README.MD

Entity Framework

Author: Uladzislau Tumilovich

Sprawozdanie

Zadanie 1

Stworzyłem projekt lTumilovichProductEF i dodałem klasę Product z polami int ProductID, string Name, int UnitsInStock

Product class

```
class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
}
```

Dalej stworzyłem klasę ProdContext dziedziczącą po DbContext.

ProdContext class

```
class ProdContext:DbContext
{
    public DbSet<Product> Products { set; get; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
        optionsBuilder.UseSqlite("DataSource=Product.db");
}
```

Następnie w klasie Program dodałem metodę AddProduct, która dodaje produkt do bazy i metodę PrintProductNames, która wyświetla nazwy produktów z bazy. Na koniec wyzywam tę metody w metodzie Main.

Program class

```
class Program
{
    private static void AddProduct(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product:");
        String prodName = Console.ReadLine();

        Product product = new Product();
        product.Name = prodName;

        prodContext.Products.Add(product);
        prodContext.SaveChanges();
    }

    private static void PrintProductsNames(ProdContext prodContext)
    {
        Console.WriteLine("List of products names from database:");
        var query = from p in prodContext.Products
                    select p.Name;

        foreach (var item in query)
        {
            Console.WriteLine(item);
        }
```

```
    }

    static void Main(string[] args)
    {
        ProdContext prodContext = new ProdContext();

        AddProduct(prodContext);
        PrintProductsNames(prodContext);
    }
}
```

Wynik działania programu

```
Input name of the product:
Chair
List of products from database
Mouse
Keyboard
Chair
```

Wygląd tabeli Products

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ProductID	INTEGER							NULL
2	Name	TEXT							NULL
3	UnitsInStock	INTEGER							NULL

Select z tabeli Product

Structure | Data | Constraints | Indexes | Triggers | DDL

Grid view | Form view

	ProductID	Name	UnitsInStock
1	1	Mouse	0
2	2	Keyboard	0
3	3	Chair	0

Zadanie 2

W drugim punkcie zmodyfikowałem kod z pierwszego punktu dodaniem klasy Supplier

Supplier class

```
class Supplier
{
    public int SupplierID { get; set; }
    public String CompanyName { get; set; }
    public String Street { get; set; }
    public String City { get; set; }
}
```

Zmodyfikowałem klasę Product dodaniem klucza obcego Supplier

Product class

```
class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
    public Supplier Supplier { get; set; }
}
```

W klasie ProdContext dodałem set Supplierów

ProdContext class

```
class ProdContext:DbContext
{
    public DbSet<Product> Products { set; get; }
    public DbSet<Supplier> Suppliers { set; get; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
        optionsBuilder.UseSqlite("DataSource=Product.db");
}
```

Na koniec dodałem metody AddSupplier, PrintSuppliersCompaniesNames, ConnectProductSupplier, PrintProductsWithSuppliers i zmodyfikowałem metodę Main klasy Program

Program class

```
class Program
{
    private static void AddProduct(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product:");
        String prodName = Console.ReadLine();

        Product product = new Product();
        product.Name = prodName;

        prodContext.Products.Add(product);
        prodContext.SaveChanges();
    }

    private static void AddSupplier(ProdContext prodContext)
    {
        Console.WriteLine("Input company name of supplier:");
        String companyName = Console.ReadLine();

        Supplier supplier = new Supplier();
        supplier.CompanyName = companyName;

        prodContext.Suppliers.Add(supplier);
        prodContext.SaveChanges();
    }

    private static void ConnectProductSupplier(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product");
        String prodName = Console.ReadLine();
        Product product = prodContext.Products.Where(p => p.Name == prodName).FirstOrDefault();

        Console.WriteLine("Input company name of supplier:");
        String companyName = Console.ReadLine();
        Supplier supplier = prodContext.Suppliers.Where(s => s.CompanyName == companyName).FirstOrDefault();

        product.Supplier = supplier;
        prodContext.SaveChanges();
    }

    public static void PrintProductsWithSuppliers(ProdContext prodContext)
    {
        Console.WriteLine("List of products with suppliers from database:");

        foreach (Product item in prodContext.Products)
        {
            prodContext.Entry(item).Reference(prod => prod.Supplier).Load();

            if (item.Supplier != null)
            {
                Console.WriteLine(item.Name + " " + item.Supplier.CompanyName);
            }
            else
```

```
            {
                Console.WriteLine(item.Name);
            }
        }
    }

    private static void PrintProductsNames(ProdContext prodContext)
    {
        Console.WriteLine("List of products names from database:");
        var query = from p in prodContext.Products
                    select p.Name;

        foreach (var item in query)
        {
            Console.WriteLine(item);
        }
    }

    private static void PrintSuppliersCompaniesNames(ProdContext prodContext)
    {
        Console.WriteLine("List of suppliers companies names from database:");
        var query = from s in prodContext.Suppliers
                    select s.CompanyName;

        foreach (var item in query)
        {
            Console.WriteLine(item);
        }
    }

    static void Main(string[] args)
    {
        ProdContext prodContext = new ProdContext();

        AddProduct(prodContext);
        PrintProductsNames(prodContext);

        AddSupplier(prodContext);
        PrintSuppliersCompaniesNames(prodContext);

        PrintProductsWithSuppliers(prodContext);

        ConnectProductSupplier(prodContext);

        PrintProductsWithSuppliers(prodContext);
    }
}
```

