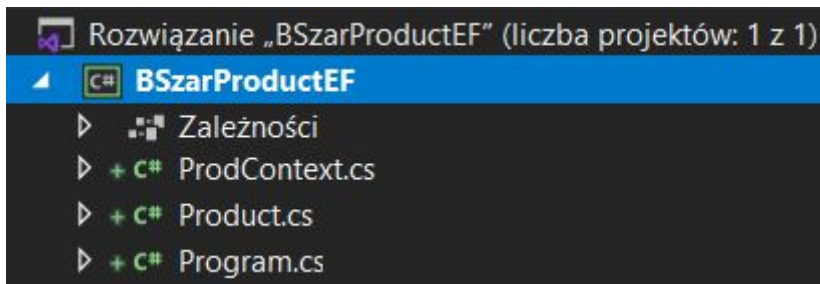


Entity Framework

Bartosz Szar

1.a - Stwórz projekt typu ConsoleApplication .Net Core.
Nazwij go INazwiskoProdotcEF:



1.b - Dodaj klasę Product z polami int ProductID, string Name, int UnitsInStock:

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

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

1.c, 1.d - Stwórz klasę ProdContext dziedziczącą po DbContext. Dodaj do klasy kontekstowej zbiór (DbSet) produktów i nazwij go Products:

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

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

1.e - 1.o - Skonfiguruj bazę danych oraz spraw by dodawanie produktów do listy, jak również ich wyświetlanie było możliwe:

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

namespace BSzarProductEF
{
    class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        protected override void OnConfiguring(DbContextOptionsBuilder options) => options.UseSqlite("DataSource=Product.db");
    }
}
```

```
C:\Users\Lenovo\source\repos\BSzarProductEF\BSzarProductEF>dotnet ef migrations add InitialProductCreation
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'


C:\Users\Lenovo\source\repos\BSzarProductEF\BSzarProductEF>dotnet ef database update
Build started...
Build succeeded.
Applying migration '20200420142216_InitialProductCreation'.
Done.
```

```

using System;
using System.Linq;

namespace BSzarProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("podaj nazwe produktu: ");
            string name = Console.ReadLine();
            Product product = new Product { Name = name };
            ProdContext prodContext = new ProdContext();
            prodContext.Products.Add(product);
            prodContext.SaveChanges();
            var products = (from prod in prodContext.Products
                           select prod).ToList();
            Console.WriteLine("lista produktow: ");
            foreach(var prod in products)
            {
                Console.WriteLine(prod.Name);
            }
            Console.WriteLine();
        }
    }
}

```

 Konsola debugowania programu Microsoft Visual Studio

podaj nazwe produktu: sok

lista produktow:

mleko

ser

jajka

sok

```

C:\Users\Lenovo\source\repos\BSzarProductEF\BSzarProductEF\bin\Debug\netcoreapp3.1>sqlite3 Product.db
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> select * from Products;
1|mleko|0
2|ser|0
3|jajka|0
4|sok|0
sqlite>

```

2.a - 2.c - Zmodyfikuj model wprowadzając pojęcie Dostawcy:

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

namespace BSzarProductEF
{
    class Supplier
    {
        [Key]
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
    }
}
```

```
using System;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;
using Microsoft.EntityFrameworkCore.Sqlite;

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

        [ForeignKey("Supplier")]
        public int SupplierID { get; set; }
        public Supplier supplier { get; set; }
    }
}
```

Po wykonaniu migracji i zaktualizowaniu bazy danych utworzona tabela Produktów zawiera klucz obcy odwołujący się do tabeli Dostawców:

```
CREATE TABLE IF NOT EXISTS "Products" (  
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,  
    "Name" TEXT NULL,  
    "UnitsInStock" INTEGER NOT NULL,  
    "SupplierID" INTEGER NOT NULL,  
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers" ("SupplierID") ON DELETE CASCADE  
);  
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
```

2.d - 2f - Dodaj do bazy nowy produkt, stwórz nowego dostawcę, oraz ustaw go jako dostawcę uprzednio wprowadzonego produktu:

