Klawiatura|
3|myszka|
sqlite>
```

Diagram wygląda następująco:



b. Odwrócenie relacji

Dodanie listy do przechowywania produktów w supplierze

```
Lab3 EF > Supplier.cs > Supplier > Products
      3 references
1     public class Supplier {
          0 references
2         public int SupplierID { get; set; }
          1 reference
3         public String? CompanyName { get; set; }
          1 reference
4         public String? Street { get; set; }
          1 reference
5         public String? City { get; set; }
6
          1 reference
7         public ICollection<Product> Products { get; set; } = new List<Product>();
8     }
```

Klasa product:

```
Lab3 EF > Product.cs > Product
      8 references
1     public class Product {
          0 references
2         public int ProductID { get; set; }
          3 references
3         public String? ProductName { get; set; }
4     }
```

Program.cs - stworzenie nowych produktów i dodanie ich do listy nowego dostawcy

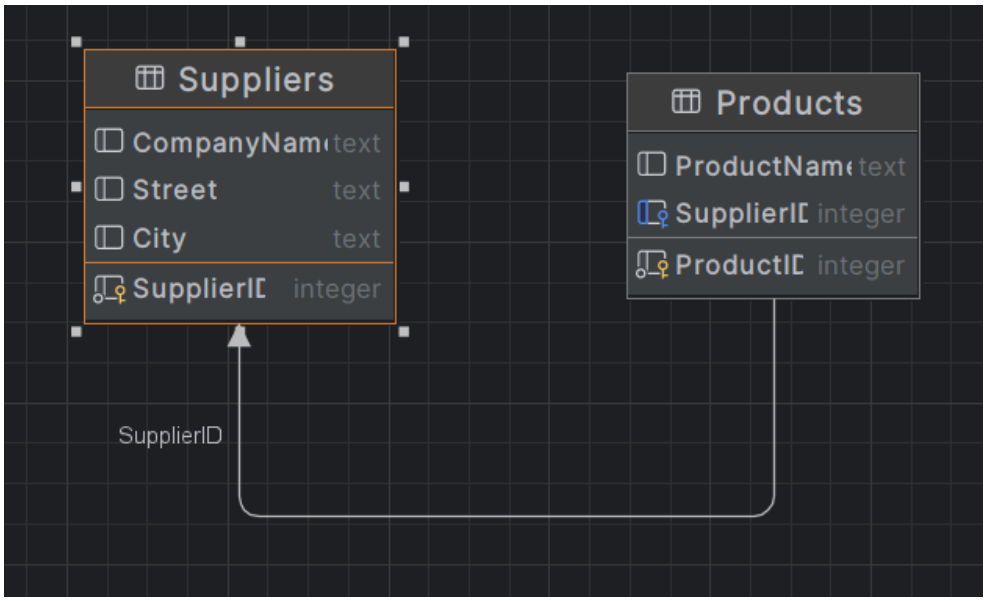
```
ProdContext prodContext = new ProdContext();

Product product1 = new Product { ProductName = "monitor" };
Product product2 = new Product { ProductName = "odubowa" };
Product product3 = new Product { ProductName = "zasilacz" };

prodContext.Products.Add(product1);
prodContext.Products.Add(product2);
prodContext.Products.Add(product3);

Supplier supplier = new Supplier
{
    CompanyName = "FirmaInna",
    City = "Poznan",
    Street = "Poznanska",
    Products = new List<Product> { product1, product2, product3 }
};

prodContext.Suppliers.Add(supplier);

prodContext.SaveChanges();
```

```
sqlite> select * from Products;
1|Klawiatura|2
2|Klawiatura|
3|myszka|
4|monitor|3
5|odubowa|3
6|zasilacz|3
```

```
sqlite> select * from Suppliers;
1|NowaFirma|Krakowska|Krakow
2|Firma2|Krakowska|Warszawa
3|FirmaInna|Poznanska|Poznan
sqlite>
```

Udało się dodać zarówno produkty jak i dostawce.

Jak możemy zauważyć, pomimo zapisania relacji w Entity Frameworku w odwrotny sposób, w bazie danych relacja wciąż wygląda tak samo.Widzimy więc, że Entity Framework „pod spodem" dokonał optymalizacji, dzięki czemu nie musimy trzymać w tabeli `Suppliers` powielonych danych dostawców, różniących się jedynie `SupplierID` oraz kluczem obcym, wskazującym na produkt z tabeli `Products`.



c. Relacja dwustronna

Połączenie dwóch poprzednich podejść. Klasa Suppliers posiada pole z listą produktów, natomiast Products posida pole przechowujące obiekt dostawcy (w rzeczywistości SupplierID).