Wynik działania programu

```
Input name of product
Bread
Input company name of supplier:
Breadfactory
List of products with suppliers from database:
Milk MilkCompany
Water
Plate
Bread Breadfactory
```

Wygląd tabeli Products

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ProductID	INTEGER							NULL
2	Name	TEXT							NULL
3	UnitsInStock	INTEGER							NULL
4	SupplierID	INTEGER							NULL

Wygląd tabeli Suppliers

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	SupplierID	INTEGER							NULL
2	CompanyName	TEXT							NULL
3	Street	TEXT							NULL
4	City	TEXT							NULL

Select z tabeli Product

Structure | Data | Constraints | Indexes | Triggers

Grid view | Form view

	ProductID	Name	UnitsInStock	SupplierID
1	1	Milk	0	1
2	2	Water	0	NULL
3	3	Plate	0	NULL
4	4	Bread	0	4

Select z tabeli Suppliers

Structure | Data | Constraints | Indexes | Triggers | DDL

Grid view | Form view

	SupplierID	CompanyName	Street	City
1	1	MilkCompany	NULL	NULL
2	2	WaterCompany	NULL	NULL
3	3	GlassFactory	NULL	NULL
4	4	Breadfactory	NULL	NULL

Zadanie 3

W trzecim punkcie odwróciłem relacje tabel Product i Suppliers przez usunięcie pola Supplier z klasy Products i dodaniem listy Produktów w klasie Suppliers

Product class

```
class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
}
```

Supplier class

```
class Supplier
{
    public Supplier()
    {
        Products = new List<Product>();
    }

    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; }
}
```

Na koniec dodałem do klasy Program metodę PrintSuppliersWithProducts oraz zmodyfikowałem metody ConnectProductSupplier i Main.

Program class

```
class Program
{
    private static void AddProduct(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product:");
        String prodName = Console.ReadLine();

        Product product = new Product();
        product.Name = prodName;

        prodContext.Products.Add(product);
        prodContext.SaveChanges();
    }

    private static void AddSupplier(ProdContext prodContext)
    {
        Console.WriteLine("Input company name of supplier:");
        String companyName = Console.ReadLine();

        Supplier supplier = new Supplier();
        supplier.CompanyName = companyName;

        prodContext.Suppliers.Add(supplier);
        prodContext.SaveChanges();
    }

    private static void ConnectProductSupplier(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product");
        String prodName = Console.ReadLine();
        Product product = prodContext.Products.Where(p => p.Name == prodName).FirstOrDefault();

        Console.WriteLine("Input company name of supplier:");
        String companyName = Console.ReadLine();
        Supplier supplier = prodContext.Suppliers.Where(s => s.CompanyName == companyName).FirstOrDefault();

        supplier.Products.Add(product);
        prodContext.SaveChanges();
    }

    public static void PrintSuppliersWithProducts(ProdContext prodContext)
    {
        Console.WriteLine("List of suppliers with products from database:");

        var data = prodContext.Suppliers.Include(s => s.Products).ToList();
        foreach (var s in data)
        {
            Console.WriteLine("Supplier: " + s.CompanyName);
            foreach (var p in s.Products)
            {
                Console.WriteLine(p.Name);
            }
        }
    }

    static void Main(string[] args)
    {
        ProdContext prodContext = new ProdContext();

        AddProduct(prodContext);
        AddSupplier(prodContext);

        ConnectProductSupplier(prodContext);

        PrintSuppliersWithProducts(prodContext);
    }
}
```