Dodałem do klasy kontekstowej zbiór dostawców i nazwałem go Suppliers:

```
using Microsoft.EntityFrameworkCore;  
using System;  
using System.Collections.Generic;  
using System.Text;  
  
namespace BSzarProductEF  
{  
    class ProdContext : DbContext  
    {  
        protected override void OnConfiguring(DbContextOptionsBuilder options) => options.UseSqlite("DataSource=Product.db");  
        public DbSet<Product> Products { get; set; }  
        public DbSet<Supplier> Suppliers { get; set; }  
    }  
}
```

Przetestowałem działanie bazy po modyfikacji przypisując przykładowego dostawcę do danego produktu:

```
using System;
using System.Linq;

namespace BSzarProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            Supplier supplier = new Supplier { CompanyName = "Amazon" };

            Console.WriteLine("podaj nazwe produktu: ");
            string productName = Console.ReadLine();

            Product product = new Product { Name = productName };
            ProdContext prodContext = new ProdContext();

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

            var s = (from supp in prodContext.Suppliers
                    select supp).First();
            product.SupplierID = s.SupplierID;
            prodContext.Products.Add(product);
            prodContext.SaveChanges();

            var products = (from prod in prodContext.Products
                           select prod).ToList();

            Console.WriteLine();
            Console.WriteLine("lista produktow: ");
            foreach(var prod in products)
            {
                Console.WriteLine(prod.Name);
            }
            Console.WriteLine();
        }
    }
}
```

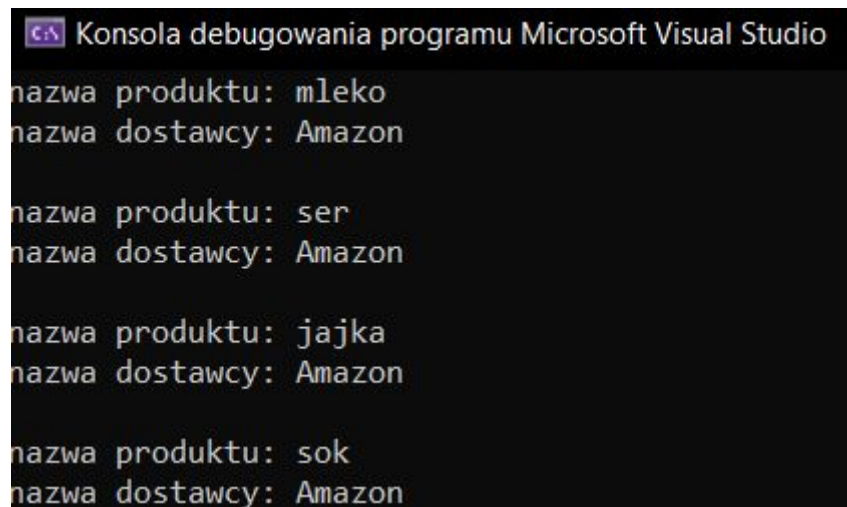
Po wykonaniu zapytania sql widzimy klucz obcy w tabeli Products, który odwołuje się do klucza głównego w tabeli Suppliers:

```
sqlite> select * from Products;
1|mleko|0|1
2|ser|0|1
3|jajka|0|1
4|sok|0|1
sqlite> select * from Suppliers;
1|Amazon||
```

2.g - Wyświetl wszystkie produkty wraz z nazwą dostawcy:

```
using System;
using System.Linq;

namespace BSzarProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();
            var tuples = (from p in prodContext.Products
                          join s in prodContext.Suppliers
                          on p.SupplierID equals s.SupplierID
                          select new
                          {
                              product = p.Name,
                              supplier = s.CompanyName
                          }).ToList();
            foreach (var tup in tuples)
            {
                Console.Write("nazwa produktu: ");
                Console.WriteLine(tup.product);
                Console.Write("nazwa dostawcy: ");
                Console.WriteLine(tup.supplier);
                Console.WriteLine();
            }
        }
    }
}
```



Konsola debugowania programu Microsoft Visual Studio

```
nazwa produktu: mleko  
nazwa dostawcy: Amazon  
  
nazwa produktu: ser  
nazwa dostawcy: Amazon  
  
nazwa produktu: jajka  
nazwa dostawcy: Amazon  
  
nazwa produktu: sok  
nazwa dostawcy: Amazon
```

3.a - 3.c - Odwróć relację tabeli product i supplier:

W klasie supplier dodałem listę produktów dostarczanych przez dostawcę:

```
using System;  
using System.Collections.Generic;  
using System.Collections.ObjectModel;  
using System.ComponentModel.DataAnnotations;  
using System.Text;  
  
namespace BSzarProductEF  
{  
    class Supplier  
    {  
        public Supplier()  
        {  
            Products = new Collection<Product>();  
        }  
  
        [Key]  
        public int SupplierID { get; set; }  
        public string CompanyName { get; set; }  
        public string Street { get; set; }  
        public string City { get; set; }  
        public ICollection<Product> Products { get; set; }  
    }  
}
```


Z klasy Product usunąłem powiązanie z dostawcą:

```
using System;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;
using Microsoft.EntityFrameworkCore.Sqlite;

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

Przetestowałem działanie bazy po modyfikacji tworząc przykładowego dostawcę oraz przypisując mu kilka przykładowych produktów:

```
using System;
using System.Linq;
using System.Collections.ObjectModel;
using Microsoft.EntityFrameworkCore;

namespace BSzarProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();
            Console.WriteLine("podaj nazwe produktu: ");
            string productname = Console.ReadLine();
            Product product = new Product { Name = productname };
            prodContext.Products.Add(product);
            Console.WriteLine();


            Supplier supplier = prodContext.Suppliers.
                FirstOrDefault(supp => supp.CompanyName == "Lipski");
            if(supplier == null)
            {
                supplier = new Supplier { CompanyName = "Lipski" };
                prodContext.Suppliers.Add(supplier);
            }
            supplier.Products.Add(product);
        }
    }
}
```

```

        prodContext.SaveChanges();

        Supplier s = prodContext.Suppliers.
            Include(s => s.Products).First();
        Console.Write("lista produktow dostarczonych przez: ");
        Console.WriteLine(s.CompanyName);
        foreach (Product p in s.Products)
        {
            Console.WriteLine(p.Name);
        }
        Console.WriteLine();
    }
}
}

```

 Konsola debugowania programu Microsoft Visual Studio

podaj nazwe produktu: waciki

```

lista produktow dostarczonych przez: Lipski
waciki
palac kultury
zurek z sheratona
sztabka zlota
pizza kiler
solniczka
biala kielbasa

```

```

SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> select * from Products;
1|palac kultury|0|1
2|zurek z sheratona|0|1
3|sztabka zlota|0|1
4|pizza kiler|0|1
5|solniczka|0|1
6|biala kielbasa|0|1
7|waciki|0|1
sqlite> select * from Suppliers;
1|Lipski|

```

4.a - 4.c - Zamodeluj relację dwustronną pomiędzy tabelą dostawców a tabelą produktów:

W klasie Product dodałem pole wiążące produkt z dostawcą:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;
using Microsoft.EntityFrameworkCore.Sqlite;

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

Klasa Supplier nie wymaga modyfikacji - każdy dostawca posiada liczbę dostarczanych przez siebie produktów.

Przetestowałem działanie bazy po modyfikacji tworząc przykładowego dostawcę, przypisując mu kilka przykładowych produktów oraz każdemu z produktów przypisując wspomnianego sprzedawcę:

```
using System;
using System.Linq;
using System.Collections.ObjectModel;
using Microsoft.EntityFrameworkCore;

namespace BSzarProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();
            Console.WriteLine("podaj nazwe produktu: ");
            string productname = Console.ReadLine();
            Product product = new Product { Name = productname };
            prodContext.Products.Add(product);
        }
    }
}
```

```

        Console.WriteLine();

        Supplier supplier = prodContext.Suppliers.
            FirstOrDefault(supp => supp.CompanyName == "Lipski");
        if(supplier == null)
        {
            supplier = new Supplier { CompanyName = "Lipski" };
            prodContext.Suppliers.Add(supplier);
        }
        product.Supplier = supplier;
        supplier.Products.Add(product);
        prodContext.SaveChanges();

        Supplier s = prodContext.Suppliers.
            Include(s => s.Products).First();
        foreach (Product p in s.Products)
        {
            Console.Write("dostawca: ");
            Console.WriteLine(p.Supplier.CompanyName);
            Console.Write("produkt: ");
            Console.WriteLine(p.Name);
            Console.WriteLine();
        }
        Console.WriteLine();
    }
}
}

```

