

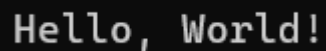
# Entity Framework

Uwagi do całego sprawozdania:

- zrzuty ekranu powstałych tabel pochodzić będą z DataGrip,
- kod C#, wyniki na standardowym wyjściu oraz inne zrzuty ekranu np. powstałych katalogów w drzewie plików pochodzić będą z Visual Studio.
- od zadania drugiego pomijam zamieszczanie w sprawozdaniu zrzutów ekranu ewentualnych migracji

## 1. Konfiguracja projektu

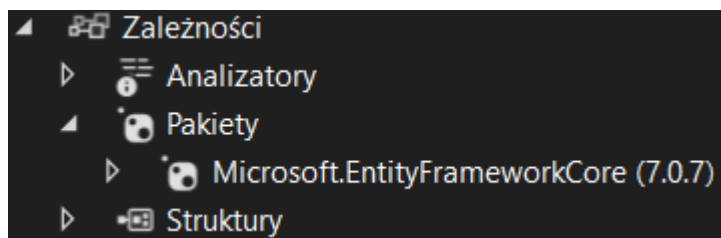
- a. Stworzyłem nowy projekt w Visual Studio
- b. Uruchomiłem projekt testowo i otrzymałem rezultat:



- c. Dodano do projektu klasę Product:

```
namespace ŁukaszDydekEFProducts
{
    internal class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
    }
}
```

- d. Dodałem do projektu Entity Framework:



- e. Po dociągnięciu przestrzeni nazw "Microsoft.EntityFrameworkCore" do kontekstu aktualnego pliku kod klasy ProductContext wygląda następująco:

```
using Microsoft.EntityFrameworkCore;
```

```
namespace ŁukaszDydekEFProducts
{
    internal class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
    }
}
```

f. Przygotowałem kod odpowiedzialny za migrację wykonując po kolei polecenia:

1. dotnet add package Microsoft.EntityFrameworkCore.Design
2. dotnet ef migrations add InitProductDatabase

g. Skonfigurowałem kontekst, aby wiedział, że chcę skorzystać z bazy SQLite:

```
using Microsoft.EntityFrameworkCore;

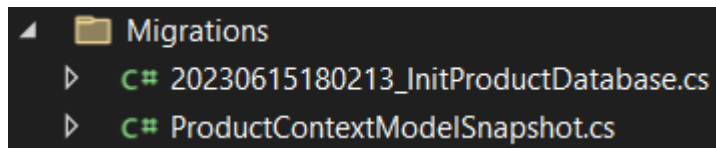
namespace ŁukaszDydekEFProducts
{
    internal class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }

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

h. Dodałem do projektu pakiet "Microsoft.EntityFrameworkCore.Sqlite" za pomocą polecenia:

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
```

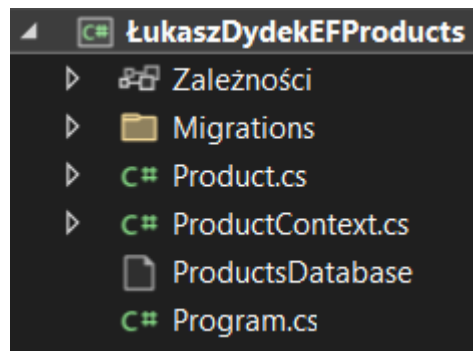
- i. Metoda UseSqlite zaczęła być widoczna.
- j. Po wykonaniu operacji dodania migracji w projekcie pojawił się katalog Migrations z definicjami migracji:



- k. Wykonałem update bazy danych za pomocą polecenia:

dotnet ef database update

- l. W strukturze plików pojawił się plik bazy danych:



- m. Dodałem produkt do bazy a następnie pobrałem wszystkie dane o produktach i wyświetliłem je w konsoli. Oto kod funkcji main odpowiedzialny za te operacje:

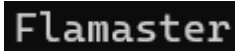
```
namespace ŁukaszDydekEFProducts
{
    class Program
    {
        static void Main(string[] args)
        {
            ProductContext productContext = new ProductContext();
            Product product = new Product { ProductName = "Flamaster" };
            productContext.Products.Add(product);
            productContext.SaveChanges();
            var query = from prod in productContext.Products
                        select prod.ProductName;
            foreach (var pName in query)
            {
                Console.WriteLine(pName);
            }
        }
    }
}
```

```

    }
}

```

n. oraz oczekiwany wynik na standardowym wyjściu:

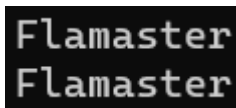


```

Flamaster

```

o. Po zmianie mechanizmu kopiowania z “Zawsze kopiuj” na “Kopiuj jeśli nowszy” na ekranie można zobaczyć następujący wynik:



```

Flamaster
Flamaster

```

p. Zmieniłem sposób dodawania nowych produktów do bazy. Teraz przed tą operacją użytkownik zostanie poproszony o podanie nazwy produktu, który chce dodać. Poniżej zamieszczam implementację opisanej idei:

```

namespace ŁukaszDydekEFProducts
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Podaj nazwę produktu");
            string prodName = Console.ReadLine();

            Console.WriteLine("Poniżej lista produktów zarejestrowanych w naszej
bazie danych");
            ProductContext productContext = new ProductContext();
            Product product = new Product { ProductName = prodName };
            productContext.Products.Add(product);
            productContext.SaveChanges();

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

```

```

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

```

q. oraz wynik operacji z podpunktu “p”:

```

Podaj nazwę produktu
ołówek
Poniżej lista produktów zarejestrowanych w naszej bazie danych
Flamaster
Flamaster
Flamaster
Flamaster
ołówek

```

r. Powstała tabela:

	ProductID	ProductName	UnitsOnStock
1	1	Flamaster	0
2	2	Flamaster	0
3	3	Flamaster	0
4	4	Flamaster	0
5	5	ołówek	0

## 2. Wprowadzenie pojęcia Dostawcy

a. Kody:

- Klasa Supplier:

```

namespace ŁukaszDydekEFProducts
{
    internal class Supplier
    {
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
    }
}

```

```

        public string Street { get; set; }
        public string City { get; set; }

        public Supplier(string CompanyName, string Street, string City)
        {
            this.CompanyName = CompanyName;
            this.Street = Street;
            this.City = City;
        }
    }
}

```

- Klasa Product:

```

namespace ŁukaszDydekEFProducts
{
    internal class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
        public Supplier? Supplier { get; set; } = null;
    }
}

```

- Klasa Program:

```

namespace ŁukaszDydekEFProducts
{
    class Program
    {
        static void Main()
        {
            ProductContext productContext = new();
            int productID = 4;
            Supplier supplier = new("DPD", "Lawendowa", "Kraków");
            productContext.Suppliers.Add(supplier);
        }
    }
}

```

```

        var query = from prod in productContext.Products
                     where prod.ProductID == productID
                     select prod;
        var product = query.FirstOrDefault();

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

```

b. Tabele w bazie przed wykonaniem operacji:

- Tabela Products:

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	Flamaster	<null>	0
2	2	Ołówek	<null>	0
3	3	Krzesło	<null>	0
4	4	Długopis	<null>	0

- Tabela Suppliers:

SupplierID	CompanyName	Street	City
------------	-------------	--------	------

c. Tabele w bazie po wykonaniu operacji:

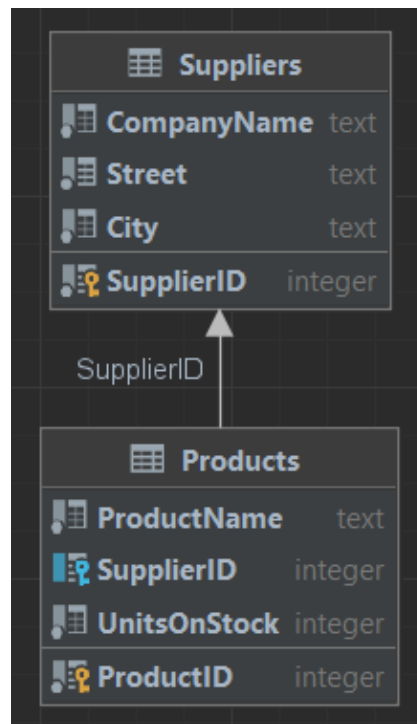
- Tabela Products:

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	Flamaster	<null>	0
2	2	Ołówek	<null>	0
3	3	Krzesło	<null>	0
4	4	Długopis	1	0

- Tabela Suppliers:

SupplierID	CompanyName	Street	City
1	DPD	Lawendowa	Kraków

d. Schemat bazy danych:



### 3. Odwrócenie relacji

a. Kody:

- Klasa Supplier:

```
namespace ŁukaszDydekEFProducts
{
    internal class Supplier
    {
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public List<Product> Products { get; set; } = new List<Product>();

        public Supplier(string CompanyName, string Street, string City)
        {
            this.CompanyName = CompanyName;
            this.Street = Street;
            this.City = City;
        }
    }
}
```



```

        public void AddProducts(List<Product> products)
        {
            Products.AddRange(products);
        }
    }
}

```

- Klasa Product:

```

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

        public Product(string productName, int unitsOnStock)
        {
            this.ProductName = productName;
            this.UnitsOnStock = unitsOnStock;
        }
    }
}

```

- Klasa Program:

```

namespace ŁukaszDydekEFProducts
{
    class Program
    {
        static void Main()
        {
            int supplierID = 1;
            ProductContext productContext = new();
            Product product1 = new("Komoda", 10);
            Product product2 = new("Woda", 5);

```

```

        Product product3 = new("Elektrolity", 3);
        Product product4 = new("Proszek Fiuu", 100);

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

        List<Product> products = new()
        {
            product1,
            product2,
            product3,
            product4
        };

        var query = from sup in productContext.Suppliers
                    where sup.SupplierID == supplierID
                    select sup;
        var supplier = query.FirstOrDefault();
        supplier.AddProducts(products);
        productContext.SaveChanges();
    }
}
}

```

b. Tabele w bazie przed wykonaniem operacji:

- Tabela Products:

ProductID	ProductName	SupplierID	UnitsOnStock
-----------	-------------	------------	--------------

- Tabela Suppliers:

SupplierID	CompanyName	Street	City
1	1 DPD	Lawendowa	Krakow

c. Tabele w bazie po wykonaniu operacji:

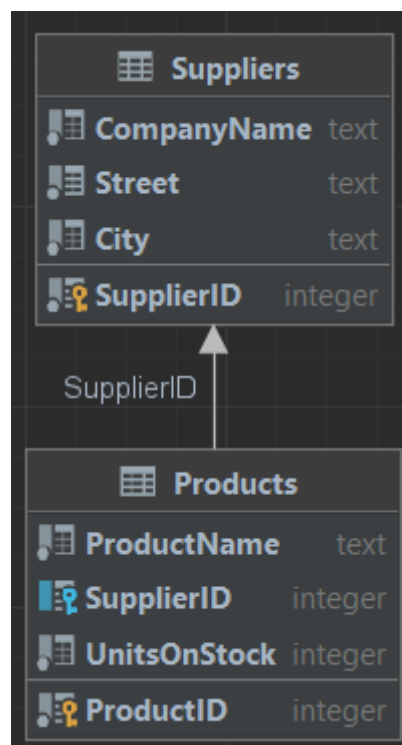
- Tabela Products:

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	Komoda	1	10
2	2	Woda	1	5
3	3	Elektrolity	1	3
4	4	Proszek Fiuu	1	100

- Tabela Suppliers:

	SupplierID	CompanyName	Street	City
1	1	DPD	Lawendowa	Krakow

- d. Schemat bazy danych:



#### 4. Modelowanie relacji dwustronnej

- a. Kody:

- Klasa Supplier:

Implementacja klasy Supplier nie uległa zmianie względem poprzedniego punktu.

- Klasa Product:

Do klasy Product dodano jedynie jedną liniijkę:

```
public Supplier? Supplier { get; set; } = null;
```

Reszta implementacji pozostała bez zmian względem poprzedniego punktu.