Wynik działania programu

```
Input name of product:
Juice
Input company name of supplier:
JuiceFactory
Input name of product
Juice
Input company name of supplier:
JuiceFactory
List of suppliers with products from database:
Supplier: MilkCompany
Milk
Butter
Supplier: WaterCompany
Supplier: JustCompany
Ice-cream
Supplier: JuiceFactory
Juice
```

Wygląd tabeli Products

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ProductID	INTEGER							NULL
2	Name	TEXT							NULL
3	UnitsInStock	INTEGER							NULL
4	SupplierID	INTEGER							NULL

Wygląd tabeli Suppliers

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	SupplierID	INTEGER							NULL
2	CompanyName	TEXT							NULL
3	Street	TEXT							NULL
4	City	TEXT							NULL

Select z tabeli Product

	ProductID	Name	UnitsInStock	SupplierID
1	1	Milk	0	1
2	2	Water	0	NULL
3	3	Butter	0	1
4	4	Ice-cream	0	3
5	5	Juice	0	4

Select z tabeli Suppliers

	SupplierID	CompanyName	Street	City
1	1	MilkCompany	NULL	NULL
2	2	WaterCompany	NULL	NULL
3	3	JustCompany	NULL	NULL
4	4	JuiceFactory	NULL	NULL

Zadanie 4

W czwartym punkcie dla otrzymania relacji dwustronnej dodałem pole Supplier w klasie Product

Product class

```
class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
```

```
    public Supplier Supplier { get; set; }
}
```

Następnie zmodyfikowałem klasę Program dla wizualizacji programu

Program class

```
class Program
{
    private static void AddProduct(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product:");
        String prodName = Console.ReadLine();

        Product product = new Product();
        product.Name = prodName;

        prodContext.Products.Add(product);
        prodContext.SaveChanges();
    }

    private static void AddSupplier(ProdContext prodContext)
    {
        Console.WriteLine("Input company name of supplier:");
        String companyName = Console.ReadLine();

        Supplier supplier = new Supplier();
        supplier.CompanyName = companyName;

        prodContext.Suppliers.Add(supplier);
        prodContext.SaveChanges();
    }

    private static void ConnectProductSupplier(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product");
        String prodName = Console.ReadLine();
        Product product = prodContext.Products.Where(p => p.Name == prodName).FirstOrDefault();

        Console.WriteLine("Input company name of supplier:");
        String companyName = Console.ReadLine();
        Supplier supplier = prodContext.Suppliers.Where(s => s.CompanyName == companyName).FirstOrDefault();

        supplier.Products.Add(product);
        product.Supplier = supplier;
        prodContext.SaveChanges();
    }

    public static void PrintSuppliersWithProducts(ProdContext prodContext)
    {
        Console.WriteLine("List of suppliers with products from database:");

        var data = prodContext.Suppliers.Include(s => s.Products).ToList();
        foreach (var s in data)
        {
            Console.WriteLine("Supplier: " + s.CompanyName);
            foreach (var p in s.Products)
            {
                Console.WriteLine(p.Name);
            }
        }
    }

    public static void PrintProductsWithSuppliers(ProdContext prodContext)
    {
        Console.WriteLine("List of products with suppliers from database:");

        foreach (Product item in prodContext.Products)
        {
            prodContext.Entry(item).Reference(prod => prod.Supplier).Load();
```

```
            if (item.Supplier != null)
            {
                Console.WriteLine(item.Name + " " + item.Supplier.CompanyName);
            }
            else
            {
                Console.WriteLine(item.Name);
            }
        }
    }

    static void Main(string[] args)
    {
        ProdContext prodContext = new ProdContext();

        AddProduct(prodContext);
        AddSupplier(prodContext);

        ConnectProductSupplier(prodContext);

        PrintSuppliersWithProducts(prodContext);
        PrintProductsWithSuppliers(prodContext);
    }
}
```