```

Konsola debugowania programu Microsoft Visual Studio
podaj nazwe produktu: telewizyjne korniszony
dostawca: Lipski
produkt: telewizyjne korniszony

dostawca: Lipski
produkt: palac kultury

dostawca: Lipski
produkt: zurek z sheratona

dostawca: Lipski
produkt: sztabka zlota

dostawca: Lipski
produkt: pizza kiler

dostawca: Lipski
produkt: solniczka

dostawca: Lipski
produkt: biala kielbasa

dostawca: Lipski
produkt: waciki

dostawca: Lipski
produkt: 500 pesos

```

```

sqlite> select * from Products;
1|palac kultury|0|1
2|zurek z sheratona|0|1
3|sztabka zlota|0|1
4|pizza kiler|0|1
5|solniczka|0|1
6|biala kielbasa|0|1
7|waciki|0|1
8|500 pesos|0|1
9|telewizyjne korniszony|0|1

```

5.a - 5.c - Dodaj klasę Category z property int CategoryID, String Name oraz listą produktów:

Stworzyłem klasę Category:

```

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Text;

namespace BSzarProductEF
{
    class Category
    {
        public Category() => Products = new Collection<Product>();

        public int CategoryID { get; set; }
        public string CategoryName { get; set; }

        public ICollection<Product> Products { get; set; }
    }
}

```

Zmodyfikowałem klasę Product, dodając atrybut wskazujący jego kategorię:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text;
using Microsoft.EntityFrameworkCore.Sqlite;

namespace BSzarProductEF
{
    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; }
    }
}
```

Dodałem do klasy kontekstowej zbiór kategorii i nazwałem go Categories:

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

namespace BSzarProductEF
{
    class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        protected override void OnConfiguring(
            DbContextOptionsBuilder options) =>
            options.UseSqlite("DataSource=Product.db");
        public DbSet<Supplier> Suppliers { get; set; }
        public DbSet<Category> Categories { get; set; }
    }
}
```

Przetestowałem działanie bazy po modyfikacji tworząc przykładową kategorię oraz przypisując do niej kilka przykładowych produktów:

```
using System;
using System.Linq;
using System.Collections.ObjectModel;
using Microsoft.EntityFrameworkCore;

namespace BSzarProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();
            Console.Write("podaj nazwe produktu: ");
            string productname = Console.ReadLine();
            Product product = new Product { Name = productname };
            prodContext.Products.Add(product);
            Console.WriteLine();

            Supplier supplier = prodContext.Suppliers.
                FirstOrDefault(supp => supp.CompanyName == "Lipski");
            if(supplier == null)
            {
                supplier = new Supplier { CompanyName = "Lipski" };
                prodContext.Suppliers.Add(supplier);
            }

            Category category = prodContext.Categories.
                FirstOrDefault(cat => cat.CategoryName == "owoce");
            if(category == null)
            {
                category = new Category { CategoryName = "owoce" };
                prodContext.Categories.Add(category);
            }


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

            Category c = prodContext.
                Categories.Include(c => c.Products).First();
            Console.Write("lista produktow kategorii: ");
```

```

        Console.WriteLine(c.CategoryName);
        foreach (Product p in c.Products)
        {
            Console.WriteLine(p.Name);
        }
        Console.WriteLine();
    }
}
}

```

 Konsola debugowania programu Microsoft Visual Studio

podaj nazwe produktu: gruszka

lista produktow kategorii: owoce

gruszka

jablko

pomarancza

banan

5.c - Wydobądź produkty z wybranej kategorii oraz kategorię do której należy wybrany produkt:

```

sqlite> select * from Products;
1|jablko|0|1|1
2|pomarancza|0|1|1
3|banan|0|1|1
4|gruszka|0|1|1
sqlite> select * from Categories;
1|owoce

```

```

sqlite> select Products.Name
...> from Products
...> join Categories
...> on Products.CategoryID = Categories.CategoryID;
jablko
pomarancza
banan
gruszka

```