```
Lab3 EF > Supplier.cs > Supplier > Products
    4 references
1   public class Supplier {
        0 references
2       public int SupplierID { get; set; }
        1 reference
3       public String? CompanyName { get; set; }
        1 reference
4       public String? Street { get; set; }
        1 reference
5       public String? City { get; set; }
6
        1 reference
7       public ICollection<Product> Products { get; set; } = new List<Product>();
8   }
```

```
Lab3 EF > Product.cs > Product
    10 references
1   public class Product {
        0 references
2       public int ProductID { get; set; }
        3 references
3       public String? ProductName { get; set; }
        0 references
4       public Supplier? Supplier { get; set; }
5   }
```

Program.cs - stworzenie nowych produktów i nowego dostawce, i połacenie ze sobą obiektów

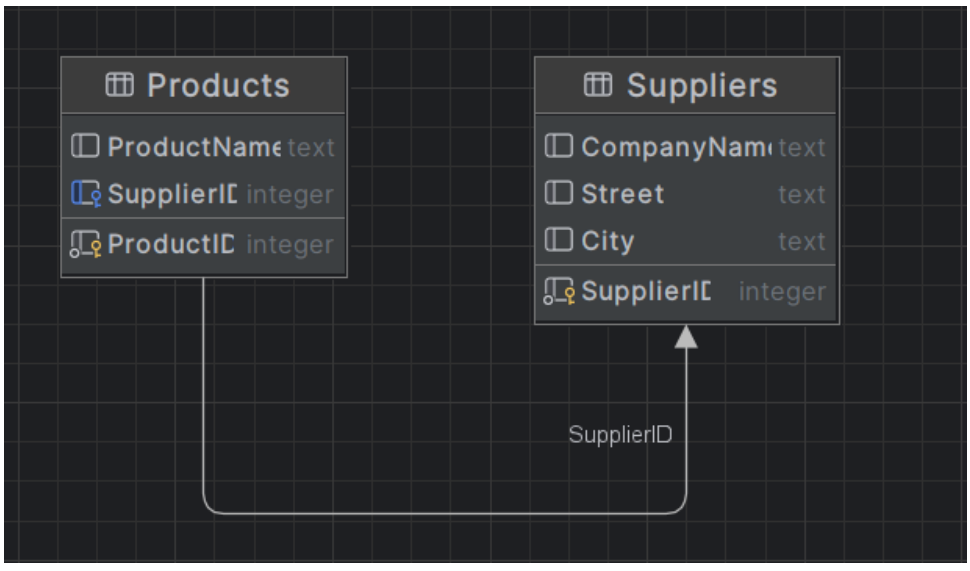
Stworzyłem kilka produktów i nowego dostawce. Dodałem produkty do listy produktów należącej do utworzonego dostawcy i na koniec dodałem do każdego z produktów dostawce.

```
ProdContext prodContext = new ProdContext();

Product product1 = new Product { ProductName = "dlugopis" };
Product product2 = new Product { ProductName = "olowek" };
Product product3 = new Product { ProductName = "linijka" };

prodContext.Products.Add(product1);
prodContext.Products.Add(product2);
prodContext.Products.Add(product3);

Supplier supplier = new Supplier
{
    CompanyName = "FirmaABC",
    City = "Szczecin",
    Street = "Baltycka",
    Products = new List<Product> { product1, product2, product3 }
};

product1.Supplier = supplier;
product2.Supplier = supplier;
product3.Supplier = supplier;

prodContext.Suppliers.Add(supplier);

prodContext.SaveChanges();
```

```
sqlite> select * from Suppliers;
1|NowaFirma|Krakowska|Krakow
2|Firma2|Krakowska|Warszawa
3|FirmaInna|Poznanska|Poznan
4|FirmaABC|Baltycka|Szczecin
```