Wynik działania programu

```
Input name of product:
Laptop
Input company name of supplier:
LaptopShop
Input name of product
Laptop
Input company name of supplier:
LaptopShop
List of suppliers with products from database:
Supplier: MilkCompany
Milk
Butter
Supplier: WaterCompany
Supplier: JustCompany
Ice-cream
Supplier: JuiceFactory
Juice
Supplier: ElectronicsCompany
Lamp
Supplier: LaptopShop
Laptop
List of products with suppliers from database:
Milk MilkCompany
Water
Butter MilkCompany
Ice-cream JustCompany
Juice JuiceFactory
Lamp ElectronicsCompany
Laptop LaptopShop
```

Wygląd tabeli Products

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ProductID	INTEGER							*NULL*
2	Name	TEXT							*NULL*
3	UnitsInStock	INTEGER							*NULL*
4	SupplierID	INTEGER							*NULL*

Wygląd tabeli Suppliers

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	SupplierID	INTEGER							NULL
2	CompanyName	TEXT							NULL
3	Street	TEXT							NULL
4	City	TEXT							NULL

Select z tabeli Product

	ProductID	Name	UnitsInStock	SupplierID
1	1	Milk	0	1
2	2	Water	0	NULL
3	3	Butter	0	1
4	4	Ice-cream	0	3
5	5	Juice	0	4

Select z tabeli Suppliers

	SupplierID	CompanyName	Street	City
1	1	MilkCompany	NULL	NULL
2	2	WaterCompany	NULL	NULL
3	3	JustCompany	NULL	NULL
4	4	JuiceFactory	NULL	NULL

Zadanie 5

W piątym punkcie dodałem klasę Category z polami CategoryID, Name, oraz listą Produktów

Category class

```
class Category
{
    public Category()
    {
        Products = new List<Product>();
    }
    public int CategoryID { get; set; }
    public String Name { get; set; }
    public List<Product> Products { get; set; }
}
```

Następnie dodałem set Categories w klasie ProdContext

ProdContext class

```
class ProdContext:DbContext
{
    public DbSet<Product> Products { set; get; }
    public DbSet<Supplier> Suppliers { set; get; }
    public DbSet<Category> Categories { set; get; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
        optionsBuilder.UseSqlite("DataSource=Product.db");
}
```

Zmodyfikowałem klasę Product przez dodanie pola Category

Product class

```
class Product
{
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
```

```
    public Supplier Supplier { get; set; }
    public Category Category { get; set; }
}
```

I na koniec do klasy Program dodałem pomocnicze metody - AddCategory, ConnectProductCategory, PrintCategoriesWithProducts, PrintProductsWithCategories oraz zmodyfikowałem metodę Main dla uruchomienia programu

Program class

```
class Program
{
    private static void AddProduct(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product:");
        String prodName = Console.ReadLine();

        Product product = new Product();
        product.Name = prodName;

        prodContext.Products.Add(product);
        prodContext.SaveChanges();
    }

    private static void AddCategory(ProdContext prodContext)
    {
        Console.WriteLine("Input name of Category:");
        String categoryName = Console.ReadLine();

        Category category = new Category();
        category.Name = categoryName;

        prodContext.Categories.Add(category);
        prodContext.SaveChanges();
    }

    private static void ConnectProductCategory(ProdContext prodContext)
    {
        Console.WriteLine("Input name of product");
        String prodName = Console.ReadLine();
        Product product = prodContext.Products.Where(p => p.Name == prodName).FirstOrDefault();

        Console.WriteLine("Input name of category:");
        String categoryName = Console.ReadLine();
        Category category = prodContext.Categories.Where(c => c.Name == categoryName).FirstOrDefault();

        category.Products.Add(product);
        product.Category = category;
        prodContext.SaveChanges();
    }

    private static void PrintCategoriesWithProducts(ProdContext prodContext)
    {
        Console.WriteLine("List of categories with products from database:");

        var data = prodContext.Categories.Include(c => c.Products).ToList();
        foreach (var c in data)
        {
            Console.WriteLine("Category: " + c.Name);
            foreach (var p in c.Products)
            {
                Console.WriteLine("Product: " + p.Name);
            }
        }
    }

    private static void PrintProductsWithCategories(ProdContext prodContext)
    {
        Console.WriteLine("List of products with categories from database:");

        foreach (Product item in prodContext.Products)
```

```
            {
                prodContext.Entry(item).Reference(prod => prod.Category).Load();

                if (item.Category != null)
                {
                    Console.WriteLine("Product: " + item.Name + " Category: " + item.Category.Name);
                }
                else
                {
                    Console.WriteLine("Product: " + item.Name);
                }
            }
        }

        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();

            for (int i = 0; i < 5; i++)
            {
                AddProduct(prodContext);
            }

            for (int i = 0; i < 2; i++)
            {
                AddCategory(prodContext);
            }

            for (int i = 0; i < 5; i++)
            {
                ConnectProductCategory(prodContext);
            }

            PrintCategoriesWithProducts(prodContext);
            PrintProductsWithCategories(prodContext);
        }
    }
```

