Bazy Danych 2

Laboratorium 4

Entity Framework

Mateusz Łopaciński

1. Kod po wprowadzeniu

1.1. Zaimplementowane klasy

1.1.1. Klasa Product

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

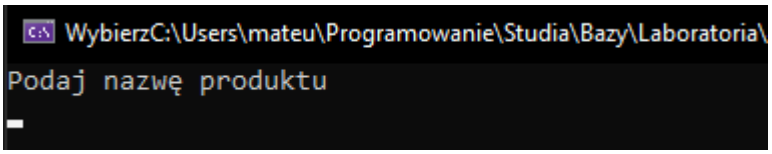
1.1.2. Klasa ShopContext

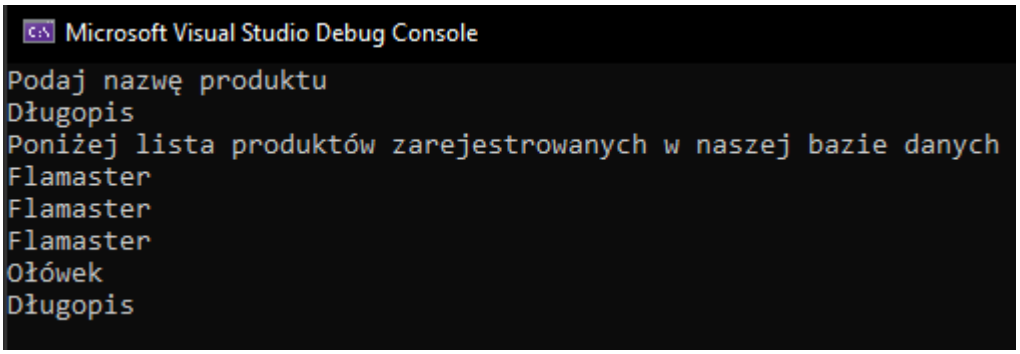
```
using Microsoft.EntityFrameworkCore;

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

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

1.1.3. Klasa Program

```
namespace MateuszLopacinskiEFProducts
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Podaj nazwę produktu");
            String prodName = Console.ReadLine();

            Console.WriteLine("Poniżej lista produktów zarejestrowanych w naszej bazie
danych");

            ShopContext shopContext = new ShopContext();
            Product product = new Product { ProductName = prodName };
            shopContext.Products.Add(product);
            shopContext.SaveChanges();

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

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

1.1. Przykład działania





2. Wprowadzenie pojęcia Dostawcy

2.1. Zaimplementowane klasy

2.1.1. Klasa Product

```
namespace MateuszLopacinskiEFProducts
{
    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 override string ToString()
        {
            return $"{ProductName} ({UnitsOnStock} szt.)";
        }
    }
}
```

2.1.2. Klasa Supplier

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

        public override string ToString()
        {
            return CompanyName;
        }
    }
}
```

2.1.3. Klasa ShopContext

```
using Microsoft.EntityFrameworkCore;

namespace MateuszLopacinskiEFProducts
{
    internal class ShopContext : DbContext
```

```
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Supplier> Suppliers { get; set; }

