

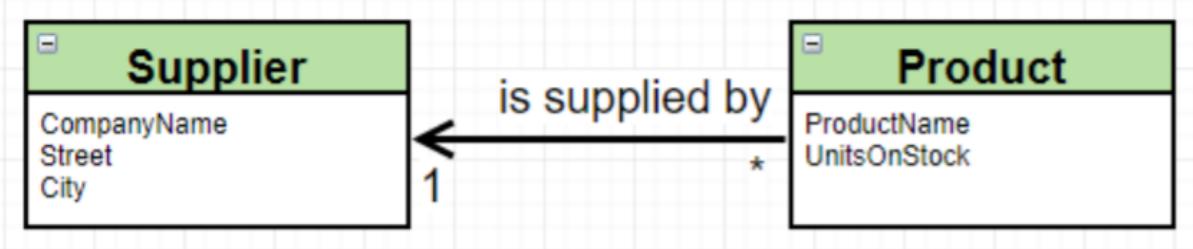
# Laboratorium 2: Entity Framework

Barbara Wojtarowicz

Część II: Praca samodzielna

## Zadania 1-6:

1. **Zadanie I:** Zmodyfikowano model wprowadzający pojęcie dostawcy według relacji:  
Product is supplied by Supplier



- 1.1. Dodano klasę Supplier

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BarbaraWojtarowiczEFProducts
{
    public class Supplier
    {
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
    }
}
```

- 1.2. Do klasy Product dodano pole Supplier

```
using System;
using System.Collections.Generic;
using System.Text;

namespace BarbaraWojtarowiczEFProducts
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
        public Supplier Supplier { get; set; }
    }
}
```

```
}
```

- 1.3. Do klasy ProductContext dodano nowy DbSet, reprezentujący wszystkich dostawców

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.EntityFrameworkCore;

namespace BarbaraWojtarowiczEFProducts
{
    class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

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

```
:SQLite version 3.35.4 2021-04-02 15:20:15
Enter ".help" for usage hints.
sqlite> .tables
Products          Suppliers          __EFMigrationsHistory
sqlite> .schema Suppliers
CREATE TABLE IF NOT EXISTS "Suppliers" (
    "SupplierID" INTEGER NOT NULL CONSTRAINT "PK_Suppliers" PRIMARY KEY AUTOINCREMENT,
    "CompanyName" TEXT NULL,
    "Street" TEXT NULL,
    "City" TEXT NULL
);
sqlite>
```

- 1.4. Przygotowano program: wczytujący nazwę nowego dostawcy z konsoli, a następnie wyszukujący ostatnio wprowadzony produkt i ustawiający jego dostawcę na właśnie dodanego:

```

using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;

namespace BarbaraWojtarowiczEFProducts
{
    class Program
    {
        static void Main(string[] args)
        {
            // Entering new supplier
            Console.WriteLine("Wprowadź nowego dostawcę");
            string supplierName = Console.ReadLine();

            ProductContext productContext = new ProductContext();
            Supplier supplier = new Supplier { CompanyName =
supplierName };
            productContext.Suppliers.Add(supplier);
            productContext.SaveChanges();

            // Recently entered product and supplier
            var lastProductID = productContext.Products.Max(x =>
x.ProductID);
            var lastProductName = from prod in productContext.Products
                                where prod.ProductID == lastProductID
                                select prod.ProductName;

            var lastSupplierID = productContext.Suppliers.Max(x =>
x.SupplierID);
            var lastSupplierName = from supp in productContext.Suppliers
                                where supp.SupplierID == lastSupplierID
                                select supplier.CompanyName;
            var supplierToSet = productContext.Suppliers.Where(s =>
s.SupplierID == lastSupplierID).FirstOrDefault();

            Product lastProduct = productContext.Products.Where(p =>
p.ProductName ==
lastProductName.FirstOrDefault().ToString()).FirstOrDefault();

            Console.WriteLine("Ostatnio wprowadzony dostawca: " +
lastSupplierName.FirstOrDefault().ToString());
            Console.WriteLine("Ostatnio wprowadzony produkt: " +
lastProduct.ProductName);

            // Changing product's supplier
        }
    }
}

```

```

productContext.Entry(lastProduct).Property("SupplierID").CurrentValue =
supplierToSet.SupplierID;
    productContext.SaveChanges();
    Console.WriteLine("Ostatnio wprowadzonemu produktowi został
przypisany dostawca.");

    // Additional product printing
    Console.WriteLine("PRODUCT LIST:");

    var query = from prod in productContext.Products
                select prod;

    foreach (var q in query)
    {
        Console.Write("Product name: " + q.ProductName);
        if (q.Supplier != null)
        {
            Console.WriteLine(" | supplier: " +
q.Supplier.SupplierID + " " + q.Supplier.CompanyName);
        }
        else
        {
            Console.WriteLine(" | supplier: None");
        }
    }
}
}
}

```

### Wynik działania programu:

The screenshot shows the Microsoft Visual Studio debugger's Output window. It displays the following text output:

```

Konsola debugowania programu Microsoft Visual Studio
Wprowadź nowego dostawcę
Audi Supp
Ostatnio wprowadzony dostawca: Audi Supp
Ostatnio wprowadzony produkt: Audi
Ostatnio wprowadzonemu produktowi został przypisany dostawca.
PRODUCT LIST:
Product name: mleko | supplier: None
Product name: cytryna | supplier: None
Product name: mercedes | supplier: None
Product name: samochód niebieski | supplier: None
Product name: samochód zielony | supplier: None
Product name: Audi | supplier: 56 Audi Supp

```

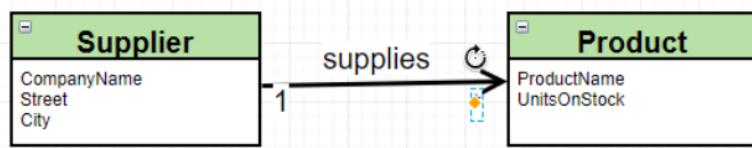
Zmiana w bazie ProductsDatabase widoczna z perspektywy klienta - sqlite

```

sqlite> select * from Products;
1|mleko||0
2|cytryna||0
3|mercedes||0
4|samochód niebieski||0
5|samochód zielony||0
6|Audi|56|0
sqlite>

```

2. **Zadanie II:** Odwrócono relację zachodzącą w pkt. 1 tak, by zachodziła wg następującego schematu:



- 2.1. Z klasy Product usunięto wcześniej dodane pole Supplier. Natomiast do klasy Supplier dodano listę produktów dostarczanych przez danego dostawcę.