Wynik działania programu

```
Input name of product
TV
Input name of category:
Electronics
Input name of product
Laptop
Input name of category:
Electronics
Input name of product
Milk
Input name of category:
Food
Input name of product
Pizza
Input name of category:
Food
List of categories with products from database:
Category: Food
Product: Milk
Product: Pizza
Category: Electronics
Product: Lamp
Product: TV
Product: Laptop
List of products with categories from database:
Product: Lamp Category: Electronics
Product: TV Category: Electronics
Product: Laptop Category: Electronics
Product: Milk Category: Food
Product: Pizza Category: Food
```

Wygląd tabeli Products

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ProductID	INTEGER							*NULL*
2	Name	TEXT							*NULL*
3	UnitsInStock	INTEGER							*NULL*
4	SupplierID	INTEGER							*NULL*
5	CategoryID	INTEGER							*NULL*

Wygląd tabeli Categories

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	CategoryID	INTEGER							*NULL*
2	Name	TEXT							*NULL*

Select z tabeli Product

	ProductID	Name	UnitsInStock	SupplierID	CategoryID
1	1	Lamp	0	*NULL*	2
2	2	TV	0	*NULL*	2
3	3	Laptop	0	*NULL*	2
4	4	Milk	0	*NULL*	1
5	5	Pizza	0	*NULL*	1

Select z tabeli Categories

	CategoryID	Name
1	1	Food
2	2	Electronics

Zadanie 6

W szóstym punkcie razem z nową klasą Invoice dodałem pomocniczą klasę InvoiceProduct, która będzie przechowywała relację wielu do wielu tabel Product i Invoice

InvoiceProduct class

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

Invoice class

```
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; }
}
```

Zmodyfikowałem klasę Product przez dodanie listy typu InvoiceProduct

Product class

```
class Product
{
    public Product()
    {
        InvoiceProducts = new List<InvoiceProduct>();
    }
    public int ProductID { get; set; }
    public string Name { get; set; }
    public int UnitsInStock { get; set; }
    public Supplier Supplier { get; set; }
    public Category Category { get; set; }
    public List<InvoiceProduct> InvoiceProducts { get; set; }
}
```

Następnie zmodyfikowałem klasę ProdContext przez dodanie dodatkowych setów Invoices i InvoiceProducts. Dodatkowo nadpisałem metodę OnModelCreating

ProdContext class

```
class ProdContext:DbContext
{
    public DbSet<Product> Products { set; get; }
    public DbSet<Supplier> Suppliers { set; get; }
    public DbSet<Category> Categories { set; get; }
    public DbSet<Invoice> Invoices { set; get; }
    public DbSet<InvoiceProduct> InvoiceProducts { set; get; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
        optionsBuilder.UseSqlite("DataSource=Product.db");

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

I na koniec dodałem pomocnicze metody dla tworzenia i wypisywania obiektów oraz zmodyfikowałem już istniejące pomocnicze metody: AddProduct, AddSupplier, AddCategory, AddInvoice, AddInvoiceProduct, ConnectProductSupplier, ConnectProductCategory, PrintProductsOfInvoice, PrintInvoicesOfProduct. Korzystam z powyższych metod dla uruchomienia i demonstrowania programu w metodzie Main.

Program class

```
class Program
{
    private static void AddProduct(ProdContext prodContext, String prodName)
    {
        Product product = new Product();
        product.Name = prodName;

        prodContext.Products.Add(product);
        prodContext.SaveChanges();
    }

    private static void AddSupplier(ProdContext prodContext, String companyName)
    {
        Supplier supplier = new Supplier();
        supplier.CompanyName = companyName;

        prodContext.Suppliers.Add(supplier);
        prodContext.SaveChanges();
    }

    private static void AddCategory(ProdContext prodContext, String categoryName)
    {
```