```
sqlite> select * from Products;
1|Klawiatura|2
2|Klawiatura|
3|myszka|
4|monitor|3
5|odubowa|3
6|zasilacz|3
7|dlugopis|4
8|olowek|4
9|linijka|4
```

Ponownie obserwujemy taki sam diagram. Możemy więc dojść do wniosku, że Entity Framework pozwala nam na stworzenie relacji dwukierunkowej (lub w odwrotnym kierunku do tego, w którym relacja zostanie zapisana, jak widzieliśmy w poprzednim przykładzie), po to, aby łatwiej móc manipulować powiązanymi ze sobą obiektami.



d. Relacja wiele do wielu

Klasy `Products` i `Invoices` będą zawierały listy odpowiednio faktur i produktów. Do zamodelowania takiej relacji będziemy potrzebowali tebeli pomocniczej `InvoicItems`.

Klasa Product

```
Lab3 EF > Product.cs > ...
      12 references
1     public class Product
2     {
          0 references
3         public int ProductID { get; set; }
          3 references
4         public string? ProductName { get; set; }
          5 references
5         public int UnitsInStock { get; set; }
          0 references
6         public Supplier? Supplier { get; set; }
          0 references
7         public virtual ICollection<InvoiceItem> InvoiceItems { get; set; } = new List<InvoiceItem>();
8     }
9
```

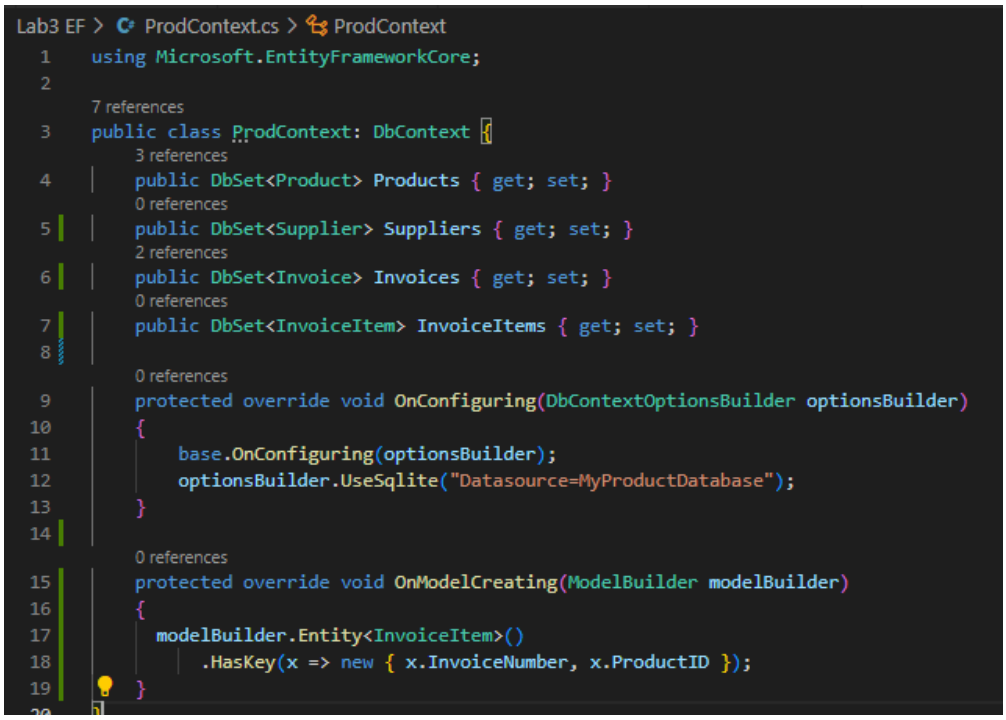
Klasa Invoice

```
Lab3 EF > Invoice.cs > Invoice
1     using System.ComponentModel.DataAnnotations;
2
      6 references
3     public class Invoice
4     {
5         [Key]
          0 references
6         public int InvoiceNumber { get; set; }
          0 references
7         public virtual ICollection<InvoiceItem> InvoiceItems { get; set; } = new List<InvoiceItem>();
8     }
```

Klasa InvoiceItem - Klasa pomocnicza, reprezentująca pozycje faktury

