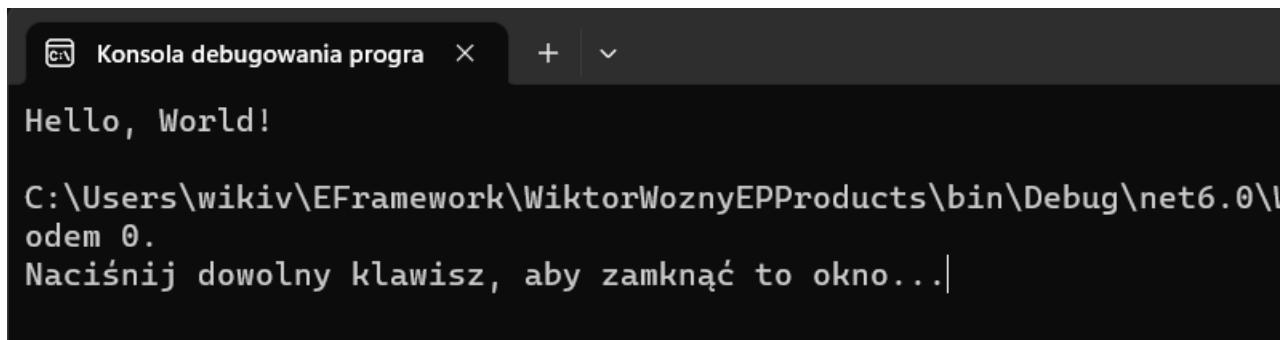


Zadanie 1

Kompilacja hello world:



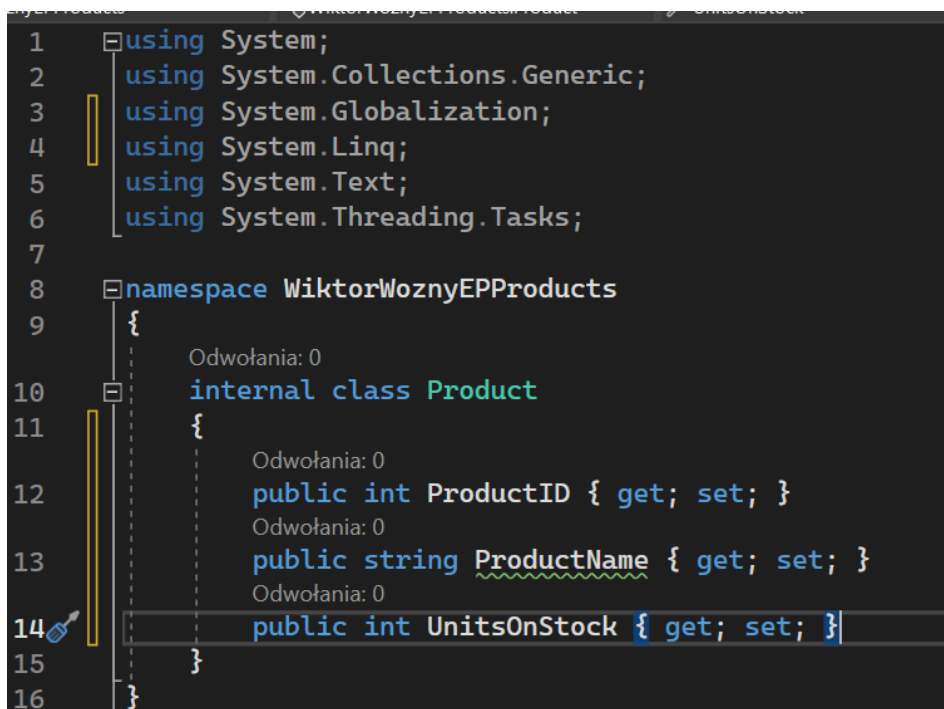
```
Konsola debugowania progra x + v

Hello, World!

C:\Users\wikiv\EFramework\WiktorWoznyEPProducts\bin\Debug\net6.0\...
odem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...|
```