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

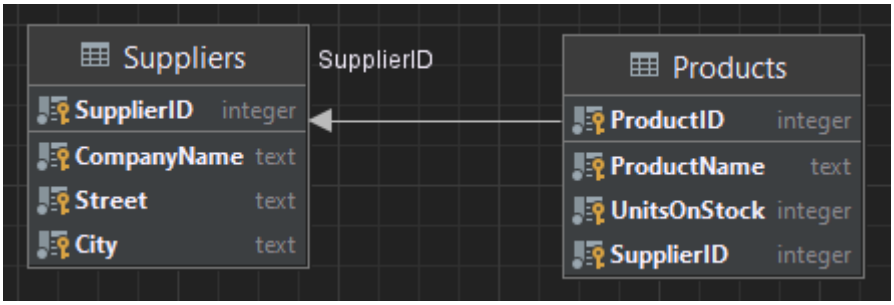
2.1.4. Klasa Program

```
namespace MateuszLopacinskiEFProducts
{
    class Program
    {
        static void Main()
        {
            ShopContext shopContext = new ShopContext();
            Product product = createNewProduct();
            Supplier? supplier = null;

            bool isCorrectChoice = false;
            bool createdNewSupplier = false;
            do {
                Console.WriteLine("Dodać nowego dostawcę? (tak/nie)");
                string choice = Console.ReadLine();
                switch (choice)
                {
                    case "tak":
                        isCorrectChoice = true;
                        supplier = createNewSupplier();
                        createdNewSupplier = true;
                        break;
                    case "nie":
                        isCorrectChoice = true;
                        displayAllSuppliers(shopContext);
                        supplier = findSupplier(shopContext);
                        break;
                }
            } while (!isCorrectChoice);


            Console.WriteLine("Dodaję dostawcę do produktu...");
            product.Supplier = supplier;

            Console.WriteLine("Zapisuję dane do bazy...");
            if (createdNewSupplier) shopContext.Suppliers.Add(supplier);
            shopContext.Products.Add(product);
            shopContext.SaveChanges();
        }

        private static Product createNewProduct()
        {
            Console.Write("Podaj nazwę produktu\n>>> ");
            string prodName = Console.ReadLine();
            Console.Write("Podaj liczbę dostępnych sztuk produktu\n>>> ");
            int quantity = Int32.Parse(Console.ReadLine());

            Console.WriteLine("Tworzę nowy produkt...");
            Product product = new Product
            {
                ProductName = prodName,
                UnitsOnStock = quantity
            };
            Console.WriteLine($"Stworzono produkt: {product}");
            return product;
        }
```

```
        private static Supplier createNewSupplier()
        {
            Console.Write("\nPodaj nazwę dostawcy\n>>> ");
            string companyName = Console.ReadLine();
            Console.Write("Podaj miasto\n>>> ");
            string city = Console.ReadLine();
            Console.Write("Podaj ulicę\n>>> ");
            string street = Console.ReadLine();

            Console.WriteLine("Tworzę nowego dostawcę...");
            Supplier supplier = new Supplier
            {
                CompanyName = companyName,
                City = city,
                Street = street
            };
            Console.WriteLine($"Stworzono dostawcę: {supplier}");
            return supplier;
        }

        private static Supplier findSupplier(ShopContext shopContext)
        {
            Console.Write("Wprowadź id dostawcy, który ma zostać przypisany do nowego
produktu\n>>>");
            int choice = Int32.Parse(Console.ReadLine());

            var query = from sup in shopContext.Suppliers
                        where sup.SupplierID == choice
                        select sup;

            return query.FirstOrDefault();
        }

        private static void displayAllSuppliers(ShopContext shopContext)
        {
            Console.WriteLine("Lista wszystkich dostawców");
            foreach (Supplier supplier in shopContext.Suppliers)
            {
                Console.WriteLine($"[{supplier.SupplierID}] {supplier}");
            }
        }
    }
}
```

2.2. Diagram bazy danych

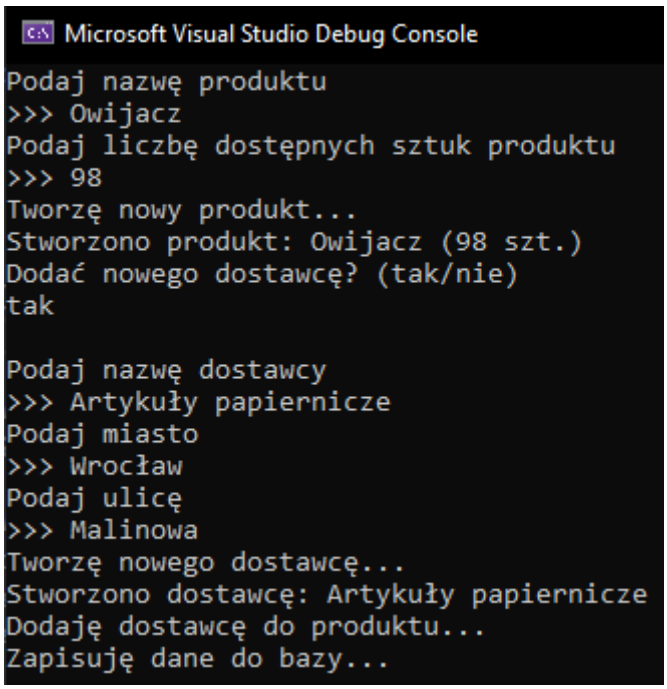
Poniżej zamieściłem diagram z programu DataGrip, przedstawiający modelowaną relację.