```

namespace BarbaraWojtarowiczEFProducts
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
    }
}

```

```

namespace BarbaraWojtarowiczEFProducts
{
    public class Supplier
    {
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public List<Product> Products { get; set; }
    }
}

```

- 2.2. Stworzono program, który: wczytuje nazwę nowego dostawcy, a następnie 4

nazwy nowych produktów, tworzy tego dostawcę i produkty, a następnie dodaje je do jego listy produktów.

```
using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using System.Collections.Generic;
using System.Text;

namespace BarbaraWojtarowiczEFProducts
{
    class Program
    {
        static void Main(string[] args)
        {
            ProductContext productContext = new ProductContext();

            // Entering new supplier
            Console.WriteLine("Wprowadź nowego dostawcę");
            string supplierName = Console.ReadLine();

            // Entering new products
            Console.WriteLine("Wprowadź kilka 4 produkty");
            string[] products = new string[4];
            for (int i = 0; i < 4; i++)
            {
                products[i] = Console.ReadLine();
            }

            Supplier supplier = new Supplier { CompanyName =
supplierName };
            productContext.Suppliers.Add(supplier);
            productContext.SaveChanges();

            Product product1 = new Product { ProductName =
products[0].ToString() };
            Product product2 = new Product { ProductName =
products[1].ToString() };
            Product product3 = new Product { ProductName =
products[2].ToString() };
            Product product4 = new Product { ProductName =
products[3].ToString() };

            productContext.Products.Add(product1);
            productContext.SaveChanges();
```

```

        productContext.Products.Add(product2);
        productContext.SaveChanges();
        productContext.Products.Add(product3);
        productContext.SaveChanges();
        productContext.Products.Add(product4);
        productContext.SaveChanges();

        // Adding products to their supplier's product list
        productContext.Suppliers.Include(s => s.Products).ToList();
        Product[] prods = { product1, product2, product3, product4
    };

        supplier.Products.AddRange(prods);
        productContext.SaveChanges();

        // Additional product printing
        Console.WriteLine("PRODUCT LIST:");

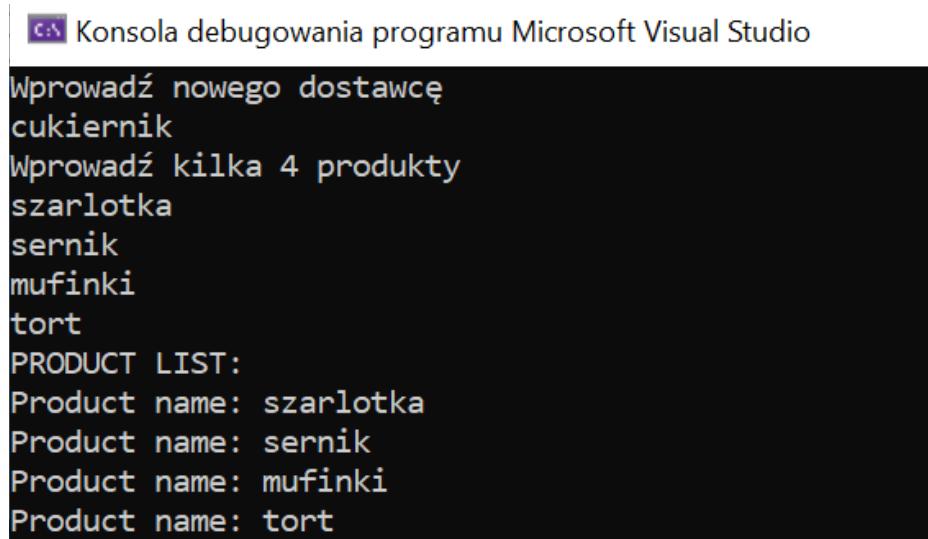
        var prodList = from supp in productContext.Suppliers
                       where supp.CompanyName == supplierName
                       select supp.Products;

        foreach(var p in prodList.FirstOrDefault())
        {
            Console.WriteLine("Product name: " +
p.ProductName.ToString());

        }
    }
}
}
}

```

Efekt działania programu:



```

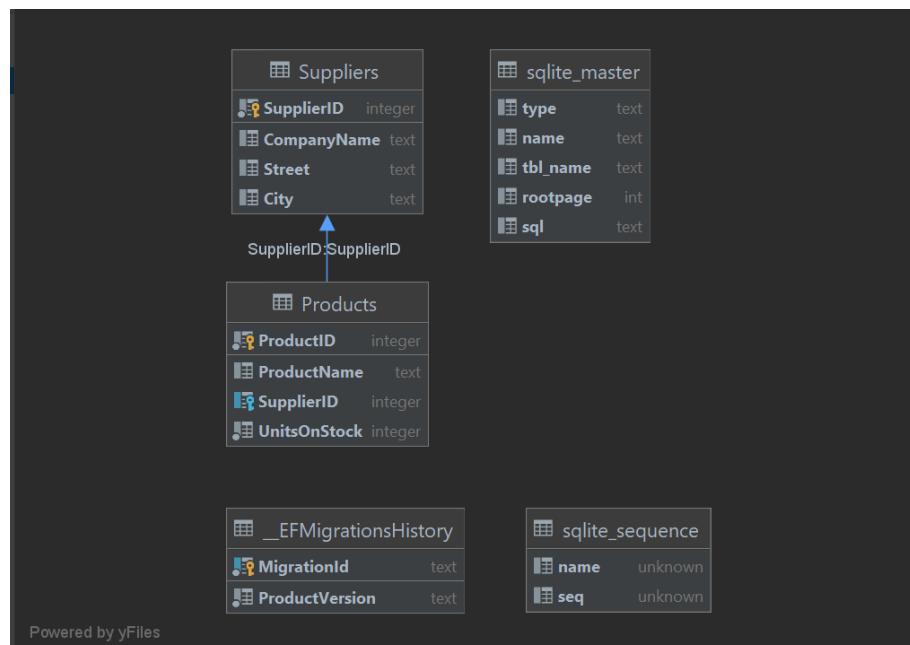
    Konsola debugowania programu Microsoft Visual Studio
Wprowadź nowego dostawcę
cukiernik
Wprowadź kilka 4 produkty
szarlotka
sernik
mufinki
tort
PRODUCT LIST:
Product name: szarlotka
Product name: sernik
Product name: mufinki
Product name: tort

```

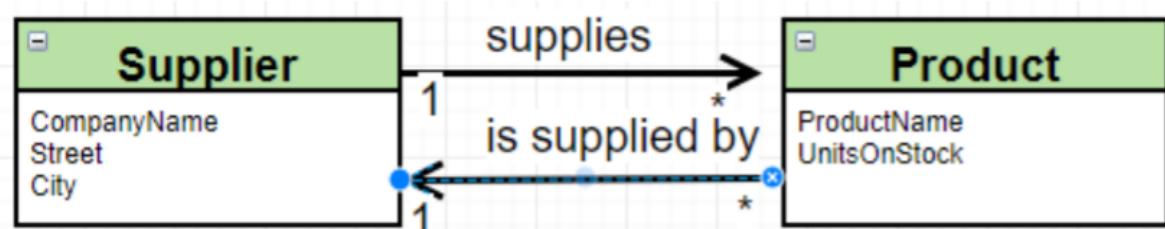
W efekcie działania programu w bazie danych zaszły następujące zmiany:

	SupplierID	CompanyName	Street	City
1		cukiernik	<null>	<null>
	ProductID	ProductName	SupplierID	UnitsOnStock
1		szarlotka	1	0
2		sernik	1	0
3		mufinki	1	0
4		tort	1	0

Schemat bazy wygląda następująco:



3. **Zadanie III:** Zamodelowano relację dwustronną, zobrazowaną następująco:



3.1. Klasa **Supplier** pozostała bez zmian (z listą dostarczanych przez dostawcę produktów). Z kolei do klasy **Product** dodano pole **Supplier**.

```
namespace BarbaraWojtarowiczEFProducts
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
        public Supplier Supplier { get; set; }

    }
}
```

```
namespace BarbaraWojtarowiczEFProducts
{
    public class Supplier
    {
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public List<Product> Products { get; set; }
    }
}
```

- 3.2. Stworzono program, który dodaje kilka produktów oraz ich dostawcę, pamiętając o dwustronności relacji:

```
namespace BarbaraWojtarowiczEFProducts
{
    class Program
    {
        static void Main(string[] args)
        {
            ProductContext productContext = new ProductContext();

            // Entering new supplier
            Console.WriteLine("Wprowadź nowego dostawcę");
            string supplierName = Console.ReadLine();

            // Entering new products
            Console.WriteLine("Wprowadź kilka 4 produkty");
            string[] products = new string[4];
            for (int i = 0; i < 4; i++)
            {
                products[i] = Console.ReadLine();
            }
        }
    }
}
```

```

    }

    Supplier supplier = new Supplier { CompanyName =
supplierName };
    productContext.Suppliers.Add(supplier);
    productContext.SaveChanges();

    Product product1 = new Product { ProductName =
products[0].ToString() };
    Product product2 = new Product { ProductName =
products[1].ToString() };
    Product product3 = new Product { ProductName =
products[2].ToString() };
    Product product4 = new Product { ProductName =
products[3].ToString() };

    productContext.Products.Add(product1);
    productContext.SaveChanges();
    productContext.Products.Add(product2);
    productContext.SaveChanges();
    productContext.Products.Add(product3);
    productContext.SaveChanges();
    productContext.Products.Add(product4);
    productContext.SaveChanges();

    // Adding products to their supplier's product list
    productContext.Suppliers.Include(s => s.Products).ToList();
    Product[] prods = { product1, product2, product3, product4
};

    supplier.Products.AddRange(prods);
    productContext.SaveChanges();

    // Add suppliers to each of the products
    foreach(Product p in prods)
    {
        productContext.Entry(p).Reference(prod =>
prod.Supplier).Load();
        if (p.Supplier == null)
        {

productContext.Entry(p).Property("SupplierID").CurrentValue =
supplier.SupplierID;
        productContext.SaveChanges();
    }
}

```

```

// Additional product printing
Console.WriteLine("PRODUCT LIST:");

var prodList = from supp in productContext.Suppliers
               where supp.CompanyName == supplierName
               select supp.Products;

foreach(var p in prodList.FirstOrDefault())
{
    Console.WriteLine("Product name: " +
p.ProductName.ToString() + " is supplied by: "
+ p.Supplier.CompanyName.ToString());
}