Działa poprawnie

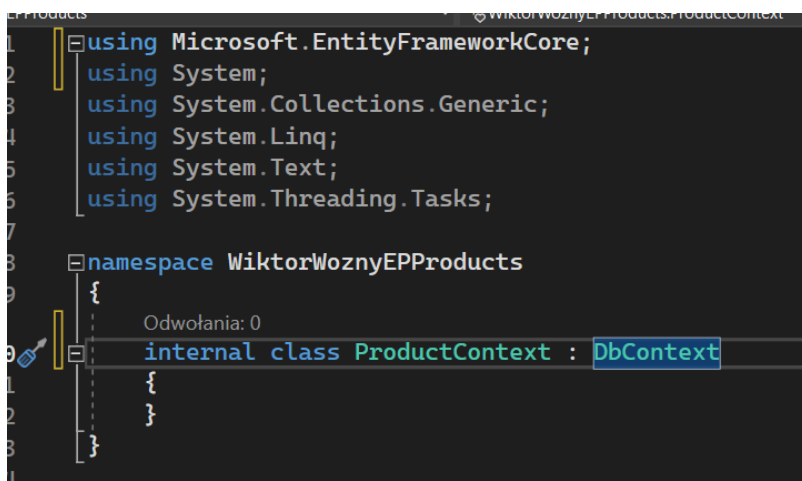
Tworzenie pierwszej klasy Product:



```
1 using System;
2 using System.Collections.Generic;
3 using System.Globalization;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace WiktorWoznyEPProducts
9 {
10     internal class Product
11     {
12         public int ProductID { get; set; }
13         public string ProductName { get; set; }
14         public int UnitsOnStock { get; set; }
15     }
16 }
```

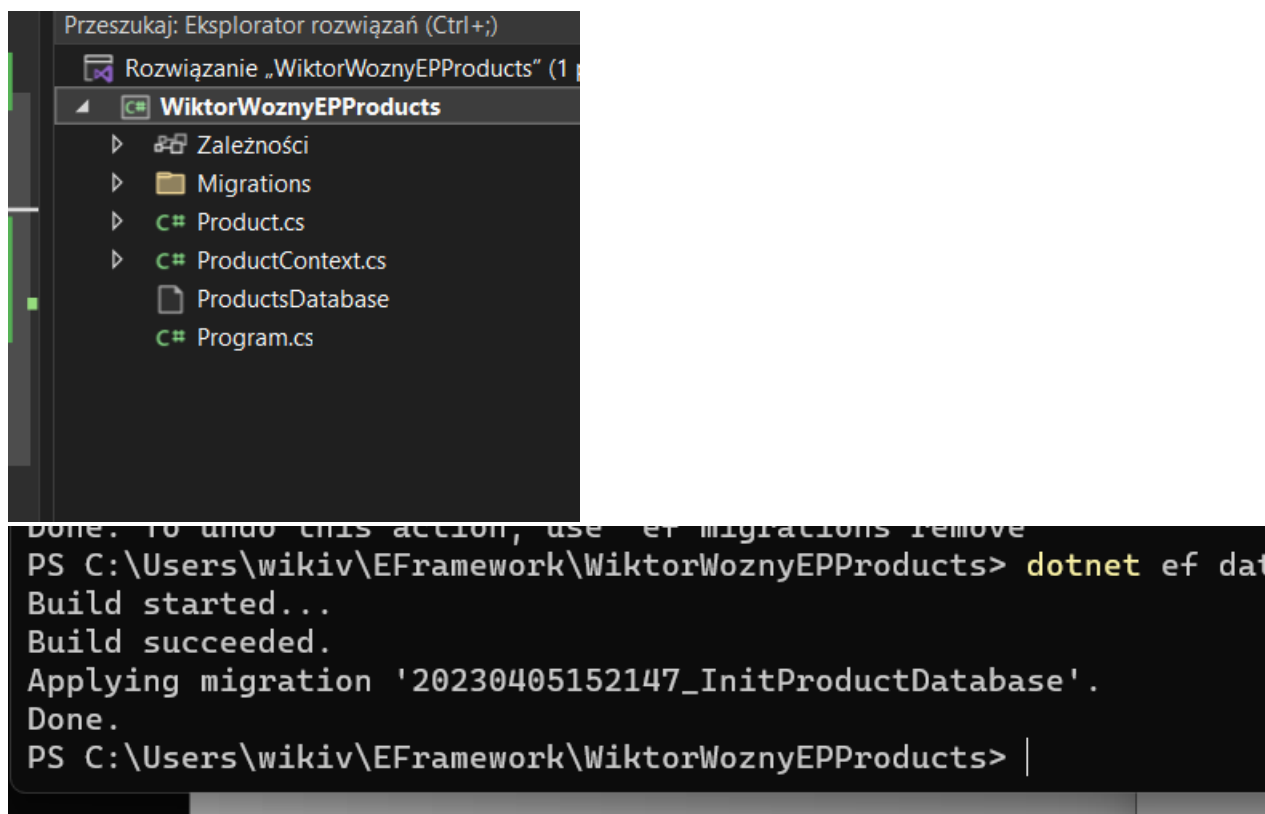
Klasa została poprawnie uzupełniona o publiczne property