- Klasa Program:

```
namespace ŁukaszDydekEFProducts
{
    class Program
    {
        static void Main()
        {
            int supplierID = 1;
            ProductContext productContext = new();
            Product product1 = new("Doniczka", 14);
            Product product2 = new("Wino czerwone", 5);
            Product product3 = new("Wino białe", 3);
            Product product4 = new("Miotła", 100);

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

            List<Product> products = new()
            {
                product1,
                product2,
                product3,
                product4
            };

            var query = from sup in productContext.Suppliers
                        where sup.SupplierID == supplierID
                        select sup;
            var supplier = query.FirstOrDefault();
            supplier.AddProducts(products);
            productContext.SaveChanges();
        }
    }
}
```

```
}  
}  
}
```

b. Tabele w bazie przed wykonaniem operacji:

- Tabela Products:

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	Komoda	1	10
2	2	Woda	1	5
3	3	Elektrolity	1	3
4	4	Proszek Fiuu	1	100

- Tabela Suppliers:

	SupplierID	CompanyName	Street	City
1	1	DPD	Lawendowa	Krakow

c. Tabele w bazie po wykonaniu operacji:

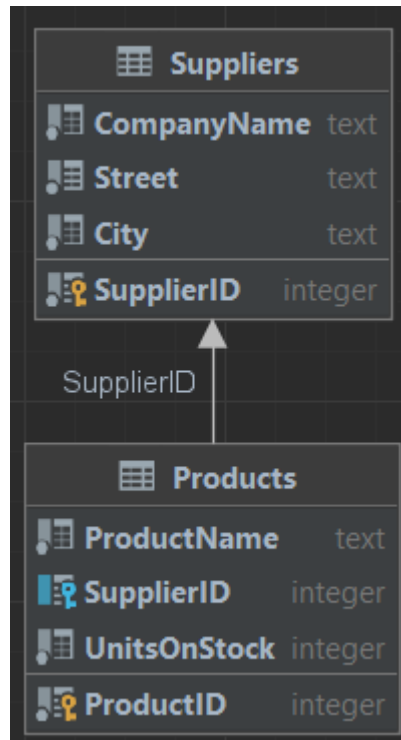
- Tabela Products:

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	Komoda	1	10
2	2	Woda	1	5
3	3	Elektrolity	1	3
4	4	Proszek Fiuu	1	100
5	5	Doniczka	1	14
6	6	Wino czerwone	1	5
7	7	Wino białe	1	3
8	8	Miotła	1	100

- Tabela Suppliers:

	SupplierID	CompanyName	Street	City
1	1	DPD	Lawendowa	Krakow

d. Schemat bazy danych:



## 5. Modelowanie relacji wiele-do-wielu

- a. Na samym początku stworzono odpowiedni model danych i dzięki migracji zaktualizowano schemat bazy danych. Kody interesujących nas klas zamieszczono poniżej:
  - Kod klasy Product:

```
namespace ŁukaszDydekEFProducts
{
    internal class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
        public Supplier? Supplier { get; set; } = null;
        public List<ProductInvoice> ProductInvoices { get; } = new();

        public Product(string productName, int unitsOnStock)
        {
            this.ProductName = productName;
            this.UnitsOnStock = unitsOnStock;
        }
    }
}
```

```
}  
}
```

- Kod klasy Invoice:

```
using System.ComponentModel.DataAnnotations;  
  
namespace ŁukaszDydekEFProducts  
{  
    internal class Invoice  
    {  
        [Key]  
        public int InvoiceNumber { get; set; }  
        public int Quantity { get; set; } = 0;  
        public List<ProductInvoice> ProductInvoices { get; } = new();  
    }  
}
```

