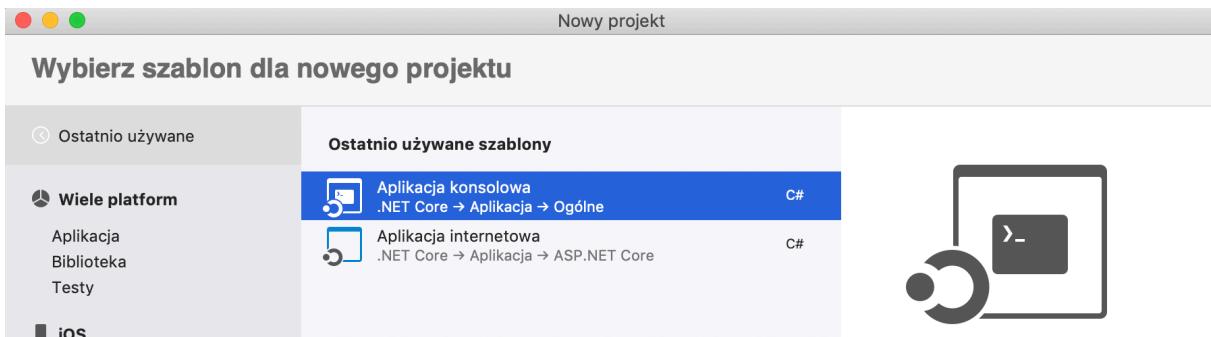


Zad 1

- a) Stwórz projekt typu ConsoleApplication .Net Core. Nazwij go INazwiskoProdutEF



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

```
using System;
namespace ANowaraProductEF
{
    public class Product
    {
        public int ProductID { get; set; }
        public String Name { get; set; }
        public int UnitsInStock { get; set; }
    }
}
```

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

```
using System;
using Microsoft.EntityFrameworkCore;
namespace ANowaraProductEF
{
    public class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
    }
}
```

e) Main

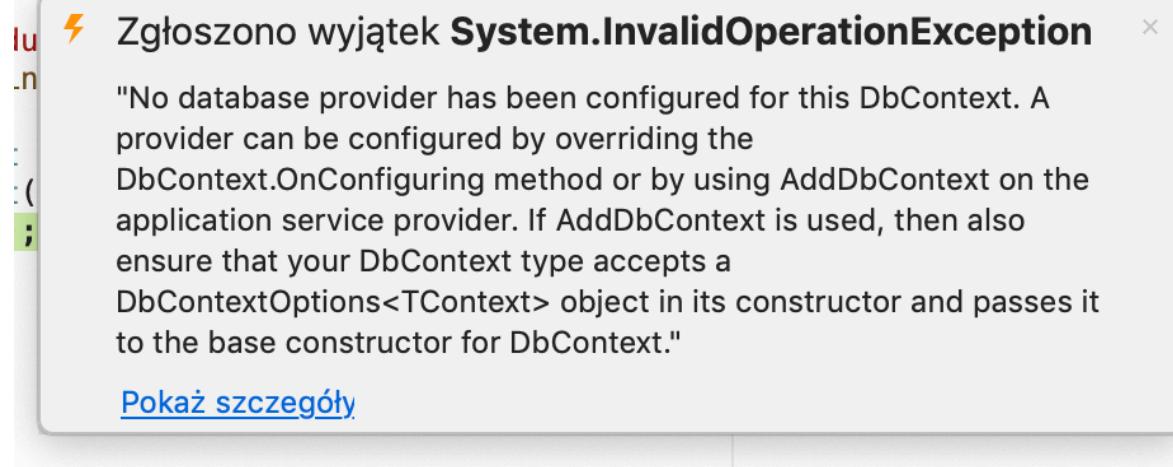
Stworzenie nowego produktu o nazwie podanej przez użytkownika i dodanie go do tabeli Products

```
using System;

namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Input product name:");
            String prodName = Console.ReadLine();

            Product product = new Product { Name = prodName };
            ProdContext context = new ProdContext();
            context.Products.Add(product);
            context.SaveChanges();
        }
    }
}
```

Otrzymanie wyjątek o braku zdefiniowanej bazy danych



Skonfigurowanie kontekstu przez wskazanie mu bazy, z której chcemy korzystać

```
using System;
using Microsoft.EntityFrameworkCore;
namespace ANowaraProductEF
{
    public class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }

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

Ponowne otrzymanie wyjątku

```
using System;

namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Input product name:");
            String prodName = Console.ReadLine();

            Product product = new Product();
            ProdContext context = new ProdContext();
            context.Products.Add(product);
            context.SaveChanges();
        }
    }
}
```

Zgłoszono wyjątek
Microsoft.EntityFrameworkCore.DbUpdateException
"An error occurred while updating the entries. See the inner exception for details."
[Pokaż szczegóły](#)

Próba dokonania migracji zakończona niepowodzeniem ze względu na brak pakietu EntityFrameworkCore.Design

```
Use "dotnet ef [command] --help" for more information about a command.
[MacBook-Pro-Agata:ANowaraProductEF agata$ dotnet ef migrations add InitialProductCreation
Build started...
Build succeeded.

Your startup project 'ANowaraProductEF' doesn't reference Microsoft.EntityFrameworkCore.Design. This package is required for the Entity Framework Core Tools to work. Ensure your startup project is correct, install the package, and try again
.
```

Po dodaniu do projektu tego pakietu, ponownie wpisuję komendę, aby stworzyć migracje modelu.

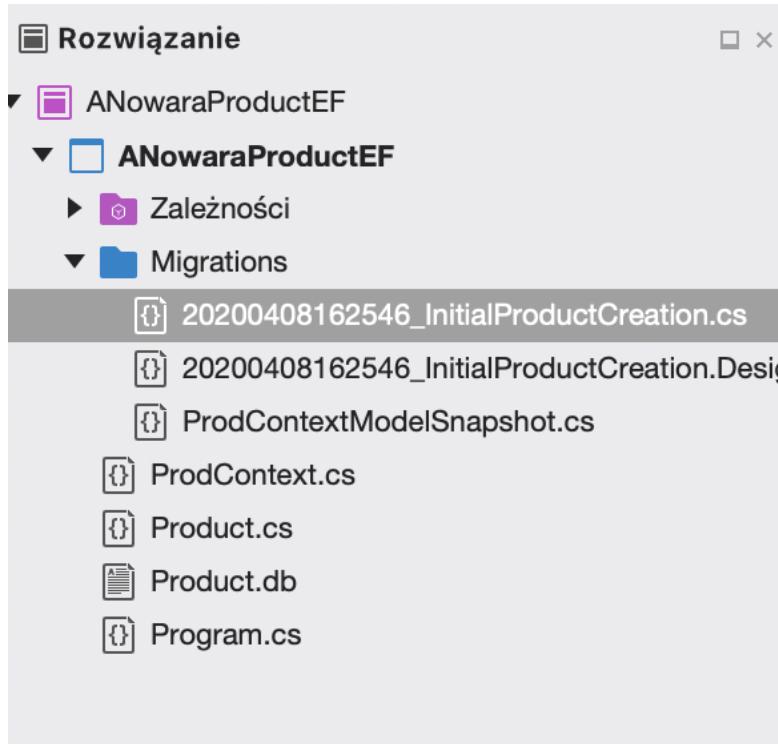
Tym razem operacja zakończyła się powodzeniem. Następnie wpisuję kolejną komendę, aby odzwierciedlić zmiany w modelu definiowane przez kolejne migracje

```
[MacBook-Pro-Agata:ANowaraProductEF agata$ dotnet ef migrations add InitialProductCreation
Build started...
Build succeeded.

Done. To undo this action, use 'ef migrations remove'
[MacBook-Pro-Agata:ANowaraProductEF agata$ dotnet ef database update
Build started...
Build succeeded.

Applying migration '20200408162546_InitialProductCreation'.
Done.
```

Po wyżej opisanych zmianach w strukturze projektu pojawił się katalog Migrations oraz plik bazy danych Product.db



Wprowadzeniu zmian w main w celu wypisywania wszystkich produktów w bazie oraz pobierania nowych produktów od użytkownika

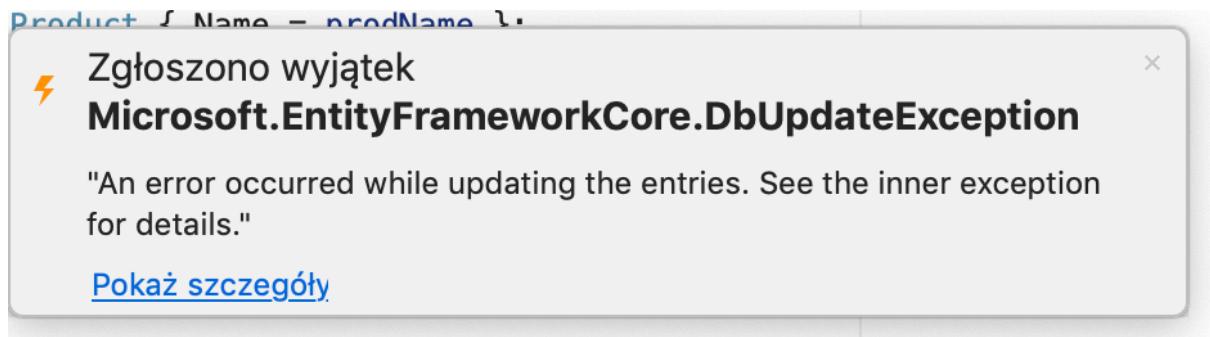
```
using System;
using System.Linq;

namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Input product name:");
            String prodName = Console.ReadLine();
            Product product = new Product { Name = prodName };
            ProdContext context = new ProdContext();
            context.Products.Add(product);
            context.SaveChanges(); ⚡

            IQueryable<Product> query = from p in context.Products
                                         orderby p.ProductID
                                         select p;

            foreach (var item in query)
            {
                Console.WriteLine("nazwa produktu:" + item.Name + "id produktu:" + item.ProductID);
            }
        }
    }
}
```

Ponowne otrzymanie wyjątku, związane z tym, że stworzona przeze mnie tabela Products nie została automatycznie skopiowana do miejsca, gdzie znajdują się pliki wykonywalne aplikacji w momencie jej uruchamiania



Po podłączeniu do klienta sqlite widać, że nasza baza Product.db faktycznie posiada tabelę Products:

```
[MacBook-Pro-Agata:ANowaraProductEF agata$ sqlite3 Product.db
SQLite version 3.24.0 2018-06-04 14:10:15
Enter ".help" for usage hints.
[sqlite]> .tables
Products
[sqlite]> Console.WriteLine("nazwa produktu:" + item.Name + "id produktu:" + item.Id)
[sqlite]>
```

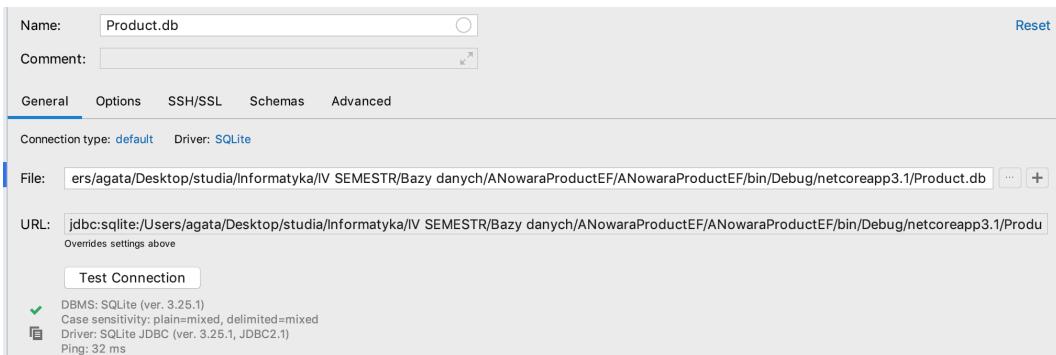
Wprowadzenie zmian we właściwościach pliku Product.db, aby umożliwić automatyczne kopiowanie pliku do odpowiedniego katalogu



Rezultat wykonania programu:

```
Input product name:
jabłko
nazwa produktu: truskawka id produktu: 1
nazwa produktu: banan id produktu: 2
nazwa produktu: jabłko id produktu: 3
```

Otwarcie stworzonej bazy danych w programie DataGrip:

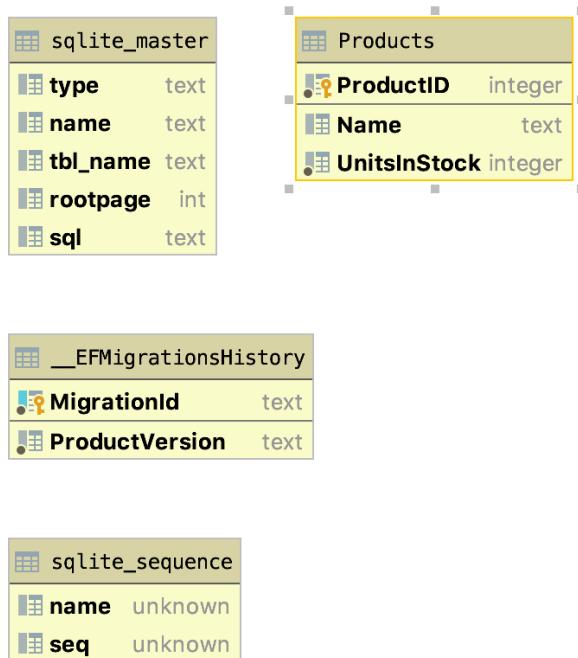


Zawartość tabeli Products widoczna w DataGrip:

The screenshot shows the DataGrip interface with two tabs: 'console [Product.db]' and 'main.Products [Product.db]'. The 'main.Products [Product.db]' tab is active, displaying the contents of the 'Products' table. The table has three columns: 'ProductID', 'Name', and 'UnitsInStock'. The data is as follows:

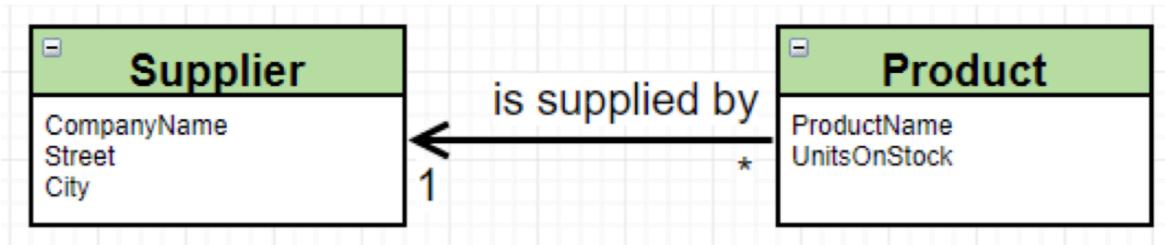
	ProductID	Name	UnitsInStock
1	1	truskawka	0
2	2	banan	0
3	3	jabłko	0

Obecna struktura bazy:



Zad 2

Polecenie: Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej:



Stworzenie nowej klasy o nazwie Supplier

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

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

Dokonanie zmian w klasie Product przez dodanie dodatkowego pola Supplier, będącego dostawcą danego produktu w celu stworzenia relacji przedstawionej na schemacie

```
using System;
using System.ComponentModel.DataAnnotations.Schema;

namespace ANowaraProductEF
{
    public class Product
    {
        public int ProductID { get; set; }
        public String Name { get; set; }
        public int UnitsInStock { get; set; }

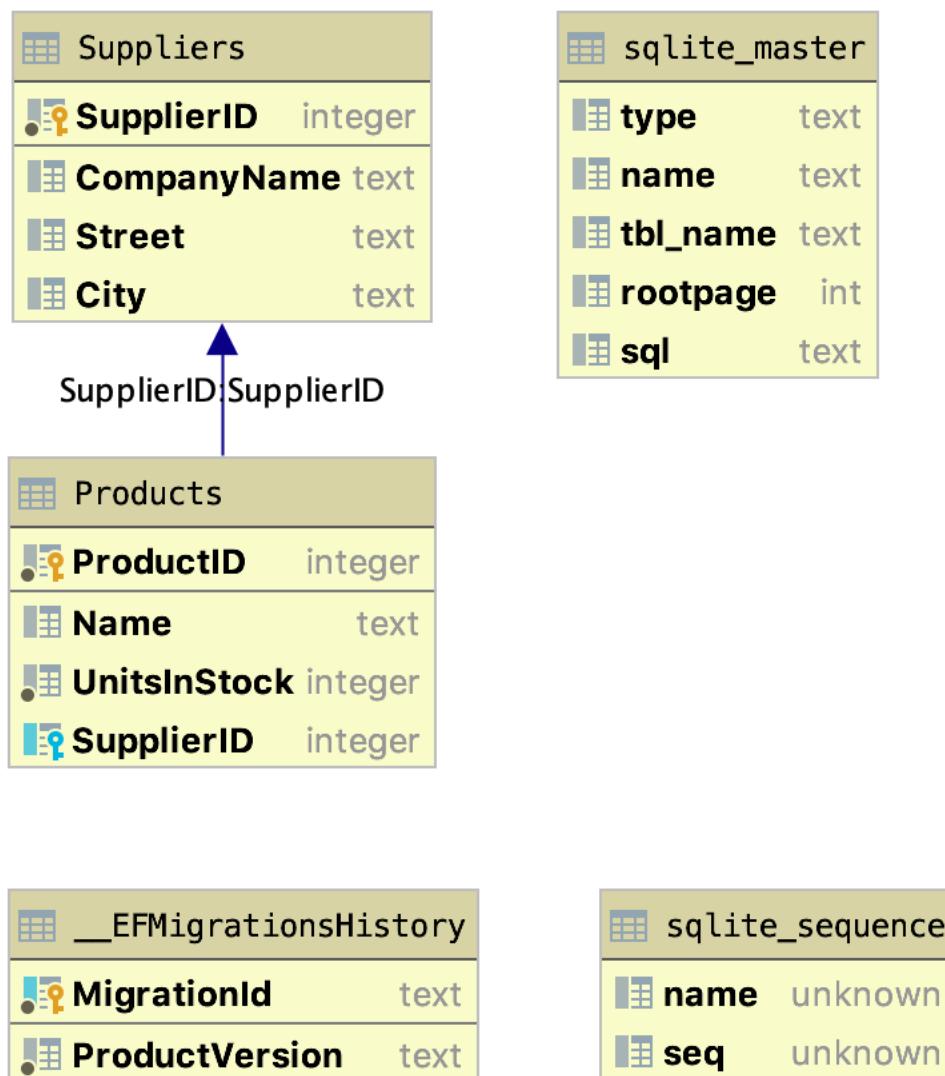
        public Supplier Supplier { get; set; }
    }
}
```

Do klasy ProdContext dodano DbSet<Supplier> reprezentujący wszystkich dostawców

```
using System;
using Microsoft.EntityFrameworkCore;
namespace ANowaraProductEF
{
    public class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

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

Schemat bazy danych



Dodanie do bazy dwóch produktów oraz jednego dostawcy

```
using System;
using System.Linq;

namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext context = new ProdContext();
            Console.WriteLine("Input first product name:");
            String prodName1 = Console.ReadLine();
            Product product1 = new Product { Name = prodName1 };

            Console.WriteLine("Input second product name:");
            String prodName2 = Console.ReadLine();
            Product product2 = new Product { Name = prodName2 };

            Console.WriteLine("Input company name:");
            String supplierName = Console.ReadLine();
            Supplier supplier = new Supplier { CompanyName = supplierName };

            context.Products.Add(product1);
            context.Products.Add(product2);
            context.Suppliers.Add(supplier);
            context.SaveChanges();
        }
    }
}
```

Przebieg wykonania powyższego programu

```
Input first product name:
malina
Input second product name:
mango
Input company name:
[Lidl]
```