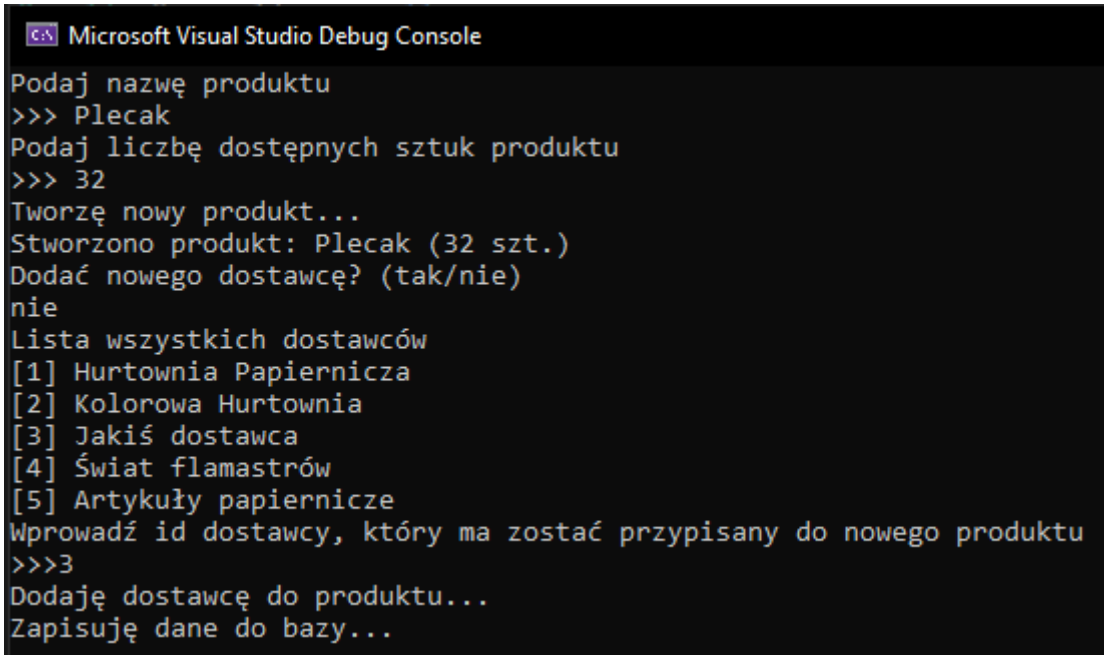
2.3. Przykład działania

2.3.1. Przykładowe wykonania programu

- **Z dodawaniem nowego dostawcy**



- **Z dodawaniem istniejącego dostawcy do nowego produktu**



2.3.2. Tabele po kilkukrotnym dodaniu produktów

- **Produkty**

	ProductID	ProductName	UnitsOnStock	SupplierID
1	1	Ołówek	12	1
2	2	Piórnik	11	1
3	3	Kredki	64	2
4	4	Nożyczki	12	2
5	5	Zeszyt	112	3
6	6	Flamaster	45	4
7	7	Owijacz	98	5
8	8	Plecak	32	3

- **Dostawcy**

	SupplierID	CompanyName	Street	City
1	1	Hurtownia Papiernicza	Nawojki	Kraków
2	2	Kolorowa Hurtownia	Kawiory	Kraków
3	3	Jakiś dostawca	Budryka	Kraków
4	4	Świat flamastrów	Królewska	Kraków
5	5	Artykuły papiernicze	Malinowa	Wrocław

3. Odwrócenie relacji

3.1. Zmienione klasy

3.1.1. Klasa Product

Usunięty został atrybut `public Supplier? Supplier`. Poza tym, klasa pozostaje bez zmian względem poprzedniej wersji.

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

        public override string ToString()
        {
            return $"{ProductName} ({UnitsOnStock} szt.)";
        }
    }
}
```

3.1.2. Klasa Supplier

W tej klasie dodana została kolekcja `public ICollection<Product> Products`. Poza tym, nie wprowadzałem innych zmian.

```
namespace MateuszLopacinskiEFProducts
{
    internal class Supplier
    {
        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; } = new List<Product>();

        public override string ToString()
        {
            return CompanyName;
        }
    }
}
```

3.1.3. Klasa Program

Zmienione zostały jednie 2 linijki w metodzie `static void Main()`, dlatego poniżej zamieszczam tylko fragment kodu, który został zamieniony oraz kod, którym go zastąpiłem.

- **Poprzedni kod (fragment)**

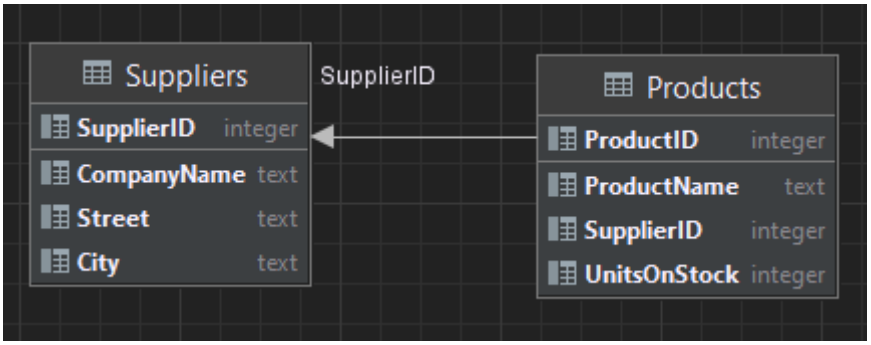
```
static void Main()
{
    …
    Console.WriteLine("Dodaję dostawcę do produktu...");
    product.Supplier = supplier;
    …
}
```

- **Nowy kod (fragment)**