```

sqlite> select CategoryName
...> from Categories
...> where CategoryID == 1;
owoce

```


6.a - Zamodeluj relacje wiele-do-wielu między tabelą Invoice a tabelą Products:

Stworzyłem klasę Invoice przechowującą dane o fakturach:

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

namespace BSzarProductEF
{
    class Invoice
    {
        public Invoice() => InvoiceRecords =
            new Collection<InvoiceRecord>();

        public int InvoiceNumber { get; set; }
        public ICollection<InvoiceRecord> InvoiceRecords { get; set; }
    }
}
```

Każda faktura dzieli się na rekordy - każdy z nich zawiera id produktu którego dotyczy, oraz jego ilość:

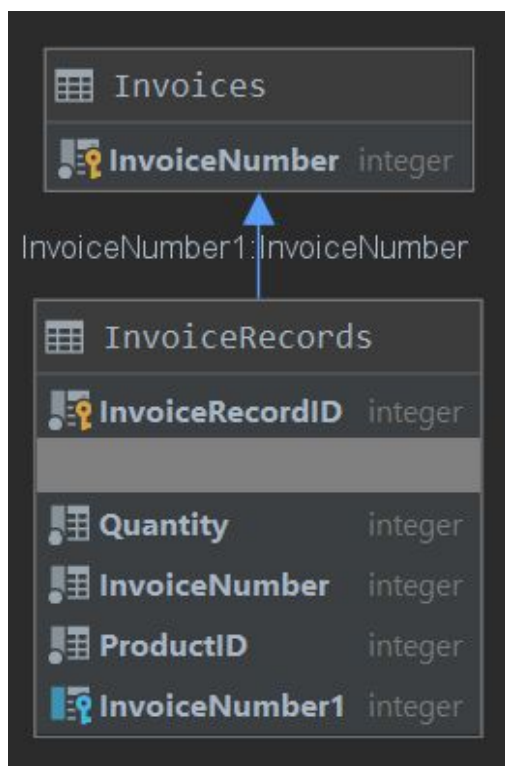
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema; using System.Text;
namespace BSzarProductEF
{
    class InvoiceRecord
    {
        public int ProductID { get; set; }
        public int Quantity { get; set; }

        [ForeignKey("Invoice")]
        public int InvoiceNumber { get; set; }
    }
}
```

Dodałem do klasy kontekstowej zbiór faktur, oraz rekordów faktur i nazwałem je odpowiednio Invoices oraz InvoiceRecords:

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

namespace BSzarProductEF
{
    class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        protected override void OnConfiguring(DbContextOptionsBuilder
            options) => options.UseSqlite("DataSource=Product.db");
        public DbSet<Supplier> Suppliers { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<InvoiceRecord> InvoiceRecords { get; set; }
    }
}
```



Przetestowałem działanie bazy po modyfikacji tworząc przykładową fakturę oraz przypisując do niej kilka przykładowych produktów:

```
using System;
using System.Linq;
using System.Collections.ObjectModel;
using Microsoft.EntityFrameworkCore;

namespace BSzarProductEF{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();
            Console.WriteLine("podaj nazwe produktu: ");
            string productname = Console.ReadLine();
            Product product = new Product { Name = productname };
            prodContext.Products.Add(product);
            Console.WriteLine();

            Supplier supplier = prodContext.Suppliers.
                FirstOrDefault(supp => supp.CompanyName == "Lipski");
            if(supplier == null)
            {
                supplier = new Supplier { CompanyName = "Lipski" };
                prodContext.Suppliers.Add(supplier);
            }

            Category category = prodContext.Categories.
                FirstOrDefault(cat => cat.CategoryName == "owoce");
            if(category == null)
            {
                category = new Category { CategoryName = "owoce" };
                prodContext.Categories.Add(category);
            }

            product.Supplier = supplier;
            product.Category = category;
            supplier.Products.Add(product);
            category.Products.Add(product);
            prodContext.SaveChanges();
            Invoice invoice = new Invoice();
            prodContext.Invoices.Add(invoice);

            InvoiceRecord invoiceRecord = new InvoiceRecord();
            invoiceRecord.InvoiceNumber = invoice.InvoiceNumber;
```

```

        invoiceRecord.ProductID = product.ProductID;
        prodContext.InvoiceRecords.Add(invoiceRecord);

        invoice.InvoiceRecords.Add(invoiceRecord);
    }
}
}