Zrzut zawartości tabeli Products w programie DataGrip

The screenshot shows the DataGrip interface with three tabs at the top: 'console [Product.db]', 'main.__EFMigrationsHistory [Product.db]', and 'main.Products [Product.db]'. The 'main.Products' tab is selected. Below the tabs is a toolbar with various icons for navigation and operations. Underneath the toolbar is a section labeled '<Filter criteria>' with a dropdown arrow. The main area displays a table with four columns: ProductID, Name, UnitsInStock, and SupplierID. There are two rows of data:

	ProductID	Name	UnitsInStock	SupplierID
1	1	malina	0	<null>
2	2	mango	0	<null>

Zrzut zawartości tabeli Suppliers w programie Data Grip

The screenshot shows the Data Grip interface with four tabs at the top: 'console [Product.db]', 'main._EFMigrationsHistory [Product.db]', 'main.Products [Product.db]', and 'main.Suppliers [Product.db]'. The 'main.Suppliers' tab is active. Below the tabs is a toolbar with various icons. The main area displays a table with the following data:

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

Połączenie dwóch produktów znajdujących się w bazie z wprowadzonym ostatnio dostawcą za pomocą stworzonej relacji

```
using System;
using System.Linq;

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

            Supplier supplier = context.Suppliers.FirstOrDefault();

            IQueryables<Product> query = from p in context.Products
                                            orderby p.ProductID
                                            select p;

            foreach (var item in query)
            {
                context.Entry(item).Reference("Supplier").CurrentValue = supplier;
                context.SaveChanges();
                Console.WriteLine("nazwa produktu: " + item.Name + " id produktu: " + item.ProductID + " nazwa dostawcy: "
                    + item.Supplier.CompanyName + " id dostawcy: " + item.Supplier.SupplierID);
            }
        }
    }
}
```

Rezultat widoczny po wykonaniu programu

```
nazwa produktu: malina id produktu: 1 nazwa dostawcy: Lidl id dostawcy: 1
nazwa produktu: mango id produktu: 2 nazwa dostawcy: Lidl id dostawcy: 1
```

Prezentacja zmian dokonanych w bazie poprzez wykonanie powyżej zamieszczonego programu

The screenshot shows the Data Grip interface with four tabs at the top: 'console [Product.db]', 'main.Products [Product.db]', 'main.Suppliers [Product.db]', and 'main.sqlite'. The 'main.Products' tab is active. Below the tabs is a toolbar with various icons. The main area displays a table with the following data:

	ProductID	Name	UnitsInStock	SupplierID
1	1	malina	0	1
2	2	mango	0	1

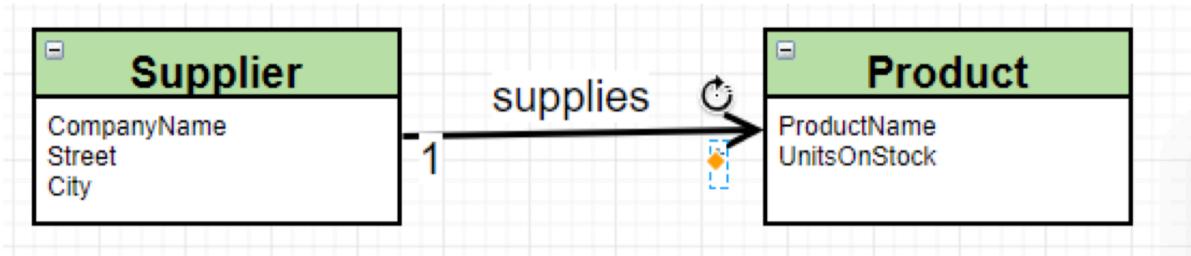
Prezentacja zawartości pozostałych tabel znajdujących się w bazie danych Product.db

main._EFMigrationsHistory [Product.db] ×	
Tx: Auto DB DDL	
Q < < 1 row > > ↻ + -	
MigrationId	ProductVersion
1 20200410092510_InitialProductCreation	3.1.3

main.sqlite_sequence [Product.db] ×	
Tx: Auto DB DDL	
Q < < 2 rows > > ↻ + -	
Filter criteria	
name	seq
1 Products	2
2 Suppliers	1

MigrationsHistory [Product.db] × main.Products [Product.db] × main.Suppliers [Product.db] × main.sqlite_sequence [Product.db] × main.sqlite_master [Product.db] ×			
Tx: Auto DB DDL			
Q < < 6 rows > > ↻ + -			
type	name	tbl_name	rootpage
1 table	_EFMigrationsHistory	_EFMigrationsHistory	2 CREATE TABLE "__EFMigrationsHistory" (
2 index	sqlite_autoindex__EFMigrationsHistory_1	_EFMigrationsHistory	3 <null>
3 table	Suppliers	Suppliers	4 CREATE TABLE "Suppliers" (
4 table	sqlite_sequence	sqlite_sequence	5 CREATE TABLE sqlite_sequence(name,seq)
5 table	Products	Products	6 CREATE TABLE "Products" (
6 index	TX_Products_SupplierID	Products	7 CREATE INDEX "TX_Products_SupplierID" ON "Products"

Zad 3 Odwróć relację zgodnie z poniższym schematem:



Usunięcie z klasy Product pola Supplier (dodanego w poprzednim zadaniu)

```
using System;
using System.ComponentModel.DataAnnotations.Schema;

namespace ANowaraProductEF
{
    public class Product
    {
        public int ProductID { get; set; }
        public String Name { get; set; }
        public int UnitsInStock { get; set; }
    }
}
```

Dodanie do klasy Supplier listy Products, czyli listy produktów dostarczonych przez danego dostawcę w celu stworzenia relacji przedstawionej na schemacie powyżej

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

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

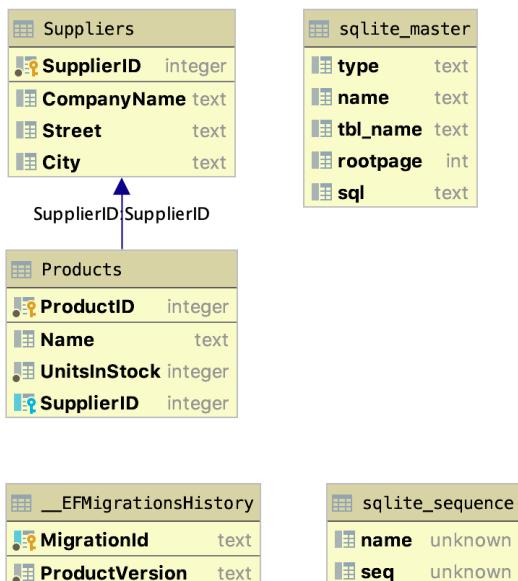
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public List<Product> Products { get; set; }
    }
}
```

Brak zmian w klasie ProdContext

```
using System;
using Microsoft.EntityFrameworkCore;
namespace AnowaraProductEF
{
    public class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

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

Prezentacja schematu bazy danych



Prezentacja automatycznie wygenerowanego skryptu z tworzenia tabeli Suppliers i Products

```
create table Suppliers
(
    SupplierID  INTEGER not null
        constraint PK_Suppliers
        primary key autoincrement,
    CompanyName TEXT,
    Street       TEXT,
    City         TEXT
);

create table Products
(
    ProductID   INTEGER not null
        constraint PK_Products
        primary key autoincrement,
    Name         TEXT,
    UnitsInStock INTEGER not null,
    SupplierID   INTEGER
        constraint FK_Products_Suppliers_SupplierID
        references Suppliers
        on delete restrict
);
```

Ponowne stworzenie produktów (trzech) oraz jednego dostawcy oraz dodanie ich do odpowiednich tabel.

```
using System;
using System.Linq;

namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext context = new ProdContext();
            Console.WriteLine("Input first product name:");
            String prodName1 = Console.ReadLine();
            Product product1 = new Product { Name = prodName1 };

            Console.WriteLine("Input second product name:");
            String prodName2 = Console.ReadLine();
            Product product2 = new Product { Name = prodName2 };

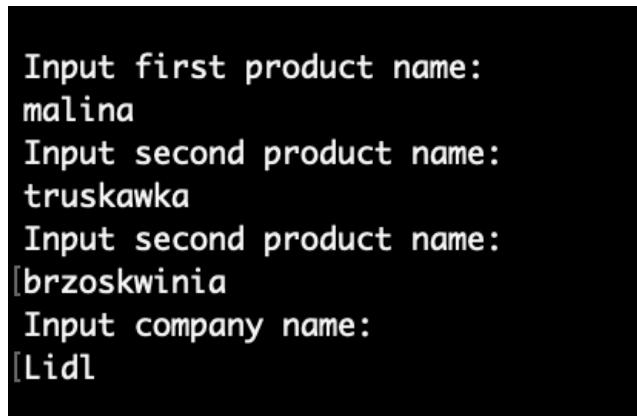
            Console.WriteLine("Input second product name:");
            String prodName3 = Console.ReadLine();
            Product product3 = new Product { Name = prodName3 };

            Console.WriteLine("Input company name:");
            String supplierName = Console.ReadLine();
            Supplier supplier = new Supplier { CompanyName = supplierName };

            context.Products.Add(product1);
            context.Products.Add(product2);
            context.Products.Add(product3);
            context.Suppliers.Add(supplier);
            context.SaveChanges();

        }
    }
}
```

Przebieg wykonania powyższego programu



```
Input first product name:
malina
Input second product name:
truskawka
Input second product name:
[brzoskwinia
Input company name:
[Lidl
```

Prezentacja zawartości tabeli Products w programie DataGrip

	ProductID	Name	UnitsInStock	SupplierID
1	1	malina	0	<null>
2	2	truskawka	0	<null>
3	3	brzoskwinia	0	<null>