Stworzenie klasy ProductContext, która dziedziczy po DbContext, po użyciu odpowiedniego usinga znikają wszystkie błędy

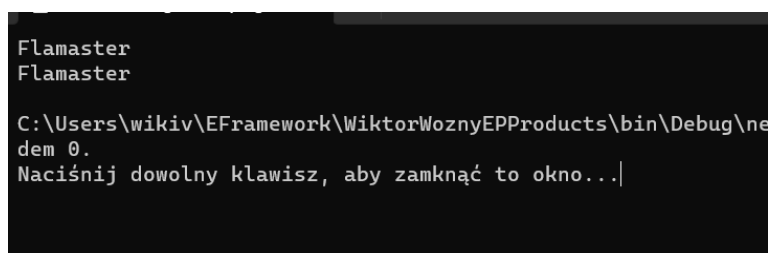
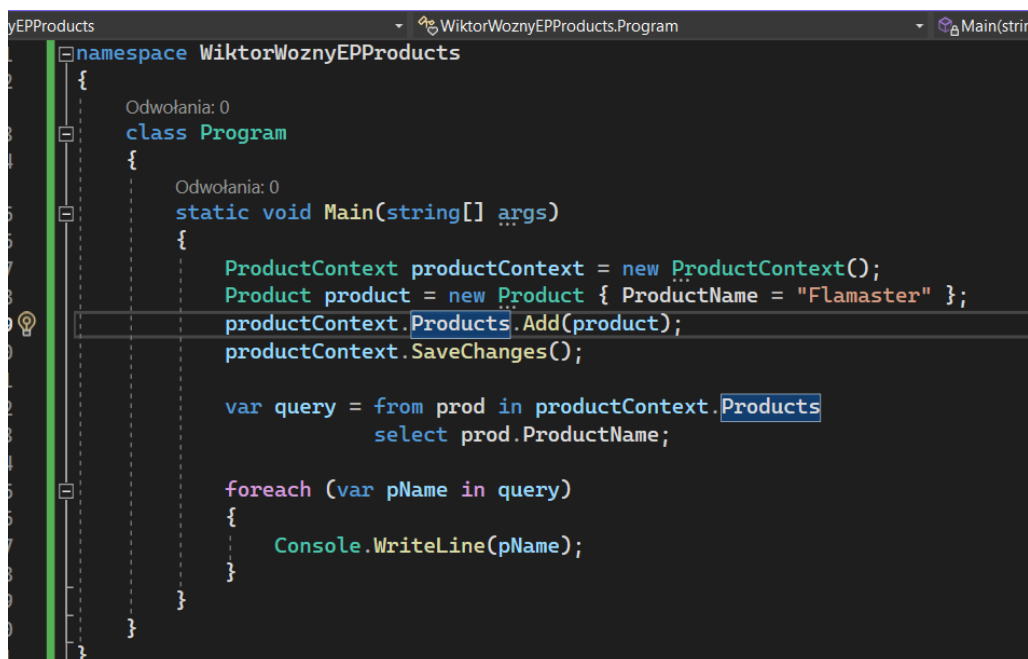


```
1 using Microsoft.EntityFrameworkCore;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7
8 namespace WiktorWoznyEPProducts
9 {
10     internal class ProductContext : DbContext
11     {
12     }
13 }
```

Dodanie migracji i update bazy danych:



Dodawanie do bazy danych i wypisanie w konsoli:



Jak widać dodawanie działa prawidłowo

Dodatkowo, po zmodyfikowaniu działania funkcji main, program działa następująco:

```
1  Podaj nazwe produktu:
2  zeszycik
3
4  Lista produktow:
5  Flamaster
6  Flamaster
7  Flamaster
8  Flamaster
9  zeszycik
10
11 C:\Users\wikiv\EFramework\WiktorWoznyEPPProducts\bin\Deb
12 odem 0.
13 Naciśnij dowolny klawisz, aby zamknąć to okno...|
```

Zadanie I

Dodałem do projektu klasę Supplier:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WiktorWoznyEPPProducts
{
    Odwołania: 3
    class Supplier
    {
        1 odwołanie
        [Key]
        public int CompanyID { get; set; }
        Odwołania: 0
        public string CompanyName { get; set; }
        Odwołania: 0
        public string Street { get; set; }
        Odwołania: 0
        public string City { get; set; }
    }
}
```

Za pomocą [Key] określiłem, że klucz główny obiektu klasy Supplier to CompanyID. Następnie zmodyfikowałem klasę Product dopisując atrybut Supplier (nullable).