```

6.b - Pokaż produkty sprzedane w ramach wybranej faktury/transakcji:

```

sqlite> select Products.Name
...> from Products
...> join InvoiceRecords
...> on Products.ProductID = InvoiceRecords.ProductID
...> join Invoices
...> on Invoices.InvoiceNumber = InvoiceRecords.InvoiceNumber;
kasztany
mleko

```

6.c - Pokaż faktury w ramach których był sprzedany wybrany produkt:

```

sqlite> select *
...> from Invoices
...> join InvoiceRecords
...> on Invoices.InvoiceNumber = InvoiceRecords.InvoiceNumber
...> where InvoiceRecords.ProductID == 8;
1|4|3|1|8|

```

7.a - Wprowadź do modelu hierarchię dziedziczenia - klasy Supplier i Customer rozszerzają klasę Company:

Stworzyłem klasę Company przechowującą informacje wspólne dla klientów i dostawców:

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

namespace BSzarProductEF
{
    class Company
    {
        [Key]
        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; }
    }
}
```

Stworzyłem klasę Customer:

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

namespace BSzarProductEF
{
    class Customer : Company
    {
        public int Discount { get; set; }
    }
}
```

Uwzględniłem zmiany w klasie Supplier:

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

namespace BSzarProductEF
{
    class Supplier : Company
    {
        public Supplier() => Products = new Collection<Product>();

        public int BankAccountNumber { get; set; }
        public virtual ICollection<Product> Products { get; set; }
    }
}
```

7.b - TablePerHierarchy:

Zmodyfikowałem klasę ProdContext, umożliwiając tym samym współistnienie pól Discount oraz BankAccountNumber w tabeli Companies:

```
namespace BSzarProductEF
{
    class ProdContext : DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder options) => options.UseSqlite("DataSource=Product.db");
        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<InvoiceRecord> InvoiceRecords { get; set; }
        public DbSet<Company> Companies { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Customer>();
            modelBuilder.Entity<Supplier>();
        }
    }
}
```

Przetestowałem działanie bazy po modyfikacji tworząc przykładowych klientów i dostawców oraz dodając ich do tabeli Companies:

```
using System;
using System.Linq;

namespace BSzarProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext prodContext = new ProdContext();

            Supplier supplier = new Supplier
            {
                City = "Lublin",
                CompanyName = "Gangsterzy inc",
                Street = "Lipowa",
                ZipCode = "30-698",
                BankAccountNumber = 113245654
            };

            Customer customer = new Customer
            {
                City = "Warszawa",
                CompanyName = "Filantropi inc",
                Street = "Miodowa",
                ZipCode = "30-442",
                Discount = 20
            };

            prodContext.Companies.Add(customer);
            prodContext.Companies.Add(supplier);

            prodContext.SaveChanges();
        }
    }
}
```

```
sqlite> select * from Companies;
1|Filantropi inc|Miodowa|Warszawa|30-442|Customer|20|
2|Gangsterzy inc|Lipowa|Lublin|30-698|Supplier|113245654|
3|Zbigniewex|Sosnowa|Wladyslawowo|60-748|Customer|30|
4|Polskie Sady|Gruszkowa|Konin|40-328|Supplier|415242154|
5|StoczinoPol|Zurawiowa|Gdynia|70-732|Customer|25|
6|StalPol|Twarda|Wroclaw|10-329|Supplier|314224233|
```

	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	Discount	BankAccountNumber
1	1	Filantropi inc	Miodowa	Warszawa	30-442	Customer	20	<null>
2	2	Gangsterzy inc	Lipowa	Lublin	30-698	Supplier	<null>	113245654
3	3	Zbigniewex	Sosnowa	Władysławowo	60-748	Customer	30	<null>
4	4	Polskie Sady	Gruszkowa	Konin	40-328	Supplier	<null>	415242154
5	5	StoczinoPol	Żurawiowa	Gdynia	70-732	Customer	25	<null>
6	6	StalPol	Twarda	Wrocław	10-329	Supplier	<null>	314224233