```
Lab3 EF > InvoiceItem.cs > ...
1     using System.ComponentModel.DataAnnotations;
2
      12 references
3     public class InvoiceItem
4     {
          1 reference
5         public int InvoiceNumber { get; set; }
          1 reference
6         public int ProductID { get; set; }
          2 references
7         public virtual Invoice Invoice { get; set; }
          2 references
8         public virtual Product Product { get; set; }
          2 references
9         public int Quantity { get; set; } = 1;
10    }
11
```

prodContext



Program.cs

Stworzyłem nowe produkty, które później dodaje do słownika gdzie przechowuje ilość zakupionych produktów. Następnie dodaje produkty do listy w `InvoiceItem`. Tworzę faktury zawierające obiekty `InvoiceItem`.

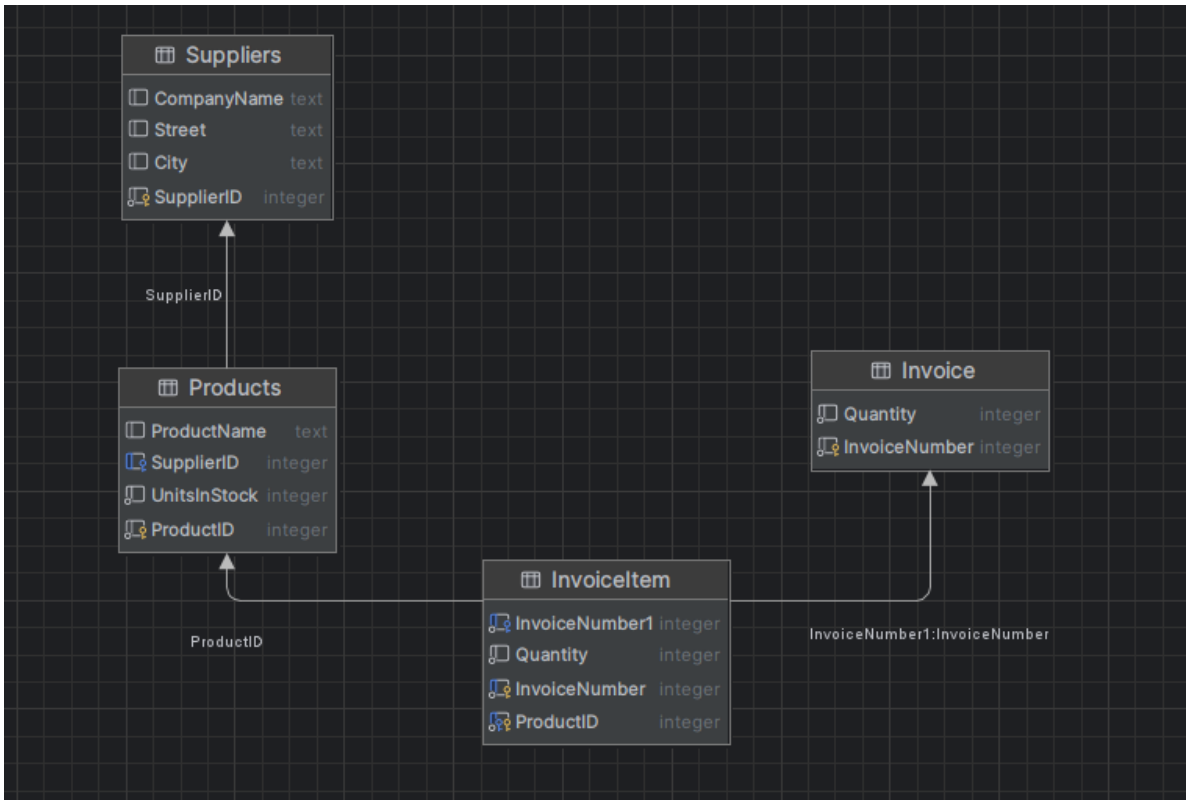
```
34
35  ProdContext prodContext = new ProdContext();
36
37  Product product1 = new Product { ProductName = "trzewiki", UnitsInStock = 2 };
38  Product product2 = new Product { ProductName = "klapki", UnitsInStock = 5 };
39  Product product3 = new Product { ProductName = "szpilki", UnitsInStock = 3 };
40
41  prodContext.Products.Add(product1);
42  prodContext.Products.Add(product2);
43  prodContext.Products.Add(product3);
44
45  Invoice invoice1 = new Invoice{};
46  Invoice invoice2 = new Invoice{};
47
48  prodContext.Invoices.Add(invoice1);
49  prodContext.Invoices.Add(invoice2);
50
51  Dictionary<Product, int> products1 = new();
52  products1.Add(product1, 1);
53  products1.Add(product2, 2);
54
55  List<InvoiceItem> transaction1 = new();
56  foreach (var kvp in products1)
57  {
58      var product = kvp.Key;
59      var quantity = kvp.Value;
60
61      InvoiceItem invoiceItem = new InvoiceItem { Product = product, Invoice = invoice1, Quantity = quantity };
62      product.UnitsInStock -= quantity;
63      prodContext.InvoiceItems.Add(invoiceItem);
64      transaction1.Add(invoiceItem);
65  }
66
67  Dictionary<Product, int> products2 = new();
68  products2.Add(product3, 2);
69
70  List<InvoiceItem> transaction2 = new();
71  foreach (var kvp in products2)
72  {
73      var product = kvp.Key;
74      var quantity = kvp.Value;
75
76      InvoiceItem invoiceItem = new InvoiceItem { Product = product, Invoice = invoice2, Quantity = quantity};
77      product.UnitsInStock -= quantity;
78      prodContext.InvoiceItems.Add(invoiceItem);
79      transaction2.Add(invoiceItem);
80  }
81
82  prodContext.SaveChanges();
83
```