- Kod klasy ProductInvoice:

```
using System.ComponentModel.DataAnnotations.Schema;  
using System.ComponentModel.DataAnnotations;  
  
namespace ŁukaszDydekEFProducts  
{  
    internal class ProductInvoice  
    {  
        [Key]  
        [Column(Order = 1)]  
        [ForeignKey("Invoice")]  
        public int InvoiceNumber { get; set; }  
  
        [Key]  
        [Column(Order = 2)]  
        [ForeignKey("Product")]  
        public int ProductID { get; set; }  
        public int Quantity { get; set; }  
    }  
}
```

```

        public virtual Invoice Invoice { get; set; }
        public virtual Product Product { get; set; }
    }
}

```

- Kod klasy ProductContext:

```

using Microsoft.EntityFrameworkCore;

namespace ŁukaszDydekEFProducts
{
    internal class ProductContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<ProductInvoice> ProductInvoice { get; set; }

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

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<ProductInvoice>()
                .HasKey(x => new { x.InvoiceNumber, x.ProductID });
        }
    }
}

```

Warto w szczególności, analizując powyższy kod, zwrócić uwagę na metodę OnModelCreating, ponieważ jest ona niezbędna podczas tworzenia relacji wiele-do-wielu.



- b. Teraz stworzono kilka produktów i faktur oraz rozpoczęto sprzedaż. Dodatkowo dodano do klas Product oraz Invoice po jednej metodzie odpowiedzialnej za sprzedaż produktów. Ich implementacje zamieszczono poniżej:

- fragment klasy Product:

```
public void Sell(Invoice invoice, int quantity)
{
    if (this.UnitsOnStock < quantity)
    {
        throw new ArgumentException("Brak wystarczającej ilości sztuk na magazynie.");
    }
    this.UnitsOnStock -= quantity;
    var productInvoice = new ProductInvoice { Product = this, Invoice = invoice, Quantity = quantity };
    this.ProductInvoices.Add(productInvoice);
    invoice.AddProduct(productInvoice, quantity);
}
```

- fragment klasy Invoice:

```
public void AddProduct(ProductInvoice productInvoice, int quantity)
{
    ProductInvoices.Add(productInvoice);
    this.Quantity += quantity;
}
```

- klasa Program:

```
using ŁukaszDydekEFProducts;

class Program
{
    static void Main()
    {
        ProductContext productContext = new();
        Product product1 = new("Golarka", 13);
        Product product2 = new("Szampon", 15);
    }
}
```

```

Product product3 = new("Mydło", 34);
Product product4 = new("Szczoteczka", 18);

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

Invoice invoice1 = new();
Invoice invoice2 = new();

productContext.Invoices.Add(invoice1);
productContext.Invoices.Add(invoice2);

product1.Sell(invoice1, 10);
product2.Sell(invoice2, 4);
product3.Sell(invoice2, 5);
product4.Sell(invoice1, 6);

productContext.SaveChanges();
}
}

```

c. Interesujące nas tabele w bazie po wykonaniu operacji:

- tabela Products:

	ProductID	ProductName	UnitsOnStock	SupplierID
1	1	Golarka	3	<null>
2	2	Szampon	11	<null>
3	3	Mydło	29	<null>
4	4	Szczoteczka	12	<null>

- tabela Invoices:

	InvoiceNumber	Quantity
1	1	16
2	2	9

- tabela ProductInvoice:

	InvoiceNumber	ProductID	Quantity
1	1	1	10
2	1	4	6
3	2	2	4
4	2	3	5

Analizując początkowe ilości poszczególnych produktów, te po wykonaniu operacji oraz ich ilości na odpowiednich fakturach, możemy dojść do wniosku, że implementacja funkcji sprzedającej produkty działa poprawnie.

d. Produkty sprzedane w ramach wybranej faktury/transakcji:

- kod klasy Program:

```
namespace ŁukaszDydekEFProducts
{
    class Program
    {
        static void Main()
        {
            int invoiceID = 1;
            ProductContext productContext = new();

            var productsNames = (
                from pi in productContext.ProductInvoice
                join p in productContext.Products on pi.ProductID equals
p.ProductID
                where pi.InvoiceNumber == invoiceID
                select p.ProductName
            ).ToList();

            foreach (var productName in productsNames)
            {

```

```

        Console.WriteLine(productName);
    }

    productContext.SaveChanges();
}
}
}