}
}
}
}

```

Konsola debugowania programu Microsoft Visual Studio

```

Wprowadź nowego dostawcę
Mleczarz
Wprowadź kilka 4 produkty
mleko krowie
mleko kozie
śmietana
jogurt
PRODUCT LIST:
Product name: mleko krowie is supplied by: Mleczarz
Product name: mleko kozie is supplied by: Mleczarz
Product name: śmietana is supplied by: Mleczarz
Product name: jogurt is supplied by: Mleczarz

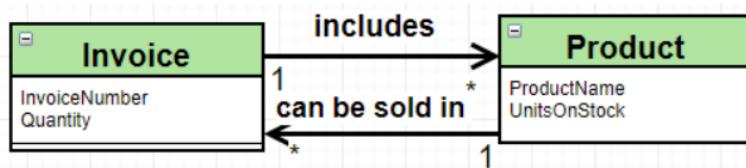
```

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	mleko krowie	1	0
2	2	mleko kozie	1	0
3	3	śmietana	1	0
4	4	jogurt	1	0

	SupplierID	CompanyName	Street	City
1	1	Mleczarz	<null>	<null>

4. **Zadanie IV:** Zamodelowano relację wiele do wielu, jak na schemacie poniżej:



#### 4.1. Dodano klasę Invoice

```
class Invoice
{
    public Invoice()
    {
        InvoiceProducts = new List<InvoiceProduct>();
    }

    public int InvoiceID { get; set; }
    public int InvoiceNumber { get; set; }
    public int Quantity { get; set; }
    public List<InvoiceProduct> InvoiceProducts { get; set; }
}
```

#### 4.2. Dodano klasę InvoiceProduct, reprezentującą relacje zachodzące między Invoice oraz Product - każda faktura może zawierać kilka produktów i jednocześnie każdy produkt może występować na kilku fakturach.

```
class InvoiceProduct
{
    public int ProductID { get; set; }
    public Product Product { get; set; }
    public int InvoiceID { get; set; }
    public Invoice Invoice { get; set; }
}
```

#### 4.3. Zmodyfikowano klasę Product

```
class Product
{
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public int UnitsOnStock { get; set; }
    public Supplier Supplier { get; set; }
    public List<InvoiceProduct> InvoiceProducts { get; set; }
}
```

#### 4.4. Zmodyfikowano klasę ProductContext, dodając dwa DbSet'y, reprezentujące: zbiór faktur oraz zbiór relacji zachodzących między fakturami a produktami. Dodano również metodę OnModelCreating.

```
class ProductContext : DbContext
```

```

{
    public DbSet<Product> Products { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Invoice> Invoices { get; set; }
    public DbSet<InvoiceProduct> InvoiceProducts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("Datasource=ProductsDatabase");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<InvoiceProduct>()
            .HasKey(ip => new { ip.ProductID, ip.InvoiceID });
    }
}

```

- 4.5. Dodano 4 produkty, dostarczane przez danego dostawcę, które umieszczone na dwóch fakturach. Operację dodania do faktury zrealizowano dodając uprzednio 6 relacji produkt-faktura (klasa InvoiceProduct). Następnie relacje te dodano do list relacji zarówno w danych produktach, jak i w dwóch fakturach, w zależności od tego, do której faktury został dodany dany produkt.

```

class Program
{
    static void Main(string[] args)
    {
        ProductContext productContext = new ProductContext();

        // Entering new supplier
        Console.WriteLine("Wprowadź nowego dostawcę");
        string supplierName = Console.ReadLine();

        // Entering new products
        Console.WriteLine("Wrowadź kilka 4 produkty");
        string[] products = new string[4];
        for (int i = 0; i < 4; i++)
        {
            products[i] = Console.ReadLine();
        }
    }
}

```

```

        Supplier supplier = new Supplier { CompanyName =
supplierName };
        productContext.Suppliers.Add(supplier);
        productContext.SaveChanges();

        Product product1 = new Product { ProductName =
products[0].ToString() };
        Product product2 = new Product { ProductName =
products[1].ToString() };
        Product product3 = new Product { ProductName =
products[2].ToString() };
        Product product4 = new Product { ProductName =
products[3].ToString() };

        productContext.Products.Add(product1);
        productContext.Products.Add(product2);
        productContext.Products.Add(product3);
        productContext.Products.Add(product4);
        productContext.SaveChanges();

        // Adding products to their suppliers and suppliers to those
products
        productContext.Suppliers.Include(s => s.Products).ToList();
        Product[] prods = { product1, product2, product3, product4
};

        supplier.Products.AddRange(prods);
        productContext.SaveChanges();

        foreach(Product p in prods)
        {
            productContext.Entry(p).Reference(prod =>
prod.Supplier).Load();
            if (p.Supplier == null)
            {

productContext.Entry(p).Property("SupplierID").CurrentValue =
supplier.SupplierID;
            productContext.SaveChanges();
            }
        }

        // Creating Invoices
        Invoice invoice1 = new Invoice { InvoiceNumber = 1, Quantity
= 3 };
        Invoice invoice2 = new Invoice { InvoiceNumber = 2, Quantity
= 3 };

```

```

// Adding invoices to productContext
productContext.Invoices.Add(invoice1);
productContext.Invoices.Add(invoice2);
productContext.SaveChanges();

// Creating invoiceProducts
InvoiceProduct invoiceProduct1 = new InvoiceProduct {
Invoice = invoice1, Product = product1 };
InvoiceProduct invoiceProduct2 = new InvoiceProduct {
Invoice = invoice1, Product = product2 };
InvoiceProduct invoiceProduct3 = new InvoiceProduct {
Invoice = invoice1, Product = product3 };
InvoiceProduct invoiceProduct4 = new InvoiceProduct {
Invoice = invoice1, Product = product4 };
InvoiceProduct invoiceProduct5 = new InvoiceProduct {
Invoice = invoice2, Product = product1 };
InvoiceProduct invoiceProduct6 = new InvoiceProduct {
Invoice = invoice2, Product = product3 };