```
        Category category = new Category();
        category.Name = categoryName;

        prodContext.Categories.Add(category);
        prodContext.SaveChanges();
    }

    private static void AddInvoice(ProdContext prodContext, int invoiceNumber, int invoiceQuantity)
    {
        Invoice invoice = new Invoice();
        invoice.InvoiceNumber = invoiceNumber;
        invoice.Quantity = invoiceQuantity;

        prodContext.Invoices.Add(invoice);
        prodContext.SaveChanges();
    }

    private static void AddInvoiceProduct(ProdContext prodContext, int invoiceNumber, String prodName)
    {
        Invoice invoice = prodContext.Invoices.Where(i => i.InvoiceNumber == invoiceNumber).FirstOrDefault();
        Product product = prodContext.Products.Where(p => p.Name == prodName).FirstOrDefault();

        InvoiceProduct invoiceProduct = new InvoiceProduct();
        invoiceProduct.Invoice = invoice;
        invoiceProduct.Product = product;

        invoice.InvoiceProducts.Add(invoiceProduct);
        product.InvoiceProducts.Add(invoiceProduct);

        prodContext.InvoiceProducts.Add(invoiceProduct);
        prodContext.SaveChanges();
    }

    private static void ConnectProductSupplier(ProdContext prodContext, String prodName, String companyName)
    {
        Product product = prodContext.Products.Where(p => p.Name == prodName).FirstOrDefault();
        Supplier supplier = prodContext.Suppliers.Where(s => s.CompanyName == companyName).FirstOrDefault();

        supplier.Products.Add(product);
        product.Supplier = supplier;
        prodContext.SaveChanges();
    }

    private static void ConnectProductCategory(ProdContext prodContext, String prodName, String categoryName)
    {
        Product product = prodContext.Products.Where(p => p.Name == prodName).FirstOrDefault();
        Category category = prodContext.Categories.Where(c => c.Name == categoryName).FirstOrDefault();

        category.Products.Add(product);
        product.Category = category;
        prodContext.SaveChanges();
    }

    private static void PrintProductsOfInvoice(ProdContext prodContext, int invoiceNumber)
    {
        Console.WriteLine("List of products of invoice: " + invoiceNumber);

        var products = prodContext.InvoiceProducts
            .Include(ip => ip.Product)
            .Where(ip => ip.Invoice.InvoiceNumber == invoiceNumber)
            .Select(ip => ip.Product.Name).ToList();

        foreach (var p in products)
        {
            Console.WriteLine(p);
        }
    }

    private static void PrintInvoicesOfProduct(ProdContext prodContext, String prodName)
    {
        Console.WriteLine("List of invoices of product: " + prodName);
```

```
        var invoices = prodContext.InvoiceProducts
            .Include(ip => ip.Invoice)
            .Where(ip => ip.Product.Name == prodName)
            .Select(ip => ip.Invoice.InvoiceNumber).ToList();

        foreach (var i in invoices)
        {
            Console.WriteLine(i);
        }
    }

    static void Main(string[] args)
    {
        ProdContext prodContext = new ProdContext();

        AddProduct(prodContext, "Laptop");
        AddProduct(prodContext, "Lamp");
        AddProduct(prodContext, "TV");
        AddProduct(prodContext, "Fridge");
        AddProduct(prodContext, "Printer");

        AddSupplier(prodContext, "ElectronicsSupplier");

        AddCategory(prodContext, "Electronics");

        AddInvoice(prodContext, 1, 3);
        AddInvoice(prodContext, 2, 3);

        AddInvoiceProduct(prodContext, 1, "Laptop");
        AddInvoiceProduct(prodContext, 1, "Lamp");
        AddInvoiceProduct(prodContext, 1, "TV");
        AddInvoiceProduct(prodContext, 2, "Fridge");
        AddInvoiceProduct(prodContext, 2, "Printer");
        AddInvoiceProduct(prodContext, 2, "Laptop");

        ConnectProductSupplier(prodContext, "Laptop", "ElectronicsSupplier");
        ConnectProductSupplier(prodContext, "Lamp", "ElectronicsSupplier");
        ConnectProductSupplier(prodContext, "TV", "ElectronicsSupplier");
        ConnectProductSupplier(prodContext, "Fridge", "ElectronicsSupplier");
        ConnectProductSupplier(prodContext, "Printer", "ElectronicsSupplier");