```

- Wynik na standardowym wyjściu:

**Gołarka  
Szczoteczka**

e. Faktury, w ramach których został sprzedany wybrany produkt:

- Kod klasy Program:

```

using ŁukaszDydekEFProducts;

class Program
{
    static void Main()
    {
        int productID = 3;
        ProductContext productContext = new();

        var invoices = (
            from pi in productContext.ProductInvoice
            where pi.ProductID == productID
            select pi
        ).ToList();

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

```

```

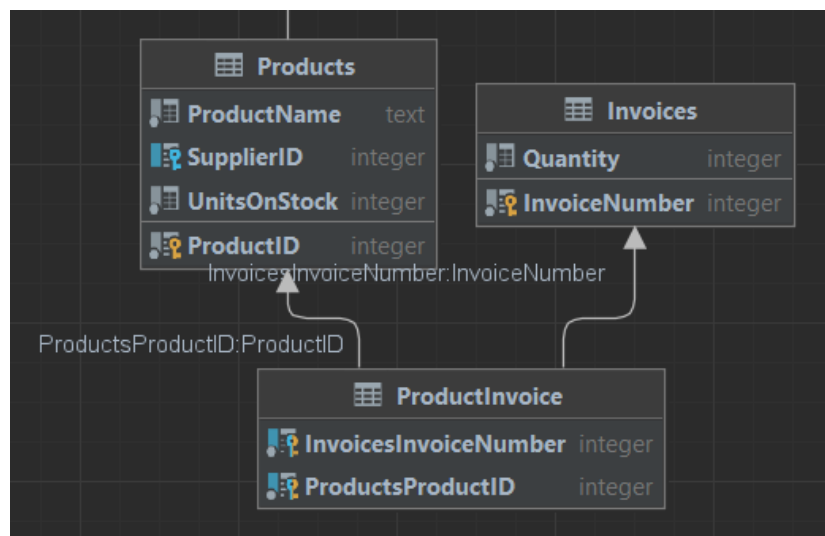
        productContext.SaveChanges();
    }
}

```

- Wynik na standardowym wyjściu:

2

- f. Schemat bazy danych:



## 6. Dziedziczenie Table-Per-Hierarchy

- a. Kod:

- klasa Company:

```

namespace ŁukaszDydekEFProducts
{
    internal abstract class Company
    {
        public int CompanyID { get; set; }
        public string CompanyName { get; set; }
        public string Street { get; set; }
        public string City { get; set; }
        public string ZipCode { get; set; }
    }
}

```

```
}
```

- klasa Supplier:

```
namespace ŁukaszDydekEFProducts
{
    internal class Supplier : Company
    {
        public string BankAccountNumber { get; set; }
    }
}
```

- klasa Customer:

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

- klasa Program:

```
using ŁukaszDydekEFProducts;

class Program
{
    static void Main()
    {
        ProductContext productContext = new();

        Supplier supplier = new Supplier
        {
            CompanyName = "ABC Suppliers",
            Street = "123 Supplier St",
            City = "Supplier City",
            ZipCode = "12345",
            BankAccountNumber = "1234567890"
        }
    }
}
```

```

    };

    productContext.Companies.Add(supplier);

    Customer customer = new Customer
    {
        CompanyName = "XYZ Customers",
        Street = "456 Customer Ave",
        City = "Customer City",
        ZipCode = "54321",
        Discount = 15
    };

    productContext.Companies.Add(customer);

    productContext.SaveChanges();
}
}

```

- klasa ProductContext:

```

using Microsoft.EntityFrameworkCore;

namespace ŁukaszDydekEFProducts
{
    internal class ProductContext : DbContext
    {
        public DbSet<Company> Companies { get; set; }
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

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

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

```

```

    }
}
}

```

b. Tabela w bazie danych przed wykonaniem operacji:

	CompanyID	BankAccountNumber	City	CompanyName	Discount	Discriminator	Street	ZipCode
1	1	1234567890	Supplier City	ABC Suppliers	<null>	Supplier	123 Supplier St	12345
2	2	9876543210	Distributor C...	XYZ Distributors	<null>	Supplier	456 Distributor...	54321
3	3	<null>	Customer City...	Customer 1 Inc.	10	Customer	123 Customer St	11111
4	4	<null>	Customer City...	Customer 2 LLC	15	Customer	456 Customer Ave	22222

Można zauważyć powstałą dodatkowo kolumnę "Discriminator", która informuje nas o dodanym typie obiektu do tabeli.

c. Tabela w bazie danych po wykonaniu operacji:

	CompanyID	BankAccountNumber	City	CompanyName	Discount	Discriminator	Street	ZipCode
1	1	1234567890	Supplier City	ABC Suppliers	<null>	Supplier	123 Supplier St	12345
2	2	9876543210	Distributor C...	XYZ Distributors	<null>	Supplier	456 Distributor...	54321
3	3	<null>	Customer City...	Customer 1 Inc.	10	Customer	123 Customer St	11111
4	4	<null>	Customer City...	Customer 2 LLC	15	Customer	456 Customer Ave	22222
5	5	<null>	Customer City	XYZ Customers	15	Customer	456 Customer Ave	54321
6	6	1234567890	Supplier City	ABC Suppliers	<null>	Supplier	123 Supplier St	12345

d. Odczytanie firm obu rodzajów z bazy:

```

using ŁukaszDydekEFProducts;

class Program
{
    static void Main()
    {
        ProductContext productContext = new();

        var result1 = productContext.Suppliers.FirstOrDefault(s => s.CompanyID
== 1);
        var result2 = productContext.Customers.FirstOrDefault(c => c.CompanyID
== 3);

        Console.WriteLine(result1.CompanyID);
        Console.WriteLine(result1.BankAccountNumber);
        Console.WriteLine(result1.City);
        Console.WriteLine(result1.CompanyName);
    }
}

```



```

        Console.WriteLine();
        Console.WriteLine(result2.CompanyID);
        Console.WriteLine(result2.Discount);
        Console.WriteLine(result2.City);
        Console.WriteLine(result2.CompanyName);
        Console.ReadKey();

        productContext.SaveChanges();
    }
}

```

e. Wynik na standardowym wyjściu:

```

1
1234567890
Supplier City
ABC Suppliers

3
10
Customer City 1
Customer 1 Inc.

```

f. Schemat bazy danych:

Companies	
BankAccountNumber	text
City	text
CompanyName	text
Discount	integer
Discriminator	text
Street	text
ZipCode	text
CompanyID	integer

## 7. Dziedziczenie Table-Per-Type

- a. W stosunku do poprzedniego przykładu wystarczyło dodać `[Table("Customers")]` oraz `[Table("Suppliers")]` przed nazwami klas, aby EF zmienił domyślny sposób dziedziczenia (czyli TPH) na TPT.
- b. Dodawanie do bazy kilku firm:

```
using ŁukaszDydekEFProducts;

class Program
{
    static void Main()
    {
        ProductContext productContext = new();

        Supplier supplier = new Supplier
        {
            CompanyName = "ABC Suppliers",
            Street = "123 Supplier St",
            City = "Supplier City",
            ZipCode = "12345",
            BankAccountNumber = "1234567890"
        };

        productContext.Companies.Add(supplier);

        Customer customer = new Customer
        {
            CompanyName = "XYZ Customers",
            Street = "456 Customer Ave",
            City = "Customer City",
            ZipCode = "54321",
            Discount = 15
        };

        productContext.Companies.Add(customer);
    }
}
```

```

        productContext.SaveChanges();
    }
}

```

c. Tabele w bazie danych po wykonaniu operacji:

- tabela Companies:

	CompanyID	City	CompanyName	Street	ZipCode
1	1	Supplier City	ABC Suppliers	123 Supplier St	12345
2	2	Customer City	XYZ Customers	456 Customer Ave	54321

- tabela Customers:

	CompanyID	Discount
1	2	15

- tabela Suppliers:

	CompanyID	BankAccountNumber
1	1	1234567890

d. Pobranie dodanych firm z bazy:

```

using ŁukaszDydekEFProducts;

class Program
{
    static void Main()
    {
        ProductContext productContext = new();

        var result1 = productContext.Suppliers.FirstOrDefault(s => s.CompanyID
== 1);
        var result2 = productContext.Customers.FirstOrDefault(c => c.CompanyID
== 2);

        Console.WriteLine(result1.CompanyID);
    }
}

```

```

        Console.WriteLine(result1.BankAccountNumber);
        Console.WriteLine(result1.City);
        Console.WriteLine(result1.CompanyName);
        Console.WriteLine();
        Console.WriteLine(result2.CompanyID);
        Console.WriteLine(result2.Discount);
        Console.WriteLine(result2.City);
        Console.WriteLine(result2.CompanyName);
        Console.ReadKey();

        productContext.SaveChanges();
    }
}

```

e. Wynik na standardowym wyjściu:

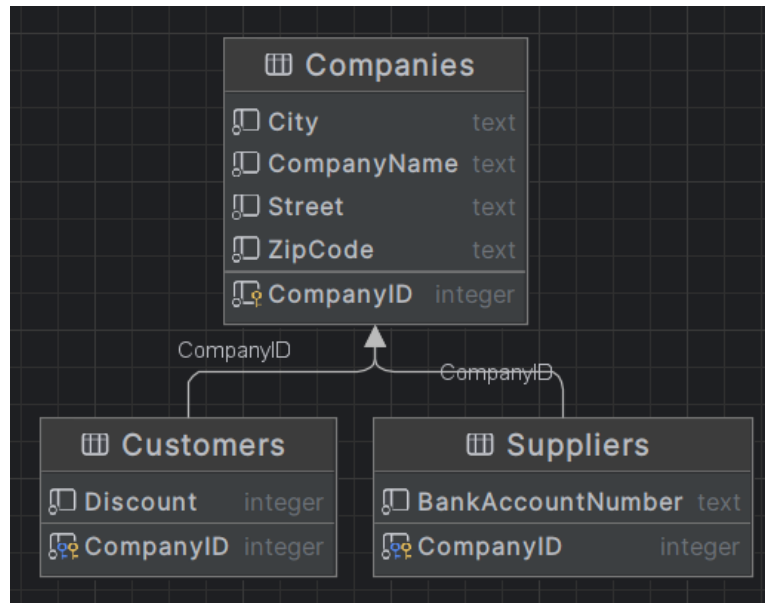
```

1
1234567890
Supplier City
ABC Suppliers

2
15
Customer City
XYZ Customers

```

f. Schemat bazy danych:



## 8. Porównanie dwóch strategii dziedziczenia w EF (TPH oraz TPT)

### a. TPH (Table-Per-Hierarchy):

Charakterystyka:

Tworzona jest jedna tabela, która zawiera kolumny zarówno z klasy nadrzędnej, jak i klas dziedziczących. Dodatkowo powstaje jeszcze jedna kolumna charakteryzująca każdą w klas dziedziczących (z którym konkretnie obiektem mamy do czynienia). Dla danego wiersza w tabeli, jeśli encja nie posiada konkretnego atrybutu, to zostaje po zmapowaniu w tabeli zapisana wartość null w kolumnie odpowiadającej temu brakującemu atrybutowi).

Zalety:

- prosta struktura - wszystkie typy dziedziczące są mapowane do jednej tabeli
- jedno zapytanie do odczytu danych - konsekwencja mapowania do jednej tabeli

Wady:

- nadmiarowa ilość wartości NULL - jest to marnotrawstwo miejsca i może wpływać na wydajność

### b. TPT (Table-Per-Type):

Charakterystyka:

Każda klasa zostaje zmapowana na osobną tabelę w bazie danych. Dodatkowo wszystkie tabele w hierarchii dziedziczenia współdzielą klucz główny (jest on dziedziczony przez wszystkie typy w hierarchii).

Zalety:

- czytelność - każdy typ danych ma swoją tabelę

- unikanie nadmiarowych wartości NULL

Wady:

- bardziej złożona struktura
- więcej zapytań w celu odczytu danych