// Adding invoiceProducts to productContext
productContext.InvoiceProducts.Add(invoiceProduct1);
productContext.InvoiceProducts.Add(invoiceProduct2);
productContext.InvoiceProducts.Add(invoiceProduct3);
productContext.InvoiceProducts.Add(invoiceProduct4);
productContext.InvoiceProducts.Add(invoiceProduct5);
productContext.InvoiceProducts.Add(invoiceProduct6);

// Adding invoiceProducts to products and to invoices
product1.InvoiceProducts.Add(invoiceProduct1);
product2.InvoiceProducts.Add(invoiceProduct2);
product3.InvoiceProducts.Add(invoiceProduct3);
product4.InvoiceProducts.Add(invoiceProduct4);
product1.InvoiceProducts.Add(invoiceProduct5);
product3.InvoiceProducts.Add(invoiceProduct6);

invoice1.InvoiceProducts.Add(invoiceProduct1);
invoice1.InvoiceProducts.Add(invoiceProduct2);
invoice1.InvoiceProducts.Add(invoiceProduct3);
invoice1.InvoiceProducts.Add(invoiceProduct4);
invoice2.InvoiceProducts.Add(invoiceProduct5);
invoice2.InvoiceProducts.Add(invoiceProduct6);

productContext.SaveChanges();

// Additional product printing

```

```

Console.WriteLine("PRODUCT LIST:");

var prodList = from supp in productContext.Suppliers
               where supp.CompanyName == supplierName
               select supp.Products;

foreach(var p in prodList.FirstOrDefault())
{
    Console.WriteLine("Product name: " +
p.ProductName.ToString() + " is supplied by: "
+ p.Supplier.CompanyName.ToString());
}

productContext.SaveChanges();
}
}

```

Konsola debugowania programu Microsoft Visual Studio

```

Wprowadź nowego dostawcę
Food Delivery
Wprowadź kilka 4 produkty
milk
coca cola
bread
butter
PRODUCT LIST:
Product name: milk is supplied by: Food Delivery
Product name: coca cola is supplied by: Food Delivery
Product name: bread is supplied by: Food Delivery
Product name: butter is supplied by: Food Delivery

```

Konsola debugowania programu - “w tle” działania programu produkty te zostały “sprzedane”, tj. stworzone zostały relacje produkt-faktura, a następnie relacje te zostały dodane do zbiorów relacji produkt-faktura w dot. je fakturach oraz produktach.

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	milk	1	0
2	2	coca cola	1	0
3	3	bread	1	0
4	4	butter	1	0

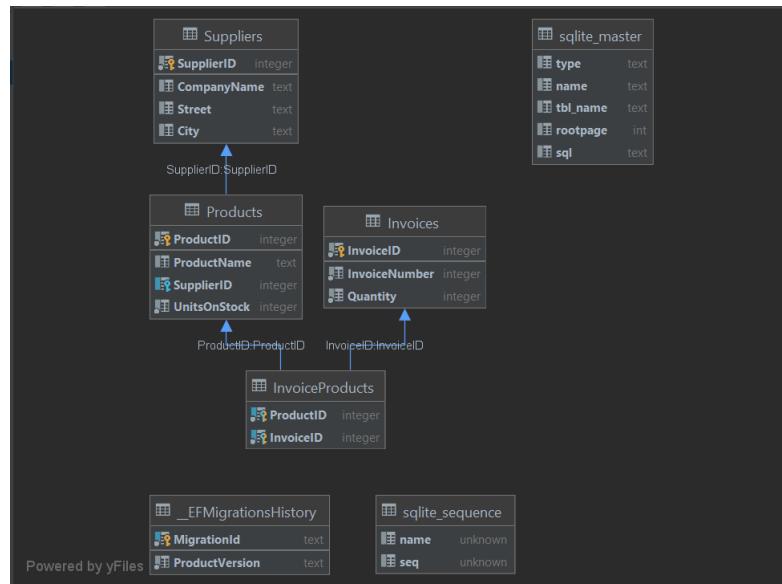
Dodane produkty, widoczne w tabeli Products,

	SupplierID	CompanyName	Street	City
1	1	Food Delivery	<null>	<null>

Dodany dostawca, widoczny w tabeli Suppliers,

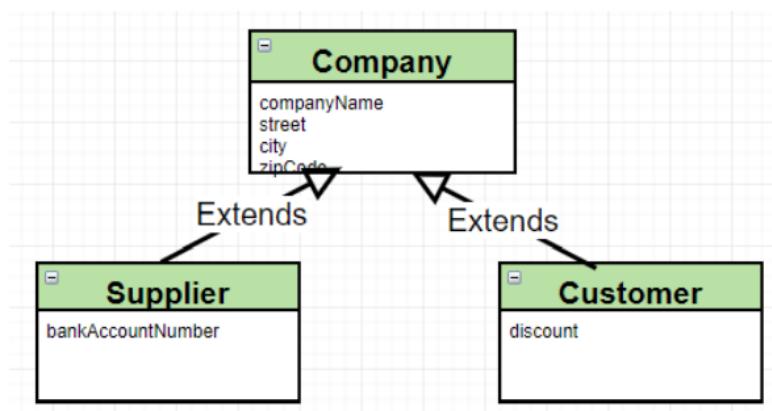
ProductID	InvoiceID
1	4
2	3
3	2
4	1
5	1
6	3

“Sprzedaż” produktów - dodane relacje Product-Invoice widoczne w tabeli InvoiceProducts,



Schemat bazy danych.

5. **Zadanie V:** Wprowadzono do modelu poniższą hierarchię dziedziczenia, używając strategii *Table-Per-Hierarchy*:



### 5.1. Dodano klasę Company