        ConnectProductCategory(prodContext, "Laptop", "Electronics");
        ConnectProductCategory(prodContext, "Lamp", "Electronics");
        ConnectProductCategory(prodContext, "TV", "Electronics");
        ConnectProductCategory(prodContext, "Fridge", "Electronics");
        ConnectProductCategory(prodContext, "Printer", "Electronics");

        PrintProductsOfInvoice(prodContext, 1);
        PrintInvoicesOfProduct(prodContext, "Laptop");
    }
}
```

Wynik działania programu

```
List of products of invoice: 1
Laptop
Lamp
TV
List of invoices of product: Laptop
1
2
```

Wygląd tabeli Product

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ProductID	INTEGER							*NULL*
2	Name	TEXT							*NULL*
3	UnitsInStock	INTEGER							*NULL*
4	SupplierID	INTEGER							*NULL*
5	CategoryID	INTEGER							*NULL*

Wygląd tabeli Invoice

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	InvoiceID	INTEGER							*NULL*
2	InvoiceNumber	INTEGER							*NULL*
3	Quantity	INTEGER							*NULL*

Wygląd tabeli InvoiceProduct

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ProductID	INTEGER							*NULL*
2	InvoiceID	INTEGER							*NULL*

Select z tabeli Product

	ProductID	Name	UnitsInStock	SupplierID	CategoryID
1	1	Laptop	0	1	1
2	2	Lamp	0	1	1
3	3	TV	0	1	1
4	4	Fridge	0	1	1
5	5	Printer	0	1	1

Select z tabeli Invoice

	InvoiceID	InvoiceNumber	Quantity
1	1	1	3
2	2	2	3

Select z tabeli InvoiceProduct

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

Zadanie 7

Dodałem klasę Company po której będą dziedziczyć klasy Supplier i Customer

Company class

```
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; }
}
```

Następnie zmodyfikowałem klasę Supplier oraz dodałem nową klasę Customer

Supplier class

```
class Supplier:Company
{
    public Supplier()
    {
        Products = new List<Product>();
    }

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

Customer class

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

TablePerHierarchy

W klasie ProdContext zamieniłem set Suppliers na Companies oraz została zmodyfikowana metoda OnModelCreating w celu dodania i pobrania z bazy firm, stosując TablePerHierarchy

ProdContext class

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

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) =>
        optionsBuilder.UseSqlite("DataSource=Product.db");

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

Na koniec zmodyfikowałem metody AddSupplier, PrintSuppliersCompaniesNames oraz dodałęm nowę metody AddCustomer PrintCustomerCompaniesNames w klasie Program

Program class

```
class Program
{
    private static void AddSupplier(ProdContext prodContext, string companyName,
        string street, string city, string zipCode, string bankAccountNumber)
    {
        Supplier supplier = new Supplier
        {
            CompanyName = companyName,
            Street = street,
            City = city,
            ZipCode = zipCode,
            BankAccountNumber = bankAccountNumber
        };
```

```
            prodContext.Companies.Add(supplier);
            prodContext.SaveChanges();
        }

        private static void AddCustomer(ProdContext prodContext, string companyName,
            string street, string city, string zipCode, float discount)
        {
            Customer customer = new Customer
            {
                CompanyName = companyName,
                Street = street,
                City = city,
                ZipCode = zipCode,
                Discount = discount
            };
            prodContext.Companies.Add(customer);
            prodContext.SaveChanges();
        }

        private static void PrintSuppliersCompaniesNames(ProdContext prodContext)
        {
            Console.WriteLine("List of suppliers companies names from database:");
            var query = prodContext.Companies.OfType<Supplier>().ToList();

            foreach (var item in query)
            {
                Console.WriteLine("Company Name: " + item.CompanyName + " Bank Account Number: " + item.BankAccountNumber)
            }
        }

        private static void PrintCustomersCompaniesNames(ProdContext prodContext)
        {
            Console.WriteLine("List of customers companies names from database:");
            var query = prodContext.Companies.OfType<Customer>().ToList();

            foreach (var item in query)
            {
                Console.WriteLine("Company Name: " + item.CompanyName + " Discount: " + item.Discount);
            }
        }

        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();

            AddSupplier(prodContext, "Supplier1", "Somewhere", "Anywhere", "12345", "1234567890");
            AddSupplier(prodContext, "Supplier2", "Somewhere", "Anywhere", "12345", "2345678901");
            AddSupplier(prodContext, "Supplier3", "Somewhere", "Anywhere", "12345", "3456789012");
            AddSupplier(prodContext, "Supplier4", "Somewhere", "Anywhere", "12345", "4567890123");
            AddSupplier(prodContext, "Supplier5", "Somewhere", "Anywhere", "12345", "5678901234");
            AddSupplier(prodContext, "Supplier6", "Somewhere", "Anywhere", "12345", "6789012345");
            AddSupplier(prodContext, "Supplier7", "Somewhere", "Anywhere", "12345", "7890123456");

            AddCustomer(prodContext, "Customer1", "Somewhere", "Anywhere", "12345", 0.05f);
            AddCustomer(prodContext, "Customer2", "Somewhere", "Anywhere", "12345", 0.10f);
            AddCustomer(prodContext, "Customer3", "Somewhere", "Anywhere", "12345", 0.15f);
            AddCustomer(prodContext, "Customer4", "Somewhere", "Anywhere", "12345", 0.20f);
            AddCustomer(prodContext, "Customer5", "Somewhere", "Anywhere", "12345", 0.25f);
            AddCustomer(prodContext, "Customer6", "Somewhere", "Anywhere", "12345", 0.35f);
            AddCustomer(prodContext, "Customer7", "Somewhere", "Anywhere", "12345", 0.45f);

            PrintSuppliersCompaniesNames(prodContext);
            PrintCustomersCompaniesNames(prodContext);
        }
    }
```