Prezentacja zawartości tabeli Suppliers w programie DataGrip

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

Powiązanie wszystkich dostępnych w bazie produktów z wybranym dostawcą

```
using System.Linq;

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

            Supplier supplier = context.Suppliers.FirstOrDefault();

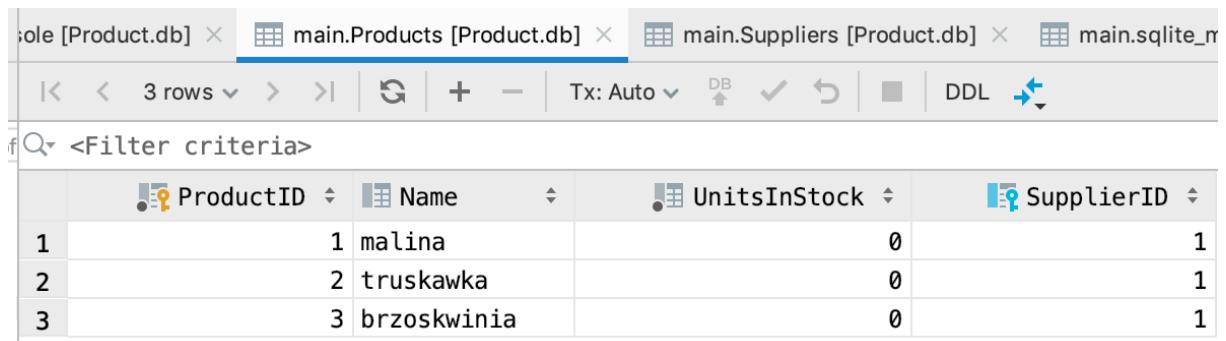
            IQueryable<Product> query = from p in context.Products
                                         orderby p.ProductID
                                         select p;

            foreach (var item in query)
            {
                supplier.Products.Add(item);
                context.SaveChanges();
                Console.WriteLine("nazwa produktu: " + item.Name + " id produktu: " + item.ProductID + " nazwa dostawcy: "
                    + supplier.CompanyName + " id dostawcy: " + supplier.SupplierID);
            }
        }
    }
}
```

Rezultat wykonania się powyższego programu – wypisanie produktu oraz jego dostawcy

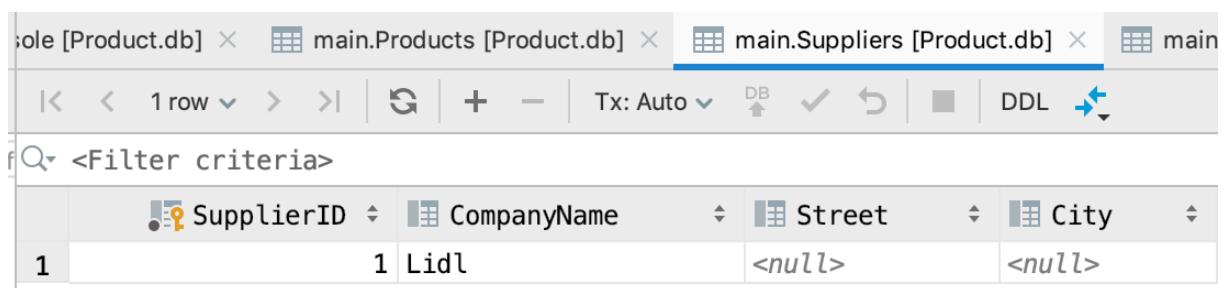
```
nazwa produktu: malina id produktu: 1 nazwa dostawcy: Lidl id dostawcy: 1
nazwa produktu: truskawka id produktu: 2 nazwa dostawcy: Lidl id dostawcy: 1
nazwa produktu: brzoskwinia id produktu: 3 nazwa dostawcy: Lidl id dostawcy: 1
```

Prezentacja zawartości tabeli Products wraz z przydzielonym do każdego produktu dostawcą



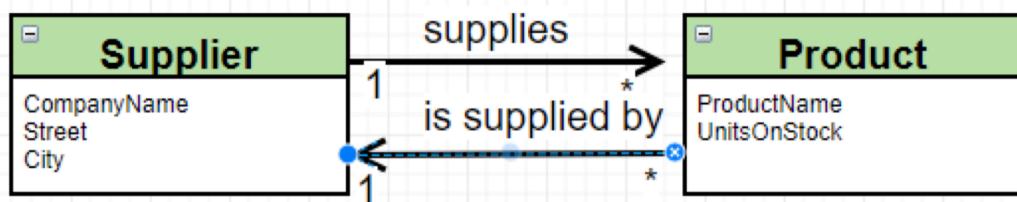
	ProductID	Name	UnitsInStock	SupplierID
1	1	malina	0	1
2	2	truskawka	0	1
3	3	brzoskwinia	0	1

Prezentacja tabeli Suppliers



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

Zad 4. Zamodeluj relacje dwustronną jak poniżej:



Ponowne dodanie pola Supplier do klasy Product, realizując tym samym relację *Product is supplied by Supplier*

```
using System;
using System.ComponentModel.DataAnnotations.Schema;

namespace ANowaraProductEF
{
    public class Product
    {
        public int ProductID { get; set; }
        public String Name { get; set; }
        public int UnitsInStock { get; set; }

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

Pozostawienie bez zmian klasy Supplier. Lista obiektów typu produkt odpowiada za realizację relacji *Supplier supplies Products*

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

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

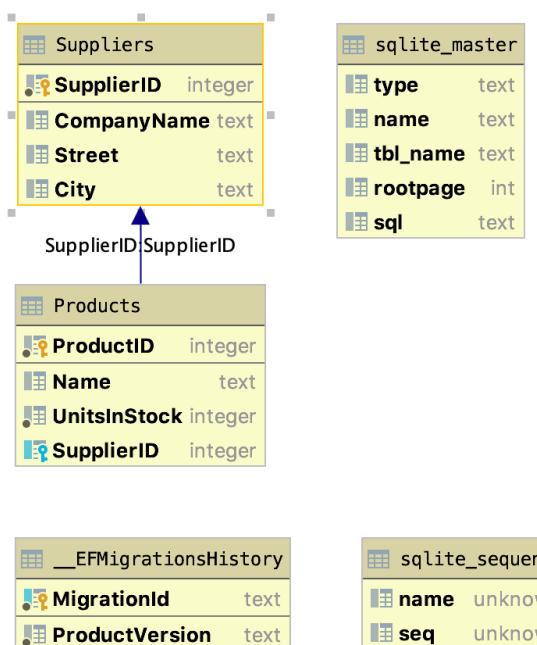
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public List<Product> Products { get; set; }
    }
}
```

Klasa ProdContext ponownie nie uległa zmianie

```
using System;
using Microsoft.EntityFrameworkCore;
namespace ANowaraProductEF
{
    public class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

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

Struktura bazy Product.db



Dodanie produktów oraz dostawcy i wzajemne ich powiązanie (dwustronna relacja)

```
using System;
using System.Linq;

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

            Console.WriteLine("Input first product name:");
            String prodName1 = Console.ReadLine();
            Product product1 = new Product { Name = prodName1 };

            Console.WriteLine("Input second product name:");
            String prodName2 = Console.ReadLine();
            Product product2 = new Product { Name = prodName2 };

            Console.WriteLine("Input second product name:");
            String prodName3 = Console.ReadLine();
            Product product3 = new Product { Name = prodName3 };

            Console.WriteLine("Input company name:");
            String supplierName = Console.ReadLine();
            Supplier supplier = new Supplier { CompanyName = supplierName };

            context.Products.Add(product1);
            context.Products.Add(product2);
            context.Products.Add(product3);
            context.Suppliers.Add(supplier);
            context.SaveChanges();

            IQueryable<Product> query = from p in context.Products
                                         orderby p.ProductID
                                         select p;

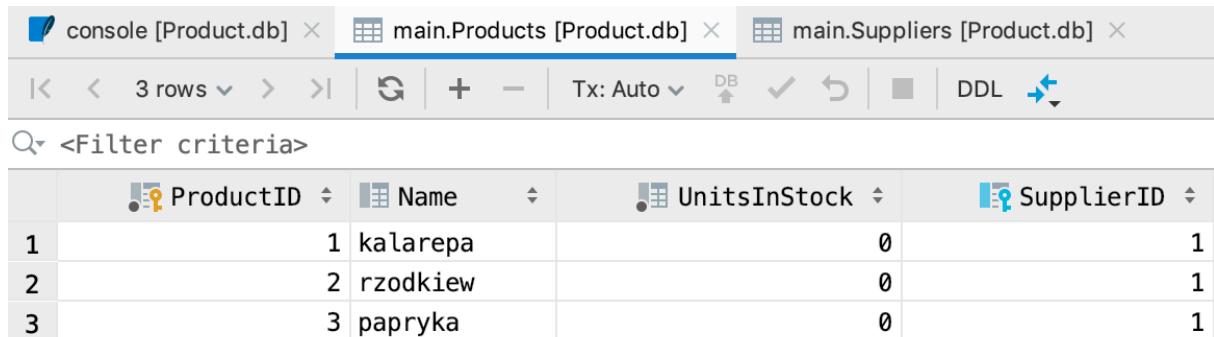
            Supplier sup = context.Suppliers.FirstOrDefault();

            foreach (var item in query)
            {
                context.Entry(item).Reference("Supplier").CurrentValue = sup;
                sup.Products.Add(item);
                context.SaveChanges();
                Console.WriteLine("nazwa produktu: " + item.Name + " id produktu: " + item.ProductID + " nazwa dostawcy: "
                    + item.Supplier.CompanyName + " id dostawcy: " + item.Supplier.SupplierID);
            }
        }
    }
}
```

Rezultat wykonania się powyższego programu – wypisanie produktu oraz jego dostawcy