```
sqlite> select * from Products;
1|trzewiki||1
2|klapki||3
3|szpilki||1
```

```
sqlite> select * from Invoices;
1
2
```

```
sqlite> select * from InvoiceItems;
0|4|3|1
0|5|3|2
0|6|4|2
```

Diagram



e. Table-Per-Hierarchy

Klasa Company

```
Lab3 EF > Company.cs > Company > Street
    6 references
1   public class Company
2   {
        0 references
3       public int CompanyID { get; set; }
        3 references
4       public string CompanyName { get; set; } = String.Empty;
        3 references
5       public string Street { get; set; } = String.Empty;
        3 references
6       public string City { get; set; } = String.Empty;
        3 references
7       public string ZipCode { get; set; } = String.Empty;
8   }
```

Supplier

```
Lab3 EF > Supplier.cs > Supplier
    5 references
1   public class Supplier : Company {
        0 references
2       public int SupplierID { get; set; }
        1 reference
3       public string BankAccountNumber { get; set; } = String.Empty;
4   }
```

Customer

```
Lab3 EF > Customer.cs > Customer
    5 references
1   public class Customer : Company
2   {
        0 references
3       public int CustomerID { get; set; }
        2 references
4       public int Discount { get; set; }
5   }
```

CompContext

Lab3 EF > CompContex.cs > CompContext > OnConfiguring

```
using Microsoft.EntityFrameworkCore;

4 references
public class CompContext : DbContext
{
    3 references
    public DbSet<Company>? Companies { get; set; }

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

        modelBuilder.Entity<Company>().ToTable("Companies"); // Map base class to table
        modelBuilder.Entity<Supplier>().HasBaseType<Company>(); // Map Supplier to base class
        modelBuilder.Entity<Customer>().HasBaseType<Company>(); // Map Customer to base class
    }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("Datasource=CompanyDB");
    }
}
```

Program.cs - dodanie klientów i dostawców

Lab3 EF > Program.cs

```
CompContext compContext = new CompContext();

Customer customer1 = new Customer
      {
        CompanyName = "FirmaABC",
        Street = "Krakowska",
        City = "Krakow",
        ZipCode = "35",
        Discount = 10
      };