```
1 odwołanie
public string ProductName { get; set; }
Odwołania: 0
public int UnitsOnStock { get; set; }
1 odwołanie
public Supplier? Supplier { get; set; }
}
```

Dodałem nową migrację:

```
7 dotnet ef migrations add AddSupplier
8 dotnet ef database update
```

A następnie nowego dostawcę do bazy danych:

```
Odwołania: 0
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();
    Supplier supplier = new Supplier { CompanyName="DHL", City="Krakow", Street="Mazowiecka"};
    productContext.Suppliers.Add(supplier);
    productContext.SaveChanges();
}
```

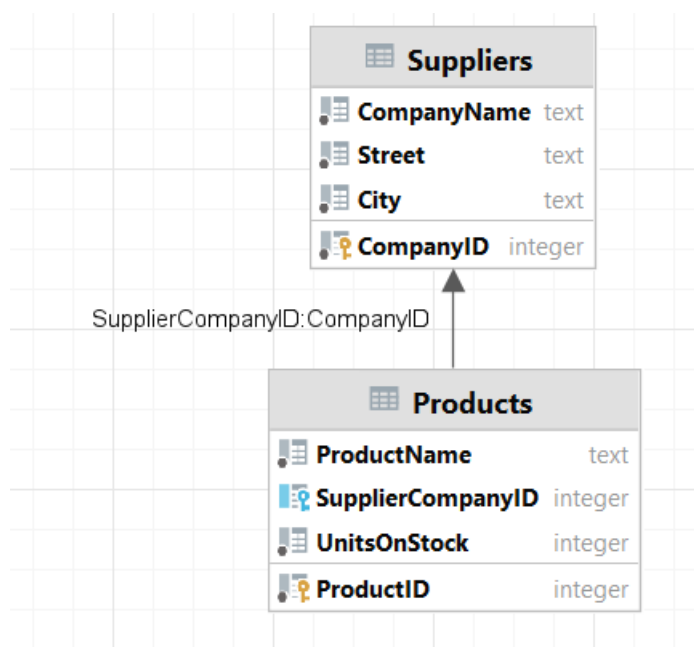
	CompanyID	CompanyName	Street	City
1	1	DHL	Mazowiecka	Krakow

Po wykonaniu tych kroków, produkty dodane uprzednio do bazy danych zostały usunięte, zatem dodałem nowy o nazwie Flamaster i z dostawcą przed chwilą dodanym:

```
Odwołania: 0
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();
    Supplier supplier = productContext.Suppliers.FirstOrDefault(s => s.CompanyID == 1);
    Product product = new Product { ProductName = "Flamaster", Supplier = supplier };
    productContext.Products.Add(product);
    productContext.SaveChanges();
}
```

	ProductID	ProductName	SupplierCompanyID	UnitsOnStock
1	1	Flamaster	1	0

Schemat bazy po tych krokach wygląda następująco:



Zadanie II

Zadanie zacząłem od zmodyfikowania klas `Supplier` i `Product`. Do klasy `Product` dodałem klucz obcy `SupplierID`, a pole `Supplier` oznaczyłem atrybutem `[NotMapped]`, aby Entity Framework zignorował to pole i nie utworzył kolumny w bazie danych.

```
class Product
{
    [Key]
    Odwołania: 0
    public int ProductID { get; set; }
    Odwołania: 0
    public string ProductName { get; set; }
    Odwołania: 0
    public int UnitsOnStock { get; set; }
    Odwołania: 0
    public int SupplierID { get; set; }
    [NotMapped]
    Odwołania: 0
    public Supplier? Supplier { get; set; }
}
```

Do klasy `Supplier` dodałem pole `Products`, które jest kolekcją obiektów z klasy `Product`:

```
Odwołania: 4
class Supplier
{
    [Key]
    Odwołania: 0
    public int CompanyID { get; set; }
    1 odwołanie
    public string CompanyName { get; set; }
    1 odwołanie
    public string Street { get; set; }
    1 odwołanie
    public string City { get; set; }
    1 odwołanie
    public ICollection<Product>? Products { get; set; }
}
```

Do bazy danych dodałem nowego dostawcę z listą nowo dodanych produktów:

```
Odwołania: 0
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();

    var products = new List<Product>
    {
        new Product { ProductName = "product1", SupplierID = 1, UnitsOnStock = 1 },
        new Product { ProductName = "product2", SupplierID = 1, UnitsOnStock = 2 },
        new Product { ProductName = "product3", SupplierID = 1, UnitsOnStock = 3 },
        new Product { ProductName = "product4", SupplierID = 1, UnitsOnStock = 4 },
        new Product { ProductName = "product5", SupplierID = 1, UnitsOnStock = 5 },
        new Product { ProductName = "product6", SupplierID = 1, UnitsOnStock = 6 },
        new Product { ProductName = "product7", SupplierID = 1, UnitsOnStock = 7 }
    };

    Supplier supplier = new Supplier { City = "Krakow", CompanyName = "DHL", Street = "mazowiecka", Products = products };
    productContext.Suppliers.Add(supplier);

    productContext.Suppliers.AddRange(supplier);

    productContext.SaveChanges();
}
```

	CompanyID	CompanyName	Street	City
1	2	DHL	mazowiecka	Krakow

	ProductID	ProductName	SupplierID	UnitsOnStock
1	8	product1	2	1
2	9	product2	2	2
3	10	product3	2	3
4	11	product4	2	4
5	12	product5	2	5
6	13	product6	2	6
7	14	product7	2	7

Zadanie III

Aby zmienić relację między klasą Product i Supplier na 1-n:n-1 (dwustronną) połączyłem dwa poprzednie rozwiązania w jedno:

```
namespace WiktorWoznyEPPProducts
{
    Odwołania: 10
    class Product
    {
        [Key]
        Odwołania: 0
        public int ProductID { get; set; }
        Odwołania: 7
        public string ProductName { get; set; }
        Odwołania: 7
        public int UnitsOnStock { get; set; }
        Odwołania: 7
        public Supplier? Supplier { get; set; }
    }
}
```

```
namespace WiktorWoznyEPPProducts
{
    Odwołania: 4
    class Supplier
    {
        [Key]
        Odwołania: 0
        public int CompanyID { get; set; }
        1 odwołanie
        public string CompanyName { get; set; }
        1 odwołanie
        public string Street { get; set; }
        1 odwołanie
        public string City { get; set; }
        1 odwołanie
        public ICollection<Product>? Products { get; set; }
    }
}
```

Dodałem migrację oraz rekordy do bazy danych:

```
namespace WiktorWoznyEPProducts
{
    Odwołania: 0
    class Program
    {
        Odwołania: 0
        static void Main(string[] args)
        {
            ProductContext productContext = new ProductContext();

            Supplier supplier = new Supplier { City = "Krakow", CompanyName = "DHL", Street = "mazowiecka" };

            var products = new List<Product>
            {
                new Product { ProductName = "product1", Supplier = supplier, UnitsOnStock = 1 },
                new Product { ProductName = "product2", Supplier = supplier, UnitsOnStock = 2 },
                new Product { ProductName = "product3", Supplier = supplier, UnitsOnStock = 3 },
                new Product { ProductName = "product4", Supplier = supplier, UnitsOnStock = 4 },
                new Product { ProductName = "product5", Supplier = supplier, UnitsOnStock = 5 },
                new Product { ProductName = "product6", Supplier = supplier, UnitsOnStock = 6 },
                new Product { ProductName = "product7", Supplier = supplier, UnitsOnStock = 7 }
            };

            supplier.Products = products;

            productContext.Suppliers.Add(supplier);
            productContext.Products.AddRange(products);

            productContext.SaveChanges();
        }
    }
}
```

	ProductID	ProductName	SupplierCompanyID	UnitsOnStock
1	1	product1	1	1
2	2	product2	1	2
3	3	product3	1	3
4	4	product4	1	4
5	5	product5	1	5
6	6	product6	1	6
7	7	product7	1	7

	CompanyID	CompanyName	Street	City
1	1	DHL	mazowiecka	Krakow

Zadanie IV

Na początku zadania dodałem nową klasę Invoice:

Odwołania: 5

```
class Invoice
{
    1 odwołanie
    public Invoice()
    {
        this.Products = new HashSet<Product>();
    }