```
Input first product name:
kalarepa
Input second product name:
rzodkiew
Input second product name:
[papryka
Input company name:
[Biedronka
nazwa produktu: kalarepa id produktu: 1 nazwa dostawcy: Biedronka id dostawcy: 1
[nazwa produktu: rzodkiew id produktu: 2 nazwa dostawcy: Biedronka id dostawcy: 1
nazwa produktu: papryka id produktu: 3 nazwa dostawcy: Biedronka id dostawcy: 1
[
```

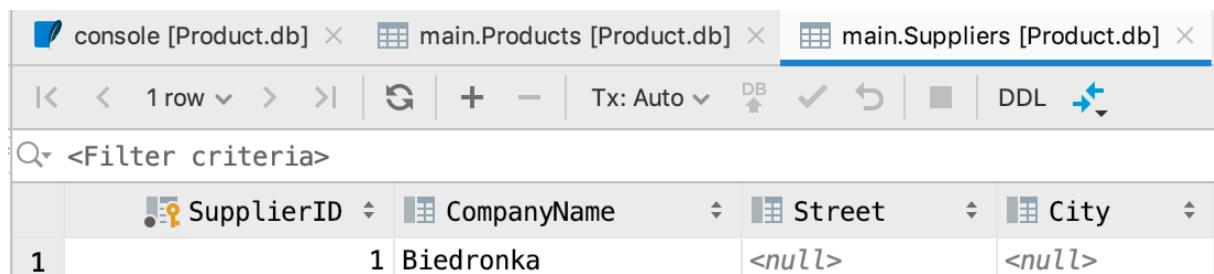
Prezentacja zawartości tabeli Products po dodaniu obiektów oraz wzajemnym powiązaniu ich



The screenshot shows a database interface with three tabs at the top: 'console [Product.db]', 'main.Products [Product.db]' (which is selected), and 'main.Suppliers [Product.db]'. Below the tabs is a toolbar with navigation icons (back, forward, search, etc.) and dropdown menus for 'Tx: Auto', 'DB', and 'DDL'. A 'Filter criteria' dropdown is open. The main area displays the 'Products' table with the following data:

	ProductID	Name	UnitsInStock	SupplierID
1	1	kalarepa	0	1
2	2	rzodkiew	0	1
3	3	papryka	0	1

Prezentacja zawartości tabeli Suppliers po dodaniu dostawcy



The screenshot shows a database interface with three tabs at the top: 'console [Product.db]', 'main.Products [Product.db]', and 'main.Suppliers [Product.db]' (selected). Below the tabs is a toolbar with navigation icons and dropdown menus for 'Tx: Auto', 'DB', and 'DDL'. A 'Filter criteria' dropdown is open. The main area displays the 'Suppliers' table with the following data:

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

Prezentacja automatycznie wygenerowanego skryptu z tworzenia tabeli Suppliers i Products

```
create table Suppliers
(
    SupplierID  INTEGER not null
        constraint PK_Suppliers
        primary key autoincrement,
    CompanyName TEXT,
    Street       TEXT,
    City         TEXT
);

create table Products
(
    ProductID   INTEGER not null
        constraint PK_Products
        primary key autoincrement,
    Name         TEXT,
    UnitsInStock INTEGER not null,
    SupplierID   INTEGER
        constraint FK_Products_Suppliers_SupplierID
        references Suppliers
        on delete restrict
);
```

Zad 5. Dodaj klase Category z property int CategoryID, String Name oraz listą produktow

```
- using System;
  using System.Collections.Generic;

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

Dodanie pola Category do klasy Product – wskazanie do jakiej kategorii należy dany produkt

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

namespace ANowaraProductEF
{
    public class Product
    {
        public int ProductID { get; set; }
        public String Name { get; set; }
        public int UnitsInStock { get; set; }

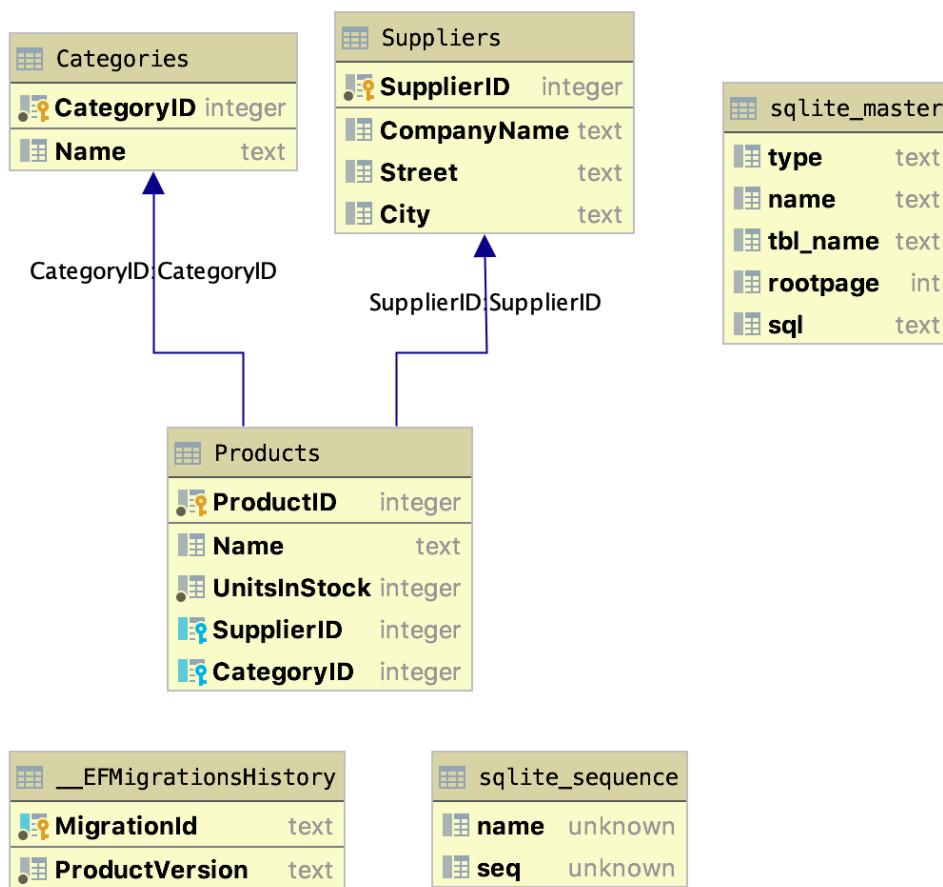
        public Supplier Supplier { get; set; }
        public Category Category { get; set; }
    }
}
```

Do klasy ProdContext dodano DbSet<Category> reprezentujący wszystkie kategorie

```
using System;
using Microsoft.EntityFrameworkCore;
namespace ANowaraProductEF
{
    public class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Company> Companies { get; set; }
        public DbSet<Category> Categories { get; set; }

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

Schemat bazy danych po dokonanych zmianach



Stworzenie czterech produktów, dwóch dostawców oraz dwóch kategorii (owoce i warzywa) oraz powiązanie ich. Na koniec wypisano kategorię, należący do niej produkt oraz jego dostawcę

```
using System;
using System.Linq;

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

            Supplier fruit_supplier = new Supplier { CompanyName = "Lidl" };
            context.Suppliers.Add(fruit_supplier);
            Category category_fruits = new Category { Name = "owoce" };
            context.Categories.Add(category_fruits);
            Product fruit1 = new Product { Name = "malina" };
            Product fruit2 = new Product { Name = "truskawka" };
            context.Products.Add(fruit1);
            context.Products.Add(fruit2);
            context.SaveChanges();

            IQueryable<Product> query = from p in context.Products
                                            orderby p.ProductID
                                            select p;

            foreach (var item in query)
            {
                context.Entry(item).Reference("Supplier").CurrentValue = fruit_supplier;
                fruit_supplier.Products.Add(item);
                context.Entry(item).Reference("Category").CurrentValue = category_fruits;
                category_fruits.Products.Add(item);
                context.SaveChanges();
                Console.WriteLine("nazwa kategorii " + item.Category.Name + " id kategorii " + item.Category.CategoryID
                    + " nazwa produktu: " + item.Name + " id produktu: " + item.ProductID + "nazwa dostawcy: "
                    + item.Supplier.CompanyName + " id dostawcy: " + item.Supplier.SupplierID);
            }

            Supplier veg_supplier = new Supplier { CompanyName = "Biedronka" };
            context.Suppliers.Add(veg_supplier);
            Category category_veg = new Category { Name = "warzywa" };
            context.Categories.Add(category_veg);
            Product veg1 = new Product { Name = "kalarepa" };
            Product veg2 = new Product { Name = "marchewka" };
            context.Products.Add(veg1);
            context.Products.Add(veg2);
            context.SaveChanges();

            foreach (var item in query)
            {
                if (item.Category == null && item.Supplier == null)
                {
                    context.Entry(item).Reference("Supplier").CurrentValue = veg_supplier;
                    veg_supplier.Products.Add(item);
                    context.Entry(item).Reference("Category").CurrentValue = category_veg;
                    category_veg.Products.Add(item);
                    context.SaveChanges();
                    Console.WriteLine("nazwa kategorii " + item.Category.Name + " id kategorii "
                        + item.Category.CategoryID + " nazwa produktu: " + item.Name + " id produktu: " + item.ProductID
                        + "nazwa dostawcy: "
                        + item.Supplier.CompanyName + " id dostawcy: " + item.Supplier.SupplierID);
                }
            }
        }
    }
}
```

Rezultat widoczny po wykonaniu programu

```
nazwa kategorii owoce id kategorii 1 nazwa produktu: malina id produktu: 1nazwa dostawcy: Lidl id dostawcy: 1
nazwa kategorii owoce id kategorii 1 nazwa produktu: truskawka id produktu: 2nazwa dostawcy: Lidl id dostawcy: 1
nazwa kategorii warzywa id kategorii 2 nazwa produktu: marchewka id produktu: 3nazwa dostawcy: Biedronka id dostawcy: 2
nazwa kategorii warzywa id kategorii 2 nazwa produktu: kalarepa id produktu: 4nazwa dostawcy: Biedronka id dostawcy: 2
```

Prezentacja zawartości tabeli Products po dodaniu do niej produktów oraz powiązaniu ich z dostawca i kategorią

The screenshot shows a database management interface with two tabs: 'console [Product.db]' and 'main.Products [Product.db]'. The 'main.Products' tab is active, displaying a table with four columns: ProductID, Name, UnitsInStock, SupplierID, and CategoryID. The data is as follows:

	ProductID	Name	UnitsInStock	SupplierID	CategoryID
1	1	malina	0	1	1
2	2	truskawka	0	1	1
3	3	marchewka	0	2	2
4	4	kalarepa	0	2	2

Pokazanie dostępnych kategorii

The screenshot shows a database management interface with one tab: 'main.Categories [Product.db]'. The table has two columns: CategoryID and Name. The data is as follows:

	CategoryID	Name
1	1	owoce
2	2	warzywa

Pokazanie dostępnych dostawców

The screenshot shows a database management interface with one tab: 'main.Suppliers [Product.db]'. The table has four columns: SupplierID, CompanyName, Street, and City. The data is as follows:

	SupplierID	CompanyName	Street	City
1	1	Lidl	<null>	<null>
2	2	Biedronka	<null>	<null>

Wydobycie listy produktów należących do danej kategorii

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

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

            IQueryables> query = from c in context.Categories.Include(c=> c.Products)
                                  orderby c.CategoryID
                                  select c;

            foreach (var category in query)
            {
                Console.WriteLine("Nazwa kategorii " + category.Name);
                foreach (var item in category.Products.ToList())
                {
                    Console.WriteLine("-" + item.Name);
                }
            }
        }
    }
}
```

Rezultat widoczny po wykonaniu się programu – wypisanie nazwy kategorii wraz z listą produktów

```
Nazwa kategorii: owoce
-malina
-truskawka
Nazwa kategorii: warzywa
-marchewka
-kalarepa
```

Wydobycie kategorii, do której należy wybrany produkt

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

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

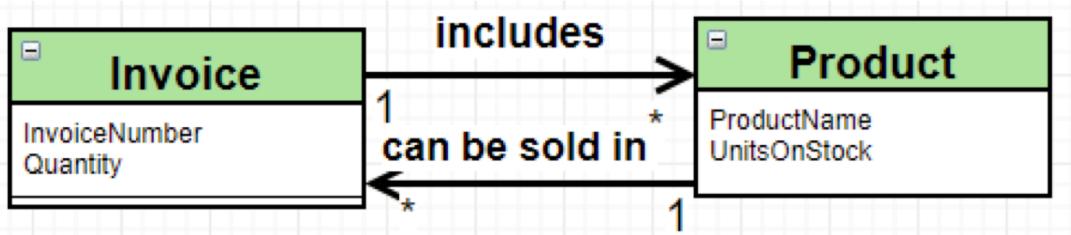
            IQueryables<Product> query = from p in context.Products
                                            orderby p.ProductID
                                            select p;

            foreach (var item in query)
            {
                context.Entry(item).Reference(item => item.Category).Load();
                if (item.Category != null)
                {
                    Console.WriteLine("nazwa produktu: " + item.Name + " nazwa kategorii: " + item.Category.Name);
                }
            }
        }
    }
}
```

Rezultat widoczny po wykonaniu się programu – wypisanie wszystkich produktów wraz z kategorią do której należą

```
nazwa produktu: malina nazwa kategorii: owoce
nazwa produktu: truskawka nazwa kategorii: owoce
nazwa produktu: marchewka nazwa kategorii: warzywa
nazwa produktu: kalarepa nazwa kategorii: warzywa
```

Zad 6. Zamodeluj relacje wiele-do-wielu, jak poniżej:



Dokonano próby wykonania relacji wiele do wielu przez umieszczenie `ICollection<Invoice>` w klasie `Product` oraz `Icollection<Product>` w klasie `Invoice`

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

namespace ANowaraProductEF
{
    public class Product
    {
        public Product()
        {
            Invoices = new HashSet<Invoice>();
        }
        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 virtual ICollection<Invoice> Invoices { get; set; }
    }
}
```

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

namespace ANowaraProductEF
{
    public class Invoice
    {
        public Invoice()
        {
            Products = new HashSet<Product>();
        }
        public int InvoiceID { get; set; }
        public int InvoiceNumber { get; set; }
        public int Quantity { get; set; }

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

Próba zakończyła się niepowodzeniem ze względu na pokazanie się poniższego wyjątku

```
ANowaraProductEF — -bash — 80x24
[net: set] at Microsoft.EntityFrameworkCore.Infrastructure.Internal.InfrastructureExtensions.GetService[TService](IInfrastructure`1 accessor)
[net: set] at Microsoft.EntityFrameworkCore.Infrastructure.AccessorExtensions.GetService[TService](IInfrastructure`1 accessor)
[net: set] at Microsoft.EntityFrameworkCore.Design.Internal.DbContextOperations.CreateContext(Func`1 factory)
[net: set] at Microsoft.EntityFrameworkCore.Design.Internal.DbContextOperations.CreateContext(String contextType)
[net: set] at Microsoft.EntityFrameworkCore.Design.Internal.MigrationsOperations.AddMigration(String name, String outputDir, String contextType)
[net: set] at Microsoft.EntityFrameworkCore.Design.OperationExecutor.AddMigrationImpl(String name, String outputDir, String contextType)
[net: set] at Microsoft.EntityFrameworkCore.Design.OperationExecutor.AddMigration.<>c__DisplayClass0_0.<.<ctor>b__0()
[net: set] at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.<>c__DisplayClass3_0`1.<Execute>b__0()
[net: set] at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.Execute(Action action)
Unable to determine the relationship represented by navigation property 'Invoice.Products' of type 'ICollection<Product>'. Either manually configure the relationship, or ignore this property using the '[NotMapped]' attribute or by using 'EntityTypeBuilder.Ignore' in 'OnModelCreating'.
MacBook-Pro-Agata:ANowaraProductEF agata$
```

W tej sytuacji w celu zrealizowania powyższej relacji stworzono nową klasę ProductInvoice, służącą do przechowywania relacji między Product a Invoice

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace ANowaraProductEF
{
    public class ProductInvoice
    {
        public int ProductID { get; set; }
        public int InvoiceID { get; set; }
        public Product Product { get; set; }
        public Invoice Invoice { get; set; }
    }
}
```

Dodatkowo w klasie Product umieszczono listę obiektów typu ProductInvoice

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

namespace ANowaraProductEF
{
    public class Product
    {
        public Product()
        {
            ProductInvoices = new List<ProductInvoice>();
        }

        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<ProductInvoice> ProductInvoices { get; set; }
    }
}
```

Taką samą listę dodano również do klasy Invoice

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

namespace ANowaraProductEF
{
    public class Invoice
    {
        public Invoice()
        {
            ProductInvoices = new List<ProductInvoice>();
        }

        public int InvoiceID { get; set; }
        public int InvoiceNumber { get; set; }
        public int Quantity { get; set; }

        public List<ProductInvoice> ProductInvoices { get; set; }
    }
}
```

Z kolei do klasy `ProductContext` dodano `DbSet<Invoice>` - zbiór faktur oraz `DbSet<ProductInvoice>` - zbiór relacji między produktami, a fakturami

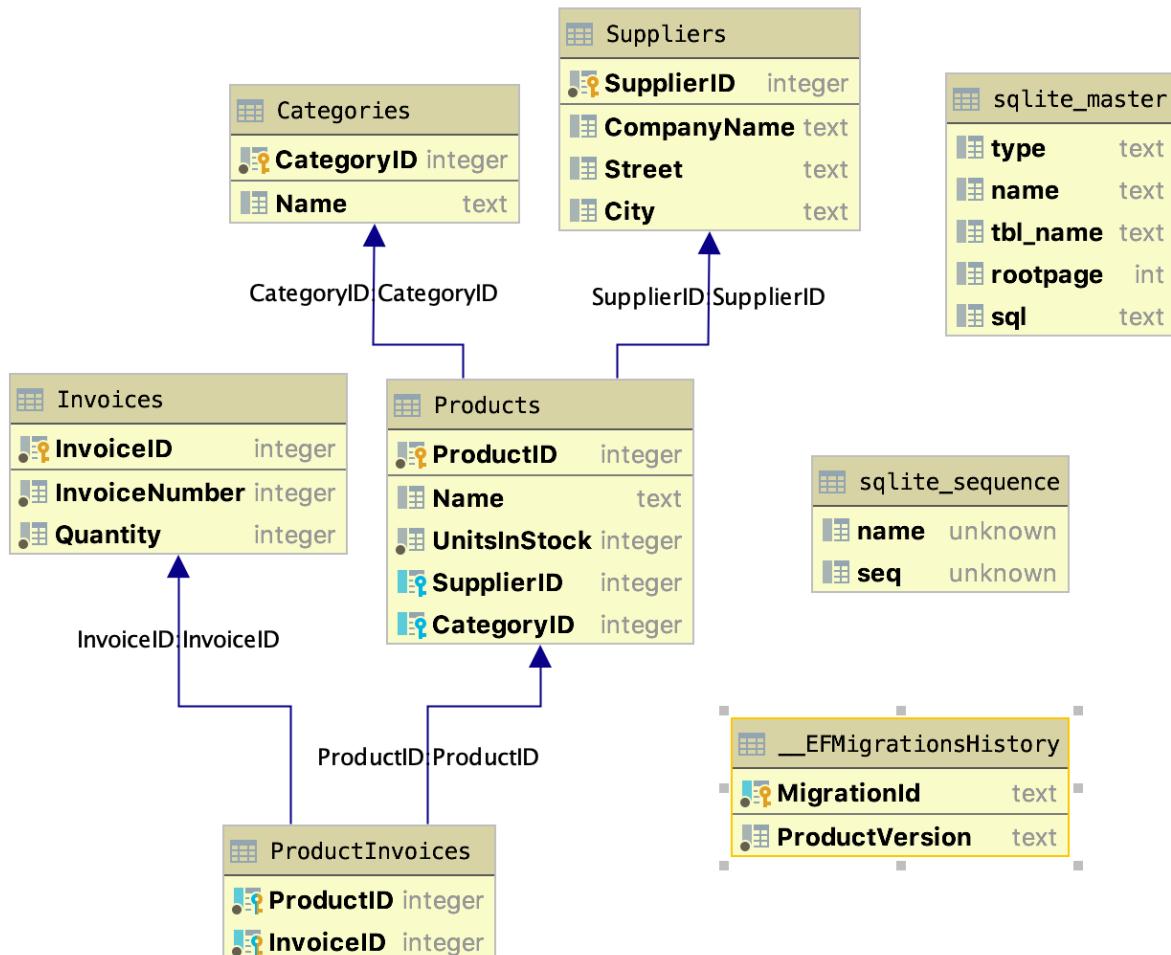
Dodano również metodę `OnModelCreating` w celu obsługi zadanej relacji