```
class Company
{
```

```
public int CompanyID { get; set; }
public string CompanyName { get; set; }
public string Street { get; set; }
public string City { get; set; }
public string ZipCode { get; set; }
}
```

## 5.2. Dodano klasę Customer, dziedziczącą z Company

```
class Customer:Company
{
    public float Discount { get; set; }
}
```

## 5.3. Zmieniono strukturę klasy Supplier, by dziedziczyła z Company

```
class Supplier:Company
{
    public Supplier()
    {
        Products = new List<Product>();
    }
    public string BankAccountNumber { get; set; }
    public virtual List<Product> Products { get; set; }
}
```

## 5.4. Zmieniono strukturę klasy ProductContext, zmieniona została metoda OnModelCreating - dodano metodę modelBuilder.Entity< TEntity > (dla typu encji Customer oraz Supplier), zwracająca obiekt, który następnie może zostać użyty do konfigurowania danej encji w modelu, ten typ encji zostaje dodany do modelu.

```
class ProductContext:DbContext
{
    public DbSet<Product> Products { get; set; }
    public DbSet<Company> Companies { get; set; }
    public DbSet<Invoice> Invoices { get; set; }
    public DbSet<InvoiceProduct> InvoiceProducts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("Datasource=ProductsDatabase.db");
    }
}
```

```

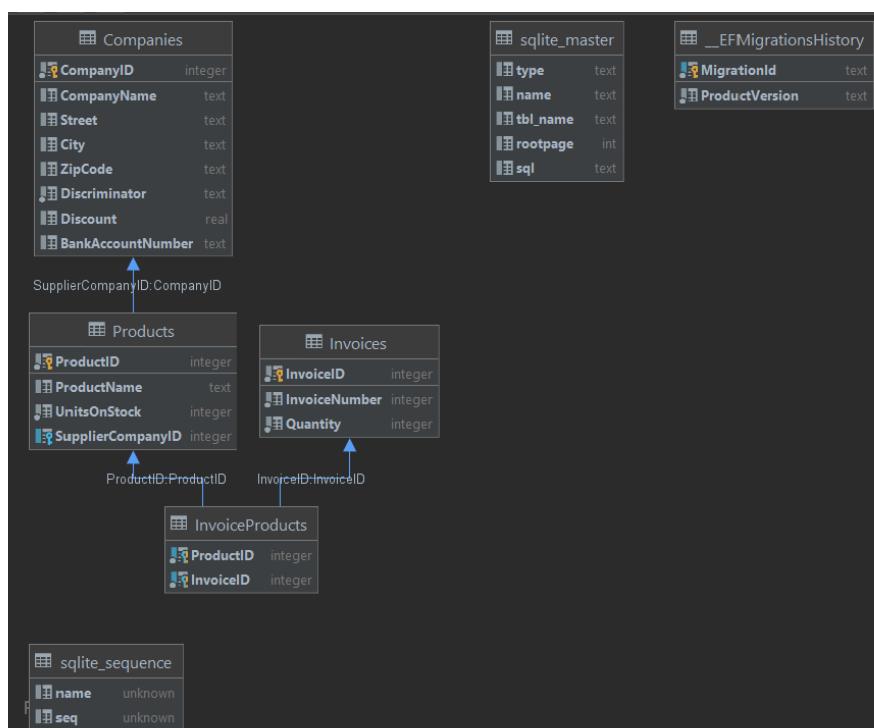
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<InvoiceProduct>()
            .HasKey(ip => new { ip.ProductID, ip.InvoiceID });
        modelBuilder.Entity<Customer>();
        modelBuilder.Entity<Supplier>();
    }

}

```

### 5.5. Schemat bazy danych przyjął postać:



### 5.6. Dodano do bazy dwóch dostawców oraz dwóch klientów

```

class Program
{
    static void Main(string[] args)
    {
        ProductContext productContext = new ProductContext();

        Supplier supplier1 = new Supplier
        {
            CompanyName = "Supplier1",
            City = "Cracow",

```

```

        Street = "Kawiory",
        ZipCode = "11-111",
        BankAccountNumber = "111111",
    };

    productContext.Add(supplier1);

    Supplier supplier2 = new Supplier
    {
        CompanyName = "Supplier2",
        City = "Warsaw",
        Street = "Marszałkowska",
        ZipCode = "22-222",
        BankAccountNumber = "222222",
    };

    productContext.Add(supplier2);

    Customer customer1 = new Customer
    {
        CompanyName = "Customer1",
        City = "London",
        Street = "Pickadilly",
        ZipCode = "33-333",
        Discount = 0.2f,
    };

    productContext.Add(customer1);

    Customer customer2 = new Customer
    {
        CompanyName = "Customer2",
        City = "Athens",
        Street = "Akropolis",
        ZipCode = "44-444",
        Discount = 0.3f,
    };

    productContext.Add(customer2);

    productContext.SaveChanges();

}

}

```

## 5.7. Pobrano z bazy dane dotyczące firm obu rodzajów

The screenshot shows the EntityDataSource interface in Visual Studio. A query 'select \* from Companies;' is run, resulting in 4 rows of data. The columns are CompanyID, CompanyName, Street, City, ZipCode, Discriminator, Discount, and BankAccountNumber. The data is as follows:

	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	Discount	BankAccountNumber
1	Supplier1	Kawiory	Cracow	11-111	Supplier	<null>	111111	
2	Supplier2	Marszałkowska	Warsaw	22-222	Supplier	<null>	222222	
3	Customer1	Pickadilly	London	33-333	Customer	0.20000000298023224	<null>	
4	Customer2	Akropolis	Athens	44-444	Customer	0.30000001192092896	<null>	

## 6. Zadanie VI: model j.w. jednak tym razem oparty na *strategii Table-Per-type*

### 6.1. Zmodyfikowano klasę Customer

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;

namespace BarbaraWojtarowiczEFProducts
{
    [Table("Customers")]
    class Customer: Company
    {
        public float Discount { get; set; }
    }
}
```