```
static void Main()
{
    …
    Console.WriteLine("Dodaję produkt to dostawcy...");
    supplier.Products.Add(product);
    …
}
```

3.2. Diagram bazy danych

Jak możemy zauważyć, pomimo zapisania relacji w Entity Frameworku w odwrotny sposób, w bazie danych relacja wciąż wygląda tak samo. Widzimy więc, że Entity Framework „pod spodem” dokonał optymalizacji, dzięki czemu nie musimy trzymać w tabeli **Suppliers** powielonych danych dostawców, różniących się jedynie **SupplierID** oraz kluczem obcym, wskazującym na produkt z tabeli **Products**. Zauważmy, że w takiej sytuacji również mielibyśmy problem z rozróżnianiem dostawców, ponieważ jeden dostawca musał by sięznaleźć w tabeli wielokrotnie, mając za każdym razem przypisane inne id (traktowany byłby jako inny dostawca).



3.3. Przykład działania

3.3.1. Przykładowe wykonania programu

- **Z dodawaniem produktu do istniejącego dostawcy**

```
Microsoft Visual Studio Debug Console
Podaj nazwę produktu
>>> Długopis
Podaj liczbę dostępnych sztuk produktu
>>> 93
Tworzę nowy produkt...
Stworzono produkt: Długopis (93 szt.)
Dodać nowego dostawcę? (tak/nie)
nie
Lista wszystkich dostawców
[1] Warszawski Hurtownik
Wprowadź id dostawcy, który ma zostać przypisany do nowego produktu
>>>1
Dodaję produkt to dostawcy...
Zapisuję dane do bazy...
```

- **Z dodawaniem nowego dostawcy**

```
Microsoft Visual Studio Debug Console
Podaj nazwę produktu
>>> Owijacz
Podaj liczbę dostępnych sztuk produktu
>>> 91
Tworzę nowy produkt...
Stworzono produkt: Owijacz (91 szt.)
Dodać nowego dostawcę? (tak/nie)
tak

Podaj nazwę dostawcy
>>> Hurtownia Papiernicza
Podaj miasto
>>> Wrocław
Podaj ulicę
>>> Konwaliowa
Tworzę nowego dostawcę...
Stworzono dostawcę: Hurtownia Papiernicza
Dodaję produkt to dostawcy...
Zapisuję dane do bazy...
```

3.3.2. Tabele po dodaniu kilku dostawców i produktów

- **Produkty**

	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	Ołówek	1	231
2	2	Długopis	1	93
3	3	Owijacz	2	91

- **Dostawcy**

	SupplierID	CompanyName	Street	City
1	1	Warszawski Hurtownik	Malinowa	Warszawa
2	2	Hurtownia Papiernicza	Konwaliowa	Wrocław
3	3	Warszawski Hurtownik	Malinowa	Warszawa

4. Relacja dwustronna

4.1. Zmienione klasy

4.1.1. Klasa Product

Dodany został atrybut `public Supplier ?Supplier { get; set; } = null;` (taki jak w pierwszej implementacji klasy `Product`). Klasa wygląda więc identycznie jak ta z podpunktu **2.1.1.**, dlatego nie umieściłem poniżej jej kodu.

4.1.2. Klasa Supplier

Ta klasa nie została zmieniona względem implementacji z punktu **3.1.2.**, dlatego nie wkleiłem jej implementacji poniżej.

4.1.3. Klasa Program

Ponownie została dodana linijka , (taka jak w implementacji z punktu **2.1.4.**). Poniżej umieściłem jedynie fragment kodu, przedstawiający modyfikację względem implementacji z punktu **3.1.3.**).

- **Poprzedni kod (fragment)**

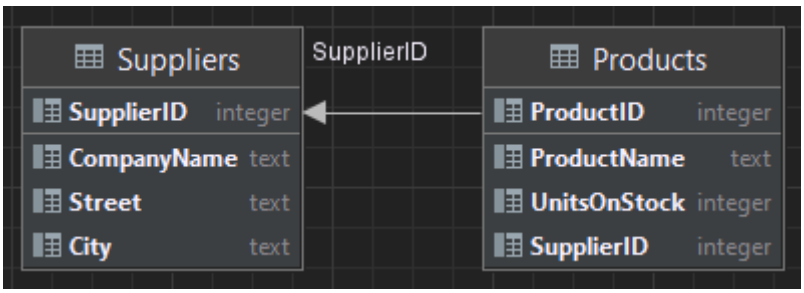
```
static void Main()
{
    …
    Console.WriteLine("Dodaję produkt to dostawcy...");
    supplier.Products.Add(product);
    …
}
```

- **Nowy kod (fragment)**