```
sqlite> select *
...> from Companies
...> where Discount is not null;
1|Filantropi inc|Miodowa|Warszawa|30-442|Customer|20|
3|Zbigniewex|Sosnowa|Wladyslawowo|60-748|Customer|30|
5|StoczinoPol|Zurawiowa|Gdynia|70-732|Customer|25|
```

7.b - TablePerType:

Zmodyfikowałem klasę Supplier:

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

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

        public int BankAccountNumber { get; set; }
        public virtual ICollection<Product> Products { get; set; }
    }
}
```


Zmodyfikowałem klasę Customer:

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

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

Zmodyfikowałem również klasę ProdContext usuwając metodę OnModelCreating dodaną w poprzednim punkcie:

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

namespace BSzarProductEF
{
    class ProdContext : DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder
options) => options.UseSqlite("DataSource=Product.db");
        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<InvoiceRecord> InvoiceRecords { get; set; }
        public DbSet<Company> Companies { get; set; }
    }
}
```

Po wstawieniu kilku dostawców i klientów do bazy, zauważyłem że dane nie są poprawnie przechowywane:

```
sqlite> select * from Companies;
1|StoczinoPol|Żurawiowa|Gdynia|70-732|Company|
2|StalPol|Twarda|Wrocław|10-329|Supplier|314224233
3|Wodnik|Szkłana|Łódź|57-152|Company|
4|TransMar|Wąska|Kraków|32-015|Supplier|114225232
```

	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	BankAccountNumber
1	1	StoczinoPol	Żurawiowa	Gdynia	70-732	Company	<null>
2	2	StalPol	Twarda	Wrocław	10-329	Supplier	314224233
3	3	Wodnik	Szkłana	Łódź	57-152	Company	<null>
4	4	TransMar	Wąska	Kraków	32-015	Supplier	114225232

Przy próbie zastosowania zasugerowanej w podlinkowanym poradniku konwencji użycia metody `OnModelCreating` w ciele klasy `ProdContext` o następującej treści:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Customer>().ToTable("Customers");
    modelBuilder.Entity<Supplier>().ToTable("Suppliers");
}
```

Baza przy próbie migracji rzuca następującym błędem:

```
The entity type 'Customer' cannot be mapped to a table because it is derived from 'Company'. Only base entity types can be mapped to a table.
```

7.b - TablePerClass:

Zmodyfikowałem klasę `ProdContext`:

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

namespace BSzarProductEF
{
    class ProdContext : DbContext
    {
        protected override void OnConfiguring(
            DbContextOptionsBuilder options) =>
            options.UseSqlite("DataSource=Product.db");
        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<InvoiceRecord> InvoiceRecords { get; set; }
    }
}
```

```

public DbSet<Company> Companies { get; set; }
protected override void OnModelCreating(
    modelBuilder modelBuilder)
{
    object p = modelBuilder.Entity<Supplier>().Map(m =>
    {
        m.MapInheritedProperties();
        m.ToTable("Suppliers");
    });

    modelBuilder.Entity<Customer>().Map(m =>
    {
        m.MapInheritedProperties();
        m.ToTable("Customers");
    });
}
}
}

```

W tym przypadku ponownie zadanie nie udało się uruchomić. Po przestudiowaniu dokumentacji okazało się że począwszy od wersji EF 3.0 podczas używania metody ToTable(), która jest wykorzystywana w ostatnich 2 zadaniach EntityFramework zgłasza wyjątek. Poniżej zamieszczam kopię rozdziału z dokumentacji.

ToTable na typ pochodny zgłasza wyjątek #11811 problemu śledzenia

Stare zachowanie

Przed EF Core 3.0, ToTable() wywoływane na typ pochodny będą ignorowane, ponieważ tylko strategia mapowania dziedziczenia był TPH, gdzie to nie jest prawidłowe.

Nowe zachowanie

Począwszy od EF Core 3.0 i w ramach przygotowań do ToTable() dodawania obsługi TPT i TPC w nowszej wersji, wywoływane na typ pochodny będzie teraz zgłosić wyjątek, aby uniknąć nieoczekiwanej zmiany mapowania w przyszłości.

Dlaczego

Obecnie nie jest prawidłowe mapowanie typu pochodnego do innej tabeli. Ta zmiana pozwala uniknąć zerwania w przyszłości, gdy staje się ważną rzeczą do zrobienia.

Środki zaradcze

Usuń wszelkie próby mapowania typów pochodnych do innych tabel.