### 6.2. Zmodyfikowano klasę Supplier

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;

namespace BarbaraWojtarowiczEFProducts
{
    [Table ("Suppliers")]
    class Supplier:Company
    {
        public Supplier()
        {
            Products = new List<Product>();
        }

        public string BankAccountNumber { get; set; }
        public virtual List<Product> Products { get; set; }
    }
}
```

- 6.3. Zmodyfikowano klasę ProductContext, dziedziczącą z DbContext. Usunięto metody modelBuilder.Entity< TEntity>()

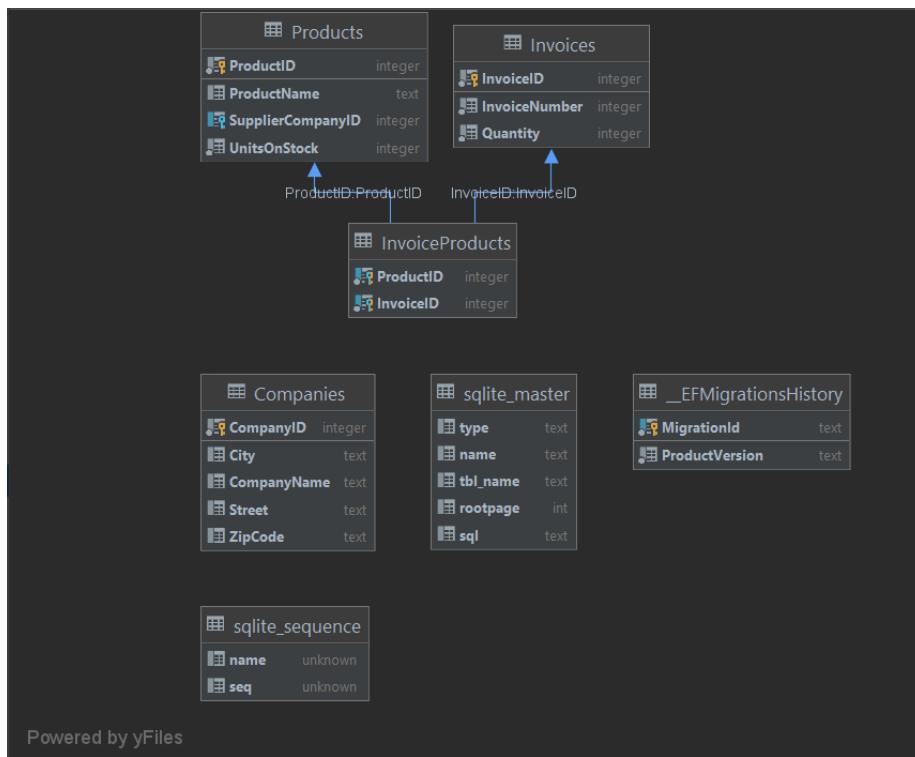
```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.EntityFrameworkCore;

namespace BarbaraWojtarowiczEFProducts
{
    class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Company> Companies { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<InvoiceProduct> InvoiceProducts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            base.OnConfiguring(optionsBuilder);
            optionsBuilder.UseSqlite("Datasource=ProductsDatabase.db");
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<InvoiceProduct>()
                .HasKey(ip => new { ip.ProductID, ip.InvoiceID });
        }
    }
}
```

- 6.4. W wyniku w/w modyfikacji, schemat bazy uległ zmianie:



Można zauważyć, że z tabeli Companies zniknęły pola:

- Discount,
- BankAccountNumber,
- Discriminator

6.5. Napisano program dodający do bazy dwóch dostawców, dwóch klientów, produkt dostarczany przez dostawcę, a także pobierający i wypisujący dane o w/w klientach i dostawcach:

```

using System;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using System.Collections.Generic;
using System.Text;

namespace BarbaraWojtarowiczEFProducts
{
    class Program
    {
        static void Main(string[] args)
        {
            ProductContext productContext = new ProductContext();
            Console.WriteLine("Table-Per-Type Strategy");

            // Adding new suppliers

```

```

Supplier supplier1 = new Supplier
{
    CompanyName = "Supplier1",
    City = "Cracow",
    Street = "Kawiory",
    ZipCode = "11-111",
    BankAccountNumber = "111111",
};

productContext.Add(supplier1);

Supplier supplier2 = new Supplier
{
    CompanyName = "Supplier2",
    City = "Warsaw",
    Street = "Marszałkowska",
    ZipCode = "22-222",
    BankAccountNumber = "222222",
};

productContext.Add(supplier2);

// Adding new customers
Customer customer1 = new Customer
{
    CompanyName = "Customer1",
    City = "London",
    Street = "Pickadilly",
    ZipCode = "33-333",
    Discount = 0.2f,
};

productContext.Add(customer1);

Customer customer2 = new Customer
{
    CompanyName = "Customer2",
    City = "Athens",
    Street = "Akropolis",
    ZipCode = "44-444",
    Discount = 0.3f,
};

productContext.Add(customer2);

Product product1 = new Product

```

```

    {
        ProductName = "Oranges",
        Supplier = supplier1,
    };

    productContext.Products.Add(product1);
    productContext.SaveChanges();

    // Printing newly added suppliers and customers
    var suppliers =
productContext.Companies.OfType<Supplier>().ToList();
    var customers =
productContext.Companies.OfType<Customer>().ToList();