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

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<ProductInvoice>()
            .HasKey(pi => new { pi.ProductID, pi.InvoiceID });
        modelBuilder.Entity<ProductInvoice>()
            .HasOne(pi => pi.Product)
            .WithMany(p => p.ProductInvoices)
            .HasForeignKey(bc => bc.ProductID);
        modelBuilder.Entity<ProductInvoice>()
            .HasOne(pi => pi.Invoice)
            .WithMany(i => i.ProductInvoices)
            .HasForeignKey(pi => pi.InvoiceID);
    }

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

Prezentacja schematu bazy danych



Stworzono kilka produktów, dwie kategorie, dwóch dostawców oraz „sprzedano” dane produkty w dwóch powstały transakcjach

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

namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {

            ProdContext context = new ProdContext();

            Supplier fruit_supplier = new Supplier { CompanyName = "Lidl" };
            Supplier veg_supplier = new Supplier { CompanyName = "Biedronka" };
            context.Suppliers.Add(fruit_supplier);
            context.Suppliers.Add(veg_supplier);

            Category category_fruits = new Category { Name = "owoce" };
            Category category_veg = new Category { Name = "warzywa" };
            context.Categories.Add(category_fruits);
            context.Categories.Add(category_veg);

            Product fruit1 = new Product { Name = "malina" };
            Product fruit2 = new Product { Name = "truskawka" };
            Product veg1 = new Product { Name = "kalarepa" };
            Product veg2 = new Product { Name = "marchewka" };
            context.Products.Add(fruit1);
            context.Products.Add(fruit2);
            context.Products.Add(veg1);
            context.Products.Add(veg2);

            fruit1.Category = category_fruits;
            category_fruits.Products.Add(fruit1);
            fruit2.Category = category_fruits;
            category_fruits.Products.Add(fruit2);

            veg1.Category = category_veg;
            category_veg.Products.Add(veg1);
            veg2.Category = category_veg;
            category_veg.Products.Add(veg2);

            fruit1.Supplier = fruit_supplier;
            Invoice invoice1 = new Invoice { InvoiceNumber = 1, Quantity = 2 };
            Invoice invoice2 = new Invoice { InvoiceNumber = 2, Quantity = 5 };
            context.Invoices.Add(invoice1);
            context.Invoices.Add(invoice2);

            ProductInvoice prodinv1 = new ProductInvoice { Product = fruit1, Invoice = invoice1 };
            ProductInvoice prodinv2 = new ProductInvoice { Product = fruit2, Invoice = invoice2 };
            ProductInvoice prodinv3 = new ProductInvoice { Product = veg1, Invoice = invoice1 };
            ProductInvoice prodinv4 = new ProductInvoice { Product = veg2, Invoice = invoice2 };

            context.ProductInvoices.Add(prodinv1);
            context.ProductInvoices.Add(prodinv2);
            context.ProductInvoices.Add(prodinv3);
            context.ProductInvoices.Add(prodinv4);

            fruit1.ProductInvoices.Add(prodinv1);
            fruit2.ProductInvoices.Add(prodinv2);
            veg1.ProductInvoices.Add(prodinv3);
            veg2.ProductInvoices.Add(prodinv4);
            context.SaveChanges();
        }
    }
}
```

Rezultat wprowadzonych zmian:

main Products [Product.db]

	ProductID	Name	UnitsInStock	SupplierID	CategoryID
1	1	malina	0	1	1
2	2	truskawka	0	1	1
3	3	kalarepa	0	2	2
4	4	marchewka	0	2	2

main Suppliers [Product.db]

	SupplierID	CompanyName	Street	City
1	1	Lidl	<null>	<null>
2	2	Biedronka	<null>	<null>

main.Invoices [Product.db]

	InvoiceID	InvoiceNumber	Quantity
1	1	1	2
2	2	2	5

main.ProductInvoices [Product.db]

	ProductID	InvoiceID
1	1	1
2	2	2
3	3	1
4	4	2

The screenshot shows a database interface with a query editor at the top containing the following SQL code:

```

select P.Name, C.Name, S.CompanyName, I.InvoiceNumber
from Products P
join Categories C on p.CategoryID = C.CategoryID
join Suppliers S on P.SupplierID = S.SupplierID
join ProductInvoices PI on P.ProductID = PI.ProductID
join Invoices I on PI.InvoiceID = I.InvoiceID

```

Below the query editor is a results pane titled "Result 2". It displays a table with four columns: Name, CompanyName, and InvoiceNumber. The table has 4 rows of data:

	Name	CompanyName	InvoiceNumber
1	malina	Lidl	1
2	truskawka	Lidl	2
3	kalarepa	Biedronka	1
4	marchewka	Biedronka	2

Prezentacja automatycznie wygenerowanego skryptu z tworzenia tabeli Invoices, Products oraz ProductInvoices

```

create table Invoices
(
    InvoiceID      INTEGER not null
        constraint PK_Invoices
            primary key autoincrement,
    InvoiceNumber  INTEGER not null,
    Quantity       INTEGER not null
);

create table Products
(
    ProductID      INTEGER not null
        constraint PK_Products
            primary key autoincrement,
    Name           TEXT,
    UnitsInStock   INTEGER not null,
    SupplierID     INTEGER
        constraint FK_Products_Suppliers_SupplierID
            references Suppliers
            on delete restrict,
    CategoryID     INTEGER
        constraint FK_Products_Categories_CategoryID
            references Categories
            on delete restrict
);

```

```

create table ProductInvoices
(
    ProductID INTEGER not null
        constraint FK_ProductInvoices_Products_ProductID
            references Products
            on delete cascade,
    InvoiceID INTEGER not null
        constraint FK_ProductInvoices_Invoices_InvoiceID
            references Invoices
            on delete cascade,
    constraint PK_ProductInvoices
        primary key (ProductID, InvoiceID)
);

create index IX_ProductInvoices_InvoiceID
    on ProductInvoices (InvoiceID);

create index IX_Products_CategoryID
    on Products (CategoryID);

create index IX_Products_SupplierID
    on Products (SupplierID);

create table __EFMigrationsHistory
(
    MigrationId TEXT not null
        constraint PK__EFMigrationsHistory
            primary key,
    ProductVersion TEXT not null
);

```

main.sqlite_sequence [Product.db] ×

4 rows ▾ > | Tx: Auto | DDL

Q <Filter criteria>

	name	seq
1	Categories	2
2	Invoices	2
3	Suppliers	2
4	Products	4

Program mający na celu pokazanie produktów sprzedanych w ramach danej faktury

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

namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext context = new ProdContext();
            IQueryables<Invoice> query = from pi in context.Invoices.Include(pi => pi.ProductInvoices)
                                            orderby pi.InvoiceID
                                            select pi;
            foreach (var invoice in query)
            {
                Console.WriteLine(invoice.InvoiceNumber + ": ");
                foreach (var item in invoice.ProductInvoices.ToList())
                {
                    context.Entry(item).Reference(item => item.Product).Load();
                    Console.WriteLine(item.Product.Name);
                }
            }
        }
    }
}
```

Rezultat widoczny po wykonaniu się powyższego programu

```
Numert transakcji: 1
Produkty sprzedane w ramach transakcji:
malina
kalarepa
Numert transakcji: 2
Produkty sprzedane w ramach transakcji:
truskawka
marchewka
```

Program mający na celu pokazanie faktur w ramach, których był sprzedany dany produkt

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

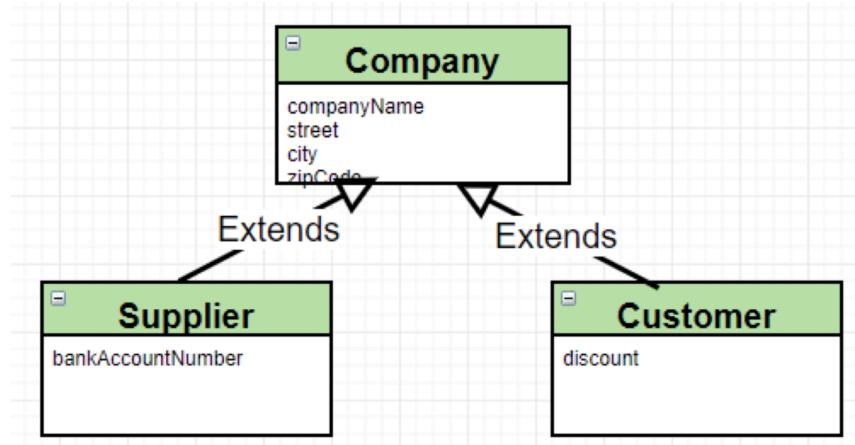
namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext context = new ProdContext();
            IQueryables<Product> query = from p in context.Products.Include(p => p.ProductInvoices)
                                            orderby p.ProductID
                                            select p;
            foreach (var product in query)
            {
                Console.WriteLine("Nazwa produktu: " + product.Name);
                Console.WriteLine("Transakcja w ramach której produkt został sprzedany:");
                foreach (var item in product.ProductInvoices.ToList())
                {
                    context.Entry(item).Reference(item => item.Invoice).Load();
                    Console.WriteLine(item.Invoice.InvoiceNumber);
                }
            }
        }
    }
}
```

Rezultat widoczny po wykonaniu się powyższego programu

```
Nazwa produktu: malina
Transakcja w ramach której produkt został sprzedany:
1
Nazwa produktu: truskawka
Transakcja w ramach której produkt został sprzedany:
2
Nazwa produktu: kalarepa
Transakcja w ramach której produkt został sprzedany:
1
Nazwa produktu: marchewka
Transakcja w ramach której produkt został sprzedany:
2
```

Zad 7. Dziedziczenie

- a) Dodaj i pobierz z bazy kilka firm obu rodzajów stosując strategie mapowania dziedziczenia TablePerHierarchy



Stworzenie klasy Company z zadanimi polami

```
using System;
namespace ANowaraProductEF
{
    public class Company
    {
        public int CompanyID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public int ZipCode { get; set; }
    }
}
```

Stworzenie klasy Customer dziedziczącej po klasie Company, posiadającej jedno dodatkowe pole tj. Discount