    [Key]
    Odwołania: 0
    public int InvoiceNumber { get; set; }
    1 odwołanie
    public int Quantity { get; set; }
    1 odwołanie
    public virtual ICollection<Product> Products { get; set; }
}
```

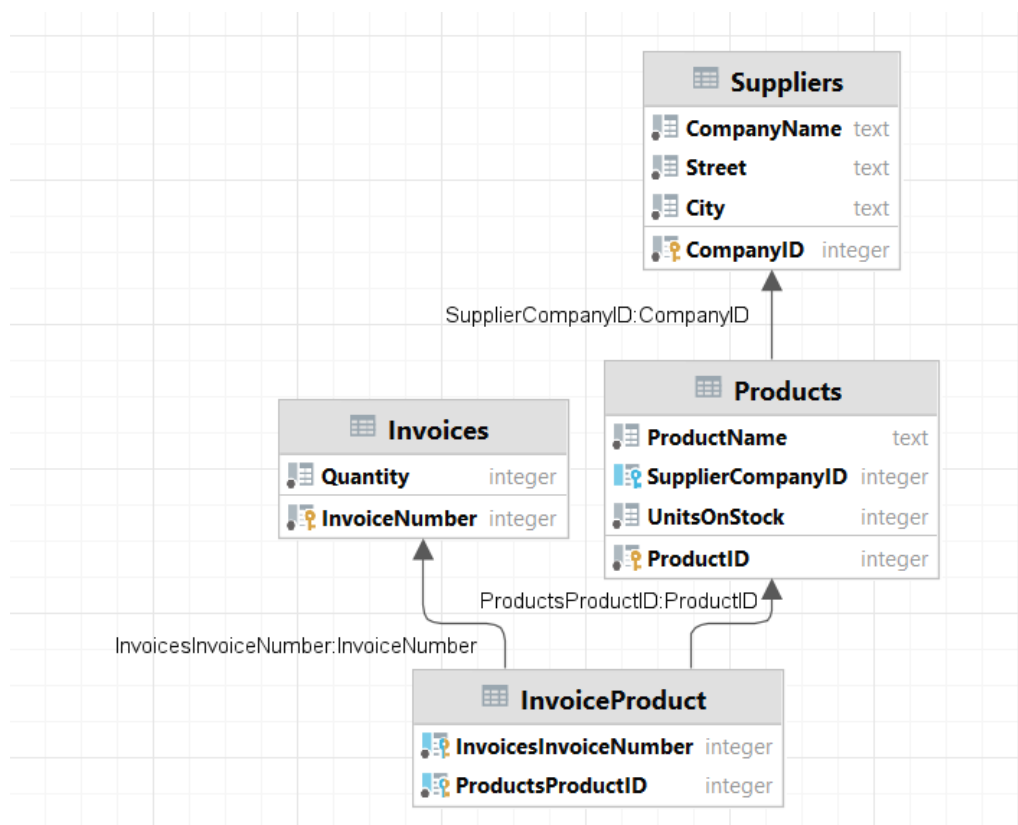
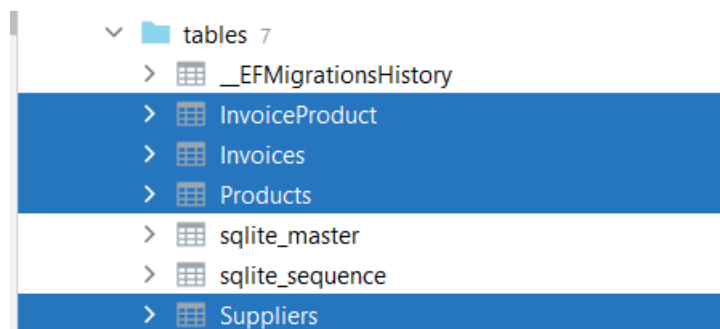
Oraz następująco zmodyfikowałem klasę Product:

Odwołania: 6

```
class Product
{
    Odwołania: 0
    public Product()
    {
        this.Invoices = new HashSet<Invoice>();
    }

    [Key]
    Odwołania: 0
    public int ProductID { get; set; }
    Odwołania: 0
    public string ProductName { get; set; }
    Odwołania: 0
    public int UnitsOnStock { get; set; }
    Odwołania: 0
    public Supplier? Supplier { get; set; }
    Odwołania: 2
    public virtual ICollection<Invoice> Invoices { get; set; }
}
```

Dzięki temu, po dodaniu nowej migracji w bazie pojawiła się nowa tabela InvoiceProduct, na schemacie wyglądało to następująco:



Przykładowy kod dodający do istniejącego produktu nowej faktury:

```
ProductContext productContext = new ProductContext();
Product product = productContext.Products.Find(3);
product.Invoices.Add(new Invoice { Quantity = 1 });
productContext.SaveChanges();
```

Po dodaniu paru innych rekordów, tabela InvoiceProduct wygląda następująco:

	InvoicesInvoiceNumber	ProductsProductID
1	1	3
2	2	5
3	2	6
4	2	3

Aby wyświetlić produkty, które zostały sprzedane przy pomocy określonej faktury, użyłem poniższego kodu:

```
Odwolania: 0
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();

    Console.WriteLine("enter invoice number: ");
    int num = Convert.ToInt32(Console.ReadLine());

    var products = productContext.Products.Where(p => p.Invoices.Any(i => i.InvoiceNumber == num)).ToList();

    Console.WriteLine("Products which were sold through invoice with number " + num + ": ");
    foreach (var product in products)
    {
        Console.WriteLine($"{product.ProductID}: {product.ProductName}");
    }
}
```

Działa to tak:

```
enter invoice number:
2
Products which were sold through invoice with number 2:
3: product3
5: product5
6: product6
```

Porównując z tabelą InvoiceProducts, faktycznie wszystko się zgadza.

Teraz aby wyświetlić faktury w ramach których był sprzedany wybrany produkt, użyłem analogicznego kodu:

```
Odwolania: 0
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();

    Console.WriteLine("enter product number: ");
    int num = Convert.ToInt32(Console.ReadLine());

    var invoices = productContext.Invoices.Where(p => p.Products.Any(i => i.ProductID == num)).ToList();

    Console.WriteLine("Invoices that contains products with id " + num + ": ");
    foreach (var invoice in invoices)
    {
        Console.WriteLine($"{invoice.InvoiceNumber}");
    }
}
```

Działa elegancko:

```
enter product number:
3
Invoices that contains products with id 3:
1
2
```

Zadanie V

Aby zastosować mechanizm dziedziczenia TPH dodałem klasę Company i zmodyfikowałem klasy Customer i Supplier:

Odwołania: 3

```
class Company
{
    [Key]
    Odwołania: 0
    public int CompanyID { get; set; }
    1 odwołanie
    public string CompanyName { get; set; }
    1 odwołanie
    public string Street { get; set; }
    1 odwołanie
    public string City { get; set; }
}
```

Odwołania: 2

```
class Supplier : Company
{
    Odwołania: 0
    public int BankAccountNumber { get; set; }
    Odwołania: 0
    public ICollection<Product>? Products { get; set; }
}
```

Odwołania: 3

```
class Customer : Company
{
    1 odwołanie
    public double Discount { get; set; }
}
```

Ważny jest dwukropek z dopiskiem Company po nazwach klas Customer i Supplier – oznacza on dziedziczenie.

Aby baza danych obsługiwała dodawanie obiektów tych klas do bazy dodałem DbSet'y do ProductContextu:

Odwołania: 11

```
class ProductContext : DbContext
{
    Odwołania: 0
    public DbSet<Product> Products { get; set; }
    Odwołania: 0
    public DbSet<Company> Companies { get; set; }
    Odwołania: 0
    public DbSet<Supplier> Suppliers { get; set; }
    1 odwołanie
    public DbSet<Customer> Customers { get; set; }
    Odwołania: 0
    public DbSet<Invoice> Invoices { get; set; }

    Odwołania: 0
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("DataSource=ProductsDatabase");
    }
}
```

Dzięki temu mogę w taki sposób dodawać rekordy do bazy:

```
ProductContext productContext = new ProductContext();

Supplier supplier = new Supplier { City = "Krakow", CompanyName = "DHL", Street = "mazowiecka", BankAccountNumber = 123123123 };

var products = new List<Product>
{
    new Product { ProductName = "product1", Supplier = supplier, UnitsOnStock = 1 },
    new Product { ProductName = "product2", Supplier = supplier, UnitsOnStock = 2 },
    new Product { ProductName = "product3", Supplier = supplier, UnitsOnStock = 3 },
    new Product { ProductName = "product4", Supplier = supplier, UnitsOnStock = 4 },
    new Product { ProductName = "product5", Supplier = supplier, UnitsOnStock = 5 },
    new Product { ProductName = "product6", Supplier = supplier, UnitsOnStock = 6 },
    new Product { ProductName = "product7", Supplier = supplier, UnitsOnStock = 7 }
};