    Console.WriteLine("SUPPLIERS: ");
    foreach (var supp in suppliers)
    {
        Console.WriteLine("Supplier: " + supp.CompanyName +
"with bank account number: " + supp.BankAccountNumber);
    }

    Console.WriteLine("CUSTOMERS: ");
    foreach (var cus in customers)
    {
        Console.WriteLine("Customer: " + cus.CompanyName +
"with discount value: " + cus.Discount);
    }

}
}
}
}

```

Program rzucił wyjątek:

```

// Printing newly added suppliers and customers
var suppliers = productContext.Companies.OfType<Supplier>().ToList();
var customers = productContext.Companies.OfType<Customer>().ToList(); ✖

Console.WriteLine("SUPPLIERS: ");
foreach (var supp in suppliers)
{
    Console.WriteLine("Supplier: " + supp.CompanyName + "with bank acco")
}

Console.WriteLine("CUSTOMERS: ");
foreach (var cus in customers)
{
    Console.WriteLine("Customer: " + cus.CompanyName + "with discount")
}

```

Wprowadzono zmianę we fragmencie kodu zawartym na powyższym zrzucie ekranu.  
Zmiana dotyczy pobierania danych:

```

// Printing newly added suppliers and customers
var suppliers = productContext.Companies.OfType<Supplier>();
var customers = productContext.Companies.OfType<Customer>();

Console.WriteLine("SUPPLIERS: ");
foreach (var supp in suppliers)
{
    Console.WriteLine("Suplier: " + supp.CompanyName +
"with bank account number: " + supp.BankAccountNumber);
}

Console.WriteLine("CUSTOMERS: ");
foreach(var cus in customers)
{
    Console.WriteLine("Customer: " + cus.CompanyName +
"with discount value: " + cus.Discount);
}

```

Program prawidłowo pobrał i wypisał dane z tabeli Suppliers:

```

C:\Users\basia\Desktop\BarbaraWojtarowiczEFProducts\BarbaraWojtarowicz
Table-Per-Type Strategy
SUPPLIERS:
Suplier: Supplier1with bank account number: 111111
Suplier: Supplier2with bank account number: 222222
Suplier: Supplier1with bank account number: 111111
Suplier: Supplier2with bank account number: 222222
Suplier: Supplier1with bank account number: 111111
Suplier: Supplier2with bank account number: 222222
CUSTOMERS:
-
```

Jednak rzucił wyjątek przy próbie dot. tabeli Customers:

```
// Printing newly added suppliers
var suppliers = productContext
var customers = productContext

Console.WriteLine("SUPPLIERS:");
foreach (var supp in suppliers)
{
    Console.WriteLine("Supplier: " + supp.CompanyName);
}

Console.WriteLine("CUSTOMERS:");
foreach (var cus in customers)
{
    Console.WriteLine("Customer: " + cus.CompanyName + " with discount value: " + cus.Discount);
}
```

Nieobsługiwany wyjątek

**System.InvalidOperationException:** „The LINQ expression 'DbSet<Company>().OfType<Customer>()' could not be translated. Either rewrite the query in a form that can be translated, or switch to client evaluation explicitly by inserting a call to 'AsEnumerable', 'AsAsyncEnumerable', 'ToList', or 'ToListAsync'. See https://go.microsoft.com/fwlink/?linkid=2101038 for more information.”

Wyświetl szczegóły | Kopij szczegóły | Rozpocznij sesję rozszerzenia Live Share  
▼ Ustawienia wyjątków

AccountNumber);

6.6. Wykonano następujące zapytania:

```
select * from Suppliers inner join Products P on Suppliers.CompanyID = P.SupplierCompanyID
```

	CompanyID	BankAccountNumber	ProductID	ProductName	SupplierCompanyID	UnitsOnStock
1		1 111111	1	Oranges	1	0

```
select * from Companies
```

	CompanyID	City	CompanyName	Street	ZipCode
1		1 Cracow	Supplier1	Kawiory	11-111
2		2 Warsaw	Supplier2	Marszałkowska	22-222
3		3 London	Customer1	Pickadilly	33-333
4		4 Athens	Customer2	Akropolis	44-444

```
select * from Suppliers
```

	CompanyID	BankAccountNumber
1		1 111111
2		2 222222

Z kolejnym zapytaniem:

```
select * from Customers
```

zwróciło błąd:

```
[1] [SQLITE_ERROR] SQL error or missing database (no such table: Customers)
```

Wynika z tego, że tabela Customers nie istnieje. Dostawcy figurują zarówno w tabeli Companies, jak i w tabeli Suppliers, skąd można pobrać numery ich kont. Klienci zaś figurują

jedynie w tabeli Companies.

Szukając przyczyny tej sytuacji, na stronie <https://docs.microsoft.com/pl-pl/ef/efcore-and-ef6/> znalazłem następujące porównanie EF Core z EF 6.4:

## Tworzenie modelu

Funkcja	EF 6.4	EF Core
Podstawowe mapowanie klas	Tak	1,0
Konstruktory z parametrami		2.1
Konwersje wartości właściwości		2.1
Mapowane typy bez kluczy		2.1
Konwencje	Tak	1,0
Konwencje niestandardowe	Tak	1,0 (częściowa; #214 ↗)
Adnotacje do danych	Tak	1,0
Interfejs API Fluent	Tak	1,0
Dziedziczenie: tabela na hierarchię (TPH)	Tak	1,0
Dziedziczenie: tabela na typ (TPT)	Tak	Planowane dla 5.0 (#2266 ↗)

Być może wersja Entity Framework, którą dysponuję (5.0.4), nie obsługuje poprawnie dziedziczenia tabel na typ, być może jednak popełniłam błąd tworząc program.