Customer customer2 = new Customer {
        CompanyName = "FirmNR1",
        Street = "Poznanska",
        City = "Wieliczka",
        ZipCode = "30",
        Discount = 100
      };

Supplier supplier1 = new Supplier {
        CompanyName = "FirmNR2",
        Street = "Reymonta",
        City = "Warszawa",
        ZipCode = "40",
        BankAccountNumber = "000123"
      };

compContext.Companies.Add(customer1);
compContext.Companies.Add(customer2);
compContext.Companies.Add(supplier1);

compContext.SaveChanges();
```

Wynik

Otrzymujemy tylko 1 tabele zawierającą zarówno pola zdefiniowane w klasie Customers jak i Suppliers.

	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	CustomerID	Discount	SupplierID	BankAccountNumber
1	1	FirmaABC	Krakowska	Krakow	35	Customer	0	10	<null>	<null>
2	2	FirmNR1	Poznanska	Wieliczka	30	Customer	0	100	<null>	<null>
3	3	FirmNR2	Reymonta	Warszawa	40	Supplier	<null>	<null>	0	000123

f. Table-Per-Type

Klasa Company

Lab3 EF > Company.cs > Company

```
4 references
public class Company
{
    0 references
    public int CompanyID { get; set; }
    3 references
    public string CompanyName { get; set; } = String.Empty;
    3 references
    public string Street { get; set; } = String.Empty;
    3 references
    public string City { get; set; } = String.Empty;
    3 references
    public string ZipCode { get; set; } = String.Empty;
}
```

Supplier

Lab3 EF > Supplier.cs > Supplier

```
6 references
public class Supplier : Company {
    0 references
    public int CompanyID { get; set; }
    1 reference
    public string BankAccountNumber { get; set; } = String.Empty;
}
```

Customer

Lab3 EF > Customer.cs > Customer

```
6 references
public class Customer : Company
{
    0 references
    public int CompanyID { get; set; }
    2 references
    public int Discount { get; set; }
}
```

CompContext

Lab3 EF > CompContex.cs > CompContext

```
using Microsoft.EntityFrameworkCore;

5 references
public class CompContext : DbContext
{
    3 references
    public DbSet<Company>? Companies { get; set; }
    0 references
    public DbSet<Supplier>? Suppliers { get; set; }
    0 references
    public DbSet<Customer>? Customers { get; set; }

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

        modelBuilder.Entity<Company>().ToTable("Companies");
        modelBuilder.Entity<Supplier>().ToTable("Suppliers");
        modelBuilder.Entity<Customer>().ToTable("Customers");
    }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("Datasource=CompanyDB2");
    }
}
```

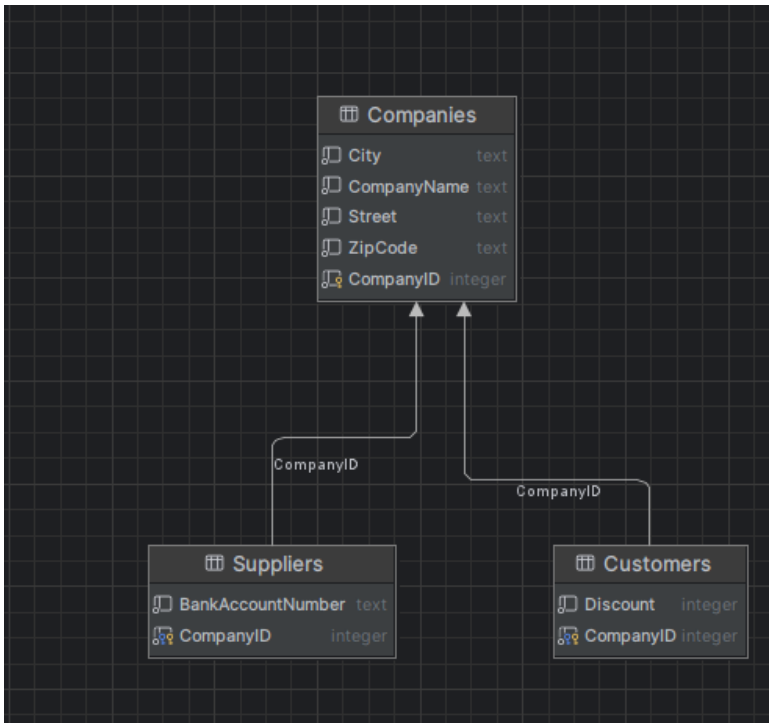
Program.cs - dodanie klientów i dostawców

Lab3 EF > Program.cs