supplier.Products = products;

productContext.Suppliers.Add(supplier);
productContext.Products.AddRange(products);

productContext.SaveChanges();
```

```
ProductContext productContext = new ProductContext();
Customer customer = new Customer { City = "Rzeszow", CompanyName = "Januszex", Discount = 0.15, Street = "mam zielone" };
productContext.Customers.Add(customer);
productContext.SaveChanges();
```

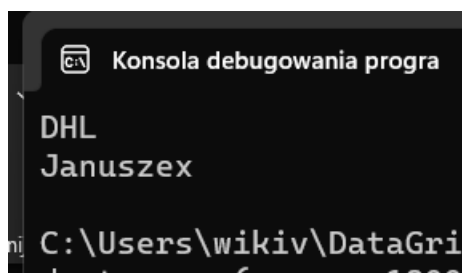
Wtedy w tabeli Companies mamy takie ładne rzeczy:

	CompanyID	BankAccountNumber	City	CompanyName	Discount	Discriminator	Street
1	1	123123123	Krakow	DHL	<null>	Supplier	mazowiecka
2	2	<null>	Rzeszow	Januszex	0.15	Customer	mam zielone

Dzięki kolumny Discriminator (btw automatycznie dodawanej przez EF) wiemy, z którą podklasą mamy do czynienia.

A tak pobieram dane z tabeli Companies:

```
ProductContext productContext = new ProductContext();
List<Company> companies = productContext.Companies.ToList();
foreach (Company company in companies)
{
    Console.WriteLine(company.CompanyName);
}
```



Zwrócić należy uwagę na klasę obiektu w liście – musi on być ustawiony na klasę nadrzędną, w tym wypadku Company. ŚMIGA

Zadanie VI

Aby uzyskać dziedziczenie TPT zmodyfikowałem klasę ProductContext dodając nową metodę OnModelCreating:

```
Odwołania: 12
class ProductContext : DbContext
{
    Odwołania: 0
    public DbSet<Product> Products { get; set; }
    Odwołania: 0
    public DbSet<Company> Companies { get; set; }
    1 odwołanie
    public DbSet<Supplier> Suppliers { get; set; }
    1 odwołanie
    public DbSet<Customer> Customers { get; set; }
    Odwołania: 0
    public DbSet<Invoice> Invoices { get; set; }

    Odwołania: 0
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("Datasource=ProductsDatabase");
    }

    Odwołania: 0
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Company>().UseTptMappingStrategy();
    }
}
```

Zmiana ta wystarczyła, aby po dodaniu nowych rekordów zostały one dodane do 3 różnych tabel:

```
Odwołania: 0
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();
    Supplier supplier = new Supplier { City = "Warszawka", CompanyName = "WWA", Street = "ciekawa", BankAccountNumber = 123423123 };
    productContext.Suppliers.Add(supplier);

    Customer customer = new Customer { Street = "niewiem", City = "Trzciana", CompanyName = "MediFaraon", Discount = 0.10 };
    productContext.Customers.Add(customer);

    productContext.SaveChanges();
}
```

	CompanyID	City	CompanyName	Street
1	1	Trzciana	MediFaraon	niewiem
2	2	Warszawka	WWA	ciekawa

	CompanyID	BankAccountNumber
1	2	123423123

	CompanyID	Discount
1	1	0.1

Dane odczytuję w następujący sposób:

Odwołania: 0

```
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();
    var suppliers = productContext.Companies.OfType<Supplier>().ToList();
    Console.WriteLine("Suppliers:");
    foreach (var supplier in suppliers)
    {
        Console.WriteLine($"{supplier.CompanyName} ({supplier.BankAccountNumber})");
    }

    var customers = productContext.Companies.OfType<Customer>().ToList();
    Console.WriteLine("Customers:");
    foreach (var customer in customers)
    {
        Console.WriteLine($"{customer.CompanyName} ({customer.Discount})");
    }
}
```

```
Suppliers:
WWA (123423123)
Customers:
MediFaraon (0,1)
```

Porównując obie strategię dziedziczenia mogę stwierdzić, że bardziej przemawia do mnie strategia TPH. Jedna tabela z dodatkową kolumną Discriminator wydaje się lepsza niż 3 różne tabele. Jednakże, gdyby klasy dziedziczące miały więcej pól niezgodnych z klasą nadrzędną, rozwiązanie to stałoby się zbyt nieczytelne. Wtedy użył bym metody TPT.