```
using System;
namespace ANowaraProductEF
{
    public class Customer : Company
    {
        public float Discount { get; set; }
    }
}
```

Zmodyfikowanie klasy Supplier, tak aby dziedziczyła po Company. Z klasy Supplier usunięte zostały wszystkie pola znajdujące się w Company oraz dodane zostało dodatkowe pole tj. BankAccountNumber.

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

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

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

Zmodyfikowanie metody OnModelCreating w klasie ProdContext oraz zastąpienie DbSet<Supplier> przez DbSet<Company>

```
using System;
using Microsoft.EntityFrameworkCore;
namespace ANowaraProductEF
{
    public class ProdContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Company> Companies { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<ProductInvoice> ProductInvoices { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<ProductInvoice>()
                .HasKey(pi => new { pi.ProductID, pi.InvoiceID });
            modelBuilder.Entity<ProductInvoice>()
                .HasOne(pi => pi.Product)
                .WithMany(p => p.ProductInvoices)
                .HasForeignKey(bc => bc.ProductID);
            modelBuilder.Entity<ProductInvoice>()
                .HasOne(pi => pi.Invoice)
                .WithMany(i => i.ProductInvoices)
                .HasForeignKey(pi => pi.InvoiceID);
            modelBuilder.Entity<Customer>();
            modelBuilder.Entity<Supplier>();

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

Dodanie dwóch dostawców i dwóch klientów z uzupełnionymi wszystkimi danymi

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

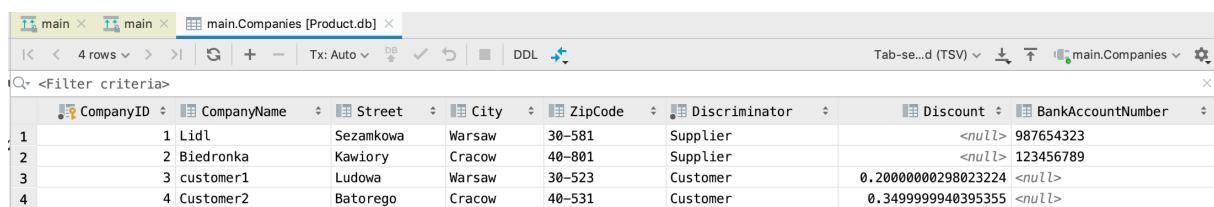
namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {

            ProdContext context = new ProdContext();

            Supplier fruit_supplier = new Supplier {
                CompanyName = "Lidl",
                City = "Warsaw",
                Street = "Sezamkowa",
                ZipCode = "30-581",
                BankAccountNumber = "987654323"};
            Supplier veg_supplier = new Supplier {
                CompanyName = "Biedronka",
                City = "Cracow",
                Street = "Kawiory",
                ZipCode = "40-801",
                BankAccountNumber = "123456789"};
            context.Add(fruit_supplier);
            context.Add(veg_supplier);

            Customer customer1 = new Customer
            {
                CompanyName = "customer1",
                City = "Warsaw",
                Street = "Ludowa",
                ZipCode = "30-523",
                Discount = 0.2f
            };
            Customer customer2 = new Customer
            {
                CompanyName = "Customer2",
                City = "Cracow",
                Street = "Batorego",
                ZipCode = "40-531",
                Discount = 0.35f
            };
            context.Add(customer1);
            context.Add(customer2);
        }
    }
}
```

Prezentacja tabeli Companies wraz z zawartością stworzoną w powyższym programie



	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	Discount	BankAccountNumber
1	1	Lidl	Sezamkowa	Warsaw	30-581	Supplier	<null>	987654323
2	2	Biedronka	Kawiory	Cracow	40-801	Supplier	<null>	123456789
3	3	customer1	Ludowa	Warsaw	30-523	Customer	0.2000000298023224	<null>
4	4	Customer2	Batorego	Cracow	40-531	Customer	0.349999940395355	<null>

Automatycznie wygenerowany skrypt tabeli Companies

```
create table Companies
(
    CompanyID      INTEGER not null
        constraint PK_Companies
        primary key autoincrement,
    CompanyName    TEXT,
    Street         TEXT,
    City           TEXT,
    ZipCode        TEXT,
    Discriminator TEXT    not null,
    Discount       REAL,
    BankAccountNumber TEXT
);
```

Pobranie z bazy danych o wszystkich obiektach typu Company z podziałem na Supplier i Customer

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

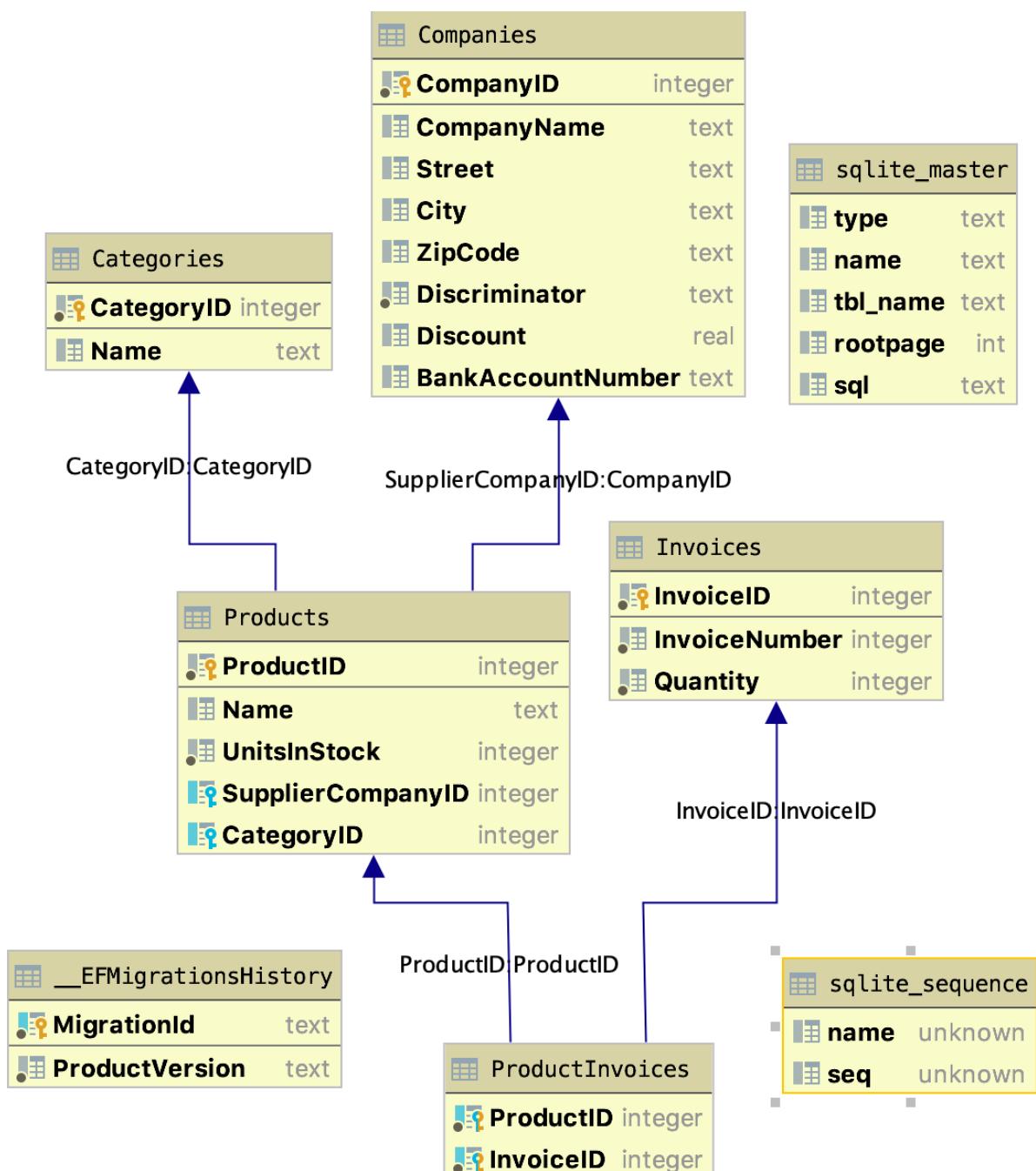
namespace ANowaraProductEF
{
    class Program
    {
        static void Main(string[] args)
        {
            ProdContext context = new ProdContext();
            var suppliers = context.Companies.OfType<Supplier>().ToList();
            var customers = context.Companies.OfType<Customer>().ToList();
            Console.WriteLine("Suppliers details: ");
            foreach (var s in suppliers)
            {
                Console.WriteLine("name: " + s.CompanyName + " city: " + s.City
                    + " street: " + s.Street + " zipcode: "
                    + s.ZipCode + " bank account: " + s.BankAccountNumber);
            }

            Console.WriteLine("Customers details: ");
            foreach (var c in customers)
            {
                Console.WriteLine("name: " + c.CompanyName+ " city: " + c.City
                    + " street: " + c.Street + " zipcode "
                    + c.ZipCode + " discount " + c.Discount);
            }
        }
    }
}
```

Rezultat wykonania się powyższego programu

```
Suppliers details:  
name: Lidl city: Warsaw street: Sezamkowa zipcode: 30-581 bank account: 987654323  
name: Biedronka city: Cracow street: Kawiory zipcode: 40-801 bank account: 123456789  
Customers details:  
name: customer1 city: Warsaw street: Ludowa zipcode 30-523 discount 0,2  
name: Customer2 city: Cracow street: Batorego zipcode 40-531 discount 0,35
```

Schemat bazy danych po dokonanych zmianach



- b) Dodaj i pobierz z bazy kilka firm obu rodzajów stosując strategie mapowania dziedziczenia `TablePerType`,

Niestety strategia mapowania dziedziczenia `TablePerType` nie jest dostępna w wersji ponad 3.0 *Entity Framework*, więc nie da się wykonać tego podpunktu.

- c) Dodaj i pobierz z bazy kilka firm obu rodzajów strategie mapowania dziedziczenia `TablePerClass`

Strategia `TablePerClass` również nie jest możliwa do wykonania, ponieważ metoda `ToTable()`, która jest niezbędna do wykonania tego zadania rzuca błędem od wersji 3.0 *Entity Framework*.