```
static void Main()
{
    …
    Console.WriteLine("Dodaję produkt to dostawcy...");
    product.Supplier = supplier;
    supplier.Products.Add(product);
    …
}
```

4.2. Diagram bazy danych

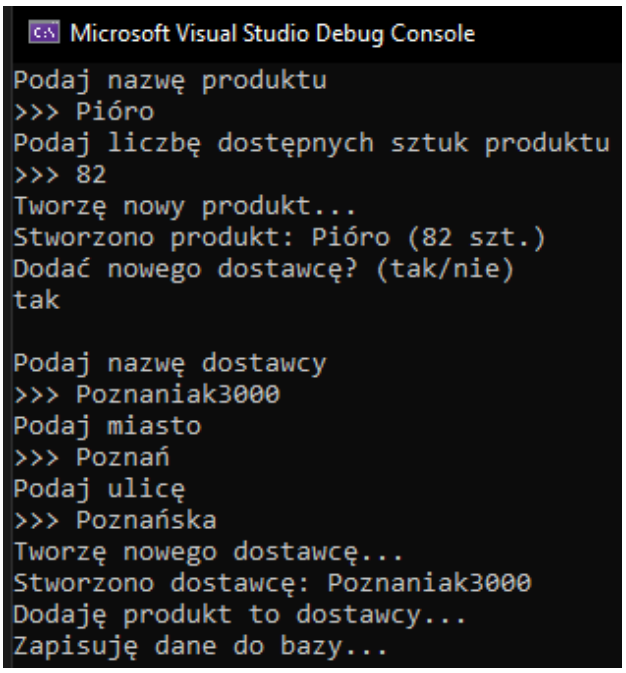
Ponownie obserwujemy taki sam diagram. Możemy więc dojść do wniosku, że Entity Framework pozwala nam na stworzenie relacji dwukierunkowej (lub w odwrotnym kierunku do tego, w którym relacja zostanie zapisana, jak widzieliśmy w poprzednim przykładzie), po to, aby łatwiej móc manipulować powiązanymi ze sobą obiektami. Mimo to, „pod spodem” zapisane przez nas relacje są przekształcane na relacje dające się zapisać w bazie danych.



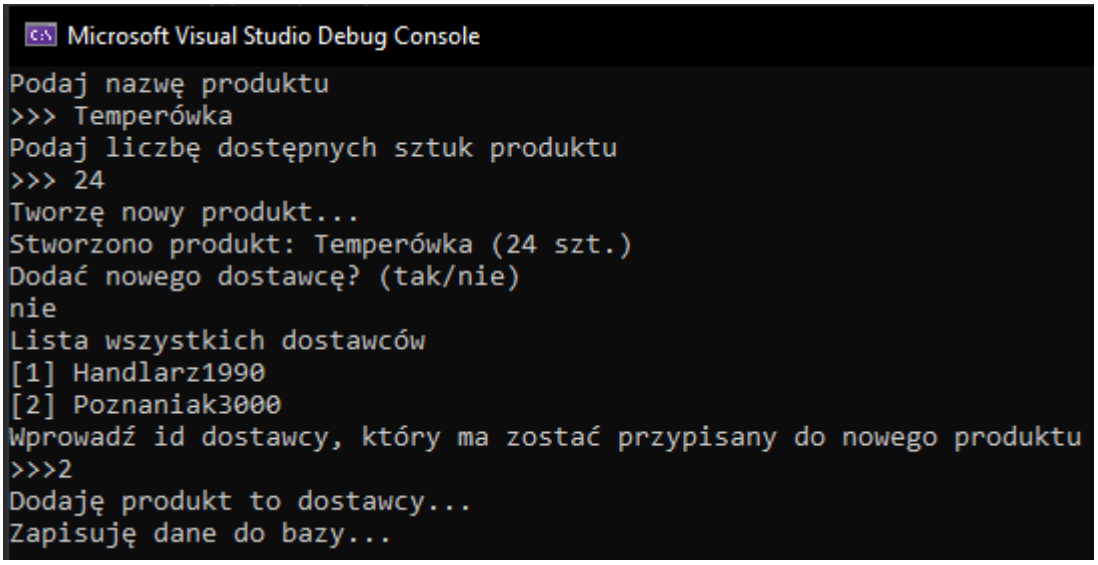
4.3. Przykład działania

4.3.1. Przykładowe wykonania programu

- **Z dodawaniem nowego dostawcy**



- **Z dodawaniem produktu do istniejącego dostawcy**



4.3.2. Tabele po dodaniu kilku dostawców i produktów

- **Produkty**

	ProductID	ProductName	UnitsOnStock	SupplierID
1	1	Ołówek	11	1
2	2	Pióro	82	2
3	3	Temperówka	24	2

- **Dostawcy**

	SupplierID	CompanyName	Street	City
1	1	Handlarz1990	Morska	Gdańsk
2	2	Poznaniak3000	Poznańska	Poznań

5. Relacja wiele do wielu

5.1. Zmienione klasy

5.1.1. Klasa Product

Dodana została kolekcja faktur, na których wystąpił produkt `public ICollection<Invoice> Invoices`.

```
namespace MateuszLopacinskiEFProducts
{
    internal class Product
    {
        public int ProductID { get; set; }

        public string ProductName { get; set; }
        public int UnitsInStock { get; set; }

        // Navigation properties
        public virtual ICollection<InvoiceItem> InvoiceItems { get; set; }

        public override string ToString()
        {
            return $"{ProductName} ({UnitsInStock} szt. dostępnych)";
        }
    }
}
```

5.1.2. Klasa Invoice

Stworzyłem również poniższą klasę.

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

namespace MateuszLopacinskiEFProducts
{
    internal class Invoice
    {
        [Key]
        public int InvoiceNumber { get; set; }

        // Navigation properties
        public virtual ICollection<InvoiceItem> InvoiceItems { get; set; }

        public override string ToString()
        {
            StringBuilder sn = new($"Invoice {InvoiceNumber}:");
            foreach (InvoiceItem item in InvoiceItems)
            {
                sn.Append($"\t- {item}");
            }
            return sn.ToString();
        }
    }
}
```

5.1.3. Klasa InvoiceItem

Klasa pomocnicza, reprezentująca pozycje faktury.