Wynik działania programu

```
List of suppliers companies names from database:
Company Name: Supplier1 Bank Account Number: 1234567890
Company Name: Supplier2 Bank Account Number: 2345678901
Company Name: Supplier3 Bank Account Number: 3456789012
Company Name: Supplier4 Bank Account Number: 4567890123
Company Name: Supplier5 Bank Account Number: 5678901234
Company Name: Supplier6 Bank Account Number: 6789012345
Company Name: Supplier7 Bank Account Number: 7890123456
List of customers companies names from database:
Company Name: Customer1 Discount: 0.05
Company Name: Customer2 Discount: 0.1
Company Name: Customer3 Discount: 0.15
Company Name: Customer4 Discount: 0.2
Company Name: Customer5 Discount: 0.25
Company Name: Customer6 Discount: 0.35
Company Name: Customer7 Discount: 0.45
```

Wygląd tabeli Company

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	CompanyID	INTEGER							NULL
2	CompanyName	TEXT							NULL
3	Street	TEXT							NULL
4	City	TEXT							NULL
5	ZipCode	TEXT							NULL
6	Discriminator	TEXT							NULL
7	Discount	REAL							NULL
8	BankAccountNumber	TEXT							NULL

Select z tabeli Company

	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	Discount	BankAccountNumber
1	1	Supplier1	Somewhere	Anywhere	12345	Supplier	NULL	1234567890
2	2	Supplier2	Somewhere	Anywhere	12345	Supplier	NULL	2345678901
3	3	Supplier3	Somewhere	Anywhere	12345	Supplier	NULL	3456789012
4	4	Supplier4	Somewhere	Anywhere	12345	Supplier	NULL	4567890123
5	5	Supplier5	Somewhere	Anywhere	12345	Supplier	NULL	5678901234
6	6	Supplier6	Somewhere	Anywhere	12345	Supplier	NULL	6789012345
7	7	Supplier7	Somewhere	Anywhere	12345	Supplier	NULL	7890123456
8	8	Customer1	Somewhere	Anywhere	12345	Customer	0.05000000074506	NULL
9	9	Customer2	Somewhere	Anywhere	12345	Customer	0.10000000149012	NULL
10	10	Customer3	Somewhere	Anywhere	12345	Customer	0.15000000596046	NULL
11	11	Customer4	Somewhere	Anywhere	12345	Customer	0.20000000298023	NULL
12	12	Customer5	Somewhere	Anywhere	12345	Customer	0.25	NULL
13	13	Customer6	Somewhere	Anywhere	12345	Customer	0.34999999403954	NULL
14	14	Customer7	Somewhere	Anywhere	12345	Customer	0.44999998807907	NULL

TablePerType, TablePerClass Strategii mapowania dziedziczenia **TablePerType** oraz **TablePerClass** nie są dostępne w wersjach od 3.0 Entity Framework, więc nie da się tego wykonać.