```
CompContext compContext = new CompContext();

Customer customer1 = new Customer
      {
        CompanyName = "FirmaABC",
        Street = "Krakowska",
        City = "Krakow",
        ZipCode = "35",
        Discount = 10
      };

Customer customer2 = new Customer {
        CompanyName = "FirmNR1",
        Street = "Poznanska",
        City = "Wieliczka",
        ZipCode = "30",
        Discount = 100
      };

Supplier supplier1 = new Supplier {
        CompanyName = "FirmNR2",
        Street = "Reymonta",
        City = "Warszawa",
        ZipCode = "40",
        BankAccountNumber = "000123"
      };

compContext.Companies.Add(customer1);
compContext.Companies.Add(customer2);
compContext.Companies.Add(supplier1);

compContext.SaveChanges();
```

Diagram

Otrzymujemy tylko 3 tabele. Tabele `Customers` i `Suppliers` są połączone z `Companies` za pomocą po `CompanyID`.



Companies

	CompanyID	City	CompanyName	Street	ZipCode
1	1	Krakow	FirmaABC	Krakowska	35
2	2	Wieliczka	FirmNR1	Poznanska	30
3	3	Warszawa	FirmNR2	Reymonta	40

Suppliers

	CompanyID	BankAccountNumber
1	3	000123

Customers

	CompanyID	Discount
1	1	10
2	2	100

g. Porównanie

Table-Per-Hierarchy

Tworzona jest jedna tabela, która zawiera wspólne dla klas dziedziczących dane oraz dane, charakteryzujące każdą z klas dziedziczących z osobna. W przypadku, gdy klasa dziedzicząca posiada atrybut, którego nie ma w klasie, z której dziedziczy, dodawana jest osobna kolumna, w której dla pozostałych klas wpisane są wartości null, a dla tej klasy, odpowiednie wartości tego parametru.

Zalety

- Prostsze i bardziej wydajne zapytania: TPH używa jednej tabeli dla wszystkich klas dziedziczących, co oznacza, że zapytania są prostsze i łatwiejsze do wykonania (nie wymagają klauzuli join).
- Mniejsza liczba tabel: TPH wymaga tylko jednej tabeli, co prowadzi do mniejszej liczby tabel w bazie danych.
- Łatwiejsze mapowanie: Mniej konfiguracji jest wymagane, ponieważ wszystkie klasy dziedziczące mapowane są do jednej tabeli.

Wady

- W przypadku wielu klas dziedziczących z tej samej klasy, jedna tabela nie jest dobrym rozwiązaniem, ponieważ będzie zawierała bardzo dużo wartości null (marnowanie miejsca),
- Grupowanie danych, w przypadku wielu klas dziedziczących, zmniejsza przejrzystość schematu bazy danych.

Table-Per-Type

Tworzone jest kilka tabel (osobne tabele dla każdej z klas, zarówno tej, z której dziedziczą klasy, jak i klas dziedziczących). Tabele klas dziedziczących są łączone z tabelą klasy, z której dziedziczą. przy pomocy relacji 1 do 1.

Zalety

- Normalizacja danych: Każda klasa dziedzicząca mapowana jest do osobnej tabeli, co prowadzi do bardziej znormalizowanej struktury danych.
- Bardziej zrozumiały digram bazy danych, gdzie widzimy jak klasy dziedziczą po innych klasach.
- Takie podejście nie wymaga trzymania pustych wartości w tabelach (null), dzięki czemu zapisywane są tylko wartości.

Wady

- Konieczne jest wykonywanie wielu operacji join (łączenie tabel klas dziedziczących z tabelą klasy nadrzędnej – tej, z której klasy dziedziczą).

Osobiście bedziej podobała mi się strategia Table-Per-Type ze względu na bardziej zrozumiałą i naturalną strukture