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

namespace MateuszLopacinskiEFProducts
{
    class InvoiceItem
```

```
    {
        [Key, Column(Order = 0)]
        public int InvoiceNumber { get; set; }

        [Key, Column(Order = 1)]
        public int ProductID { get; set; }

        // Navigation properties
        public virtual Invoice Invoice { get; set; }
        public virtual Product Product { get; set; }

        // Additional values
        public int Quantity { get; set; }

        public override string ToString()
        {
            return $"{Product} ({Quantity} szt.)";
        }
    }
}
```

5.1.4. Klasa ShopContext

W przypadku tej klasy, dodałem zbiór faktur `public DbSet<Invoice> Invoices { get; set; }`. Jednocześnie usunąłem zbiór sprzedawców (**Suppliers**), ponieważ nie jest on potrzebny w tym zadaniu.

```
using Microsoft.EntityFrameworkCore;

namespace MateuszLopacinskiEFProducts
{
    internal class ShopContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Invoice> Invoices { get; set; }
        public DbSet<InvoiceItem> InvoiceItems { get; set; }

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

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

5.1.5. Klasa Program

```
namespace MateuszLopacinskiEFProducts
{
    class Program
    {
        public const string ADD = "add";              // Add a new product
        public const string REMOVE = "remove";        // Remove a product
        public const string SELL = "sell";            // Sell products
        public const string ALL = "all";              // Display all products
        public const string AVAIALBLE = "available";  // Display available products
        public const string EXIT = "exit";            // Exit

        public static List<string> ALL_CMDS = new() {
            ADD, REMOVE, SELL, ALL, AVAIALBLE, EXIT
        };

        static void Main()
```

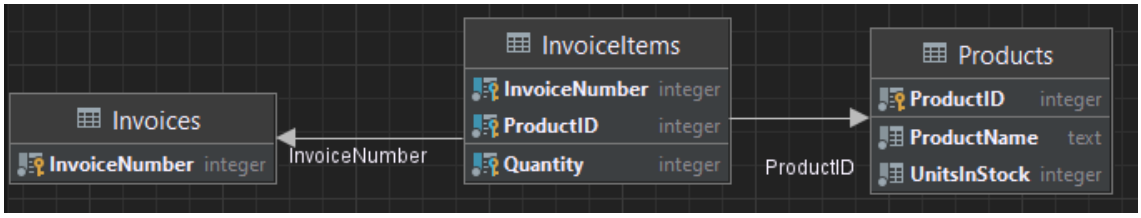
```
        {
            using ShopContext shopContext = new();

            bool exited = false;
            while (!exited)
            {
                switch (GetCommand())
                {
                    case ADD:
                        AddNewProduct(shopContext);
                        break;
                    case REMOVE:
                        RemoveProduct(shopContext);
                        break;
                    case SELL:
                        SellProduct(shopContext);
                        break;
                    case ALL:
                        DisplayAllProducts(shopContext);
                        break;
                    case AVAIALBLE:
                        DisplayAvailableProducts(shopContext);
                        break;
                    case EXIT:
                        exited = true;
                        Console.WriteLine("To dzisiaj na tyle, dzięki za współpracę");
                        break;
                    default:
                        Console.WriteLine("Polecenie nie zostało rozpoznane, spróbuj
jeszcze raz.");
                        break;
                }
                Console.WriteLine();
            }
        }

        private static string GetCommand()
        {
            Console.WriteLine("Napisz, co chcesz zrobić. Lista dostępnych komend:");
            DisplayAvailableCommands();
            Console.Write(">>> ");
            return Console.ReadLine();
        }

        private static void AddNewProduct(ShopContext shopContext)
        {
            Product product = CreateNewProduct();
            Console.WriteLine("Zapisuję produkt do bazy danych...");
            shopContext.Products.Add(product);
            shopContext.SaveChanges();
        }

        private static void SellProduct(ShopContext shopContext)
        {
            List<InvoiceItem> items = ChooseInvoiceItems(shopContext);
            Console.WriteLine("Aktualizuję liczbę dostępnych produktów...");
            foreach (InvoiceItem item in items)
            {
                item.Product.UnitsInStock -= item.Quantity;
                Console.WriteLine($"Pozostało {item.Product.UnitsInStock} szt.
{item.Product.ProductName}");
            }

            Invoice invoice = CreateInvoice(items);
            shopContext.Invoices.Add(invoice);
            shopContext.SaveChanges();
        }

        private static void RemoveProduct(ShopContext shopContext)
```

```
        {
            Console.WriteLine("Lista wszystkich produktów:");
            DisplayAllProducts(shopContext);
            Console.Write("Podaj id produktu do usunięcia\n>>> ");
            int id = Int32.Parse(Console.ReadLine());

            var query = from product in shopContext.Products
                        where product.ProductID == id
                        select product;

            if (query?.Count() > 0)
            {
                shopContext.Remove(query.First());
                shopContext.SaveChanges();
                Console.WriteLine("Produkt został pomyślnie usunięty");
            } else
            {
                Console.WriteLine($"Nie można usunąć produktu. Produkt o id równym {id}
nie istnieje");
            }
        }

        private static Invoice CreateInvoice(List<InvoiceItem> items)
        {
            return new Invoice
            {
                InvoiceItems = items
            };
        }

        private static Product CreateNewProduct()
        {
            Console.Write("Podaj nazwę produktu\n>>> ");
            string prodName = Console.ReadLine();
            Console.Write("Podaj liczbę dostępnych sztuk produktu\n>>> ");
            int quantity = Int32.Parse(Console.ReadLine());

            Console.WriteLine("Tworzę nowy produkt...");
            Product product = new()
            {
                ProductName = prodName,
                UnitsInStock = quantity
            };
            Console.WriteLine($"Stworzono produkt: {product}");
            return product;
        }

        private static List<InvoiceItem> ChooseInvoiceItems(ShopContext shopContext)
        {
            Console.WriteLine("Z poniższej listy wybierz produkty, które mają zostać
dodane do faktury");
            Console.WriteLine("- wprowadź id produktu, a po spacji liczbę sprzedawanych
sztuk");
            Console.WriteLine("- aby zakończyć wybieranie produktów, naciśniej
Enter\n");
            DisplayAvailableProducts(shopContext);

            Dictionary<Product, int> addedItems = new();

            while (true)
            {
                Console.Write("\n>>> ");
                string input = Console.ReadLine();

                if (input == String.Empty)
                {
                    Console.WriteLine("Zakończono wybieranie produktów");
                    break;
                } else
```

```
                {
                    string[] splitted = input.Split();
                    int id = Int32.Parse(splitted[0]);
                    int quantity = Int32.Parse(splitted[1]);

                    Product product = shopContext.Products.First(p => p.ProductID ==
id);
                    int newQuantity = quantity + (addedItems.ContainsKey(product) ?
addedItems[product] : 0);
                    if (newQuantity > product.UnitsInStock)
                    {
                        Console.WriteLine($"Nie można dodać {product.ProductName}.
Dostępnych jest tylko {product.UnitsInStock} szt.");
                    } else
                    {
                        Console.WriteLine($"Dodano {quantity} szt. {product.ProductName}
do faktury. Razem na fakturze jest {newQuantity} szt.");
                        if (addedItems.ContainsKey(product)) addedItems.Remove(product);
                        addedItems.Add(product, newQuantity);
                    }
                }
            }

            List<InvoiceItem> items = new();
            foreach (KeyValuePair<Product, int> item in addedItems)
            {
                items.Add(new InvoiceItem { Product = item.Key, Quantity = item.Value
});
            }

            return items;
        }

        private static void DisplayAvailableCommands()
        {
            foreach (var cmd in Program.ALL_CMDS)
            {
                Console.WriteLine($"\t- {cmd},");
            }
        }

        private static void DisplayAllProducts(ShopContext shopContext)
        {
            var query = from product in shopContext.Products
                        select product;

            foreach (var product in query)
            {
                Console.WriteLine($"[{product.ProductID}] {product.ProductName}
(dostępne: {product.UnitsInStock})");
            }
            if (query.Count() == 0)
            {
                Console.WriteLine("Brak produktów w bazie danych");
            }
        }

        private static void DisplayAvailableProducts(ShopContext shopContext)
        {
            var query = from product in shopContext.Products
                        where product.UnitsInStock > 0
                        select product;

            foreach (var product in query)
            {
                Console.WriteLine($"[{product.ProductID}] {product.ProductName}
(dostępne: {product.UnitsInStock})");
            }
            if (query?.Count() == 0)
```

```
            {
                Console.WriteLine("Brak produktów w bazie danych");
            }
        }
    }
}
```

5.2. Diagram bazy danych



5.3. Przykład działania

5.3.1. Przykładowe wykonania programu

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> add
Podaj nazwę produktu
>>> Zszywacz
Podaj liczbę dostępnych sztuk produktu
>>> 8
Tworzę nowy produkt...
Stworzono produkt: Zszywacz (8 szt.)
Zapisuję produkt do bazy danych...
```

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> all
[1] Ołówek (dostępne: 9)
[3] Temperówka (dostępne: 64)
[4] Kredki (dostępne: 18)
[5] Zeszyt (dostępne: 100)
[7] Piórnik (dostępne: 15)
[9] Pióro (dostępne: 16)
[10] Zszywacz (dostępne: 8)
```

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> remove
Lista wszystkich produktów:
[1] Ołówek (dostępne: 9)
[3] Temperówka (dostępne: 64)
[4] Kredki (dostępne: 18)
[5] Zeszyt (dostępne: 100)
[7] Piórnik (dostępne: 15)
[9] Pióro (dostępne: 16)
[10] Zszywacz (dostępne: 8)
Podaj id produktu do usunięcia
>>> 9
Produkt został pomyślnie usunięty
```

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> sell
Z poniższej listy wybierz produkty, które mają zostać dodane do faktury
- wprowadź id produktu, a po spacji liczbę sprzedawanych sztuk
- aby zakończyć wybieranie produktów, naciśniej Enter

[1] Ołówek (dostępne: 9)
[3] Temperówka (dostępne: 64)
[4] Kredki (dostępne: 18)
[5] Zeszyt (dostępne: 100)
[7] Piórnik (dostępne: 15)
[10] Zszywacz (dostępne: 8)

>>> 1 5
Dodano 5 szt. Ołówek do faktury. Razem na fakturze jest 5 szt.

>>> 4 11
Dodano 11 szt. Kredki do faktury. Razem na fakturze jest 11 szt.

>>> 5 9
Dodano 9 szt. Zeszyt do faktury. Razem na fakturze jest 9 szt.

>>> 5 12
Dodano 12 szt. Zeszyt do faktury. Razem na fakturze jest 21 szt.

>>>
Zakończono wybieranie produktów
Aktualizuję liczbę dostępnych produktów...
Pozostało 4 szt. Ołówek
Pozostało 7 szt. Kredki
Pozostało 79 szt. Zeszyt
```

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> available
[1] Ołówek (dostępne: 4)
[3] Temperówka (dostępne: 64)
[4] Kredki (dostępne: 7)
[5] Zeszyt (dostępne: 79)
[7] Piórnik (dostępne: 15)
[10] Zszywacz (dostępne: 8)

Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> sell
Z poniższej listy wybierz produkty, które mają zostać dodane do faktury
- wprowadź id produktu, a po spacji liczbę sprzedawanych sztuk
- aby zakończyć wybieranie produktów, naciśniej Enter

[1] Ołówek (dostępne: 4)
[3] Temperówka (dostępne: 64)
[4] Kredki (dostępne: 7)
[5] Zeszyt (dostępne: 79)
[7] Piórnik (dostępne: 15)
[10] Zszywacz (dostępne: 8)

>>> 1 4
Dodano 4 szt. Ołówek do faktury. Razem na fakturze jest 4 szt.

>>>
Zakończono wybieranie produktów
Aktualizuję liczbę dostępnych produktów...
Pozostało 0 szt. Ołówek
```

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> available
[3] Temperówka (dostępne: 64)
[4] Kredki (dostępne: 7)
[5] Zeszyt (dostępne: 79)
[7] Piórnik (dostępne: 15)
[10] Zszywacz (dostępne: 8)
```

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> all
[1] Ołówek (dostępne: 0)
[3] Temperówka (dostępne: 64)
[4] Kredki (dostępne: 7)
[5] Zeszyt (dostępne: 79)
[7] Piórnik (dostępne: 15)
[10] Zszywacz (dostępne: 8)
```

```
Napisz, co chcesz zrobić. Lista dostępnych komend:
        - add,
        - remove,
        - sell,
        - all,
        - available,
        - exit,
>>> exit
To dzisiaj na tyle, dzięki za współpracę

Zapisuję dane do bazy...
```

5.3.2. Tabele po dodaniu / usunięciu / sprzedaniu kilku produktów

- **Products**

	ProductID	ProductName	UnitsInStock
1	1	Ołówek	0
2	3	Temperówka	64
3	4	Kredki	7
4	5	Zeszyt	79
5	7	Piórnik	15
6	10	Zszywacz	8

- **Invoices**

	InvoiceNumber
1	1
2	2

- **InvoiceItems**

	InvoiceNumber	ProductID	Quantity
1	1	1	5
2	1	4	11
3	1	5	21
4	2	1	4

6. Dziedziczenie Table-Per-Hierarchy

6.1. Wykorzystane klasy

6.1.1. Klasa Company

```
namespace MateuszLopacinskiEFProducts
{
    internal abstract class Company
    {
        public int CompanyID { get; set; }
        public string CompanyName { get; set; } = String.Empty;
        public string Street { get; set; } = String.Empty;
        public string City { get; set; } = String.Empty;
        public string ZipCode { get; set; } = String.Empty;

        public override string ToString()
        {
            return $"[{CompanyID}] {CompanyName}";
        }
    }
}
```

6.1.2. Klasa CompanyType (jako enum na stringach)

```
namespace MateuszLopacinskiEFProducts
{
    internal static class CompanyType
    {
        public const string CUSTOMER = "customer";
        public const string SUPPLIER = "supplier";

        public static List<string> ALL_TYPES = new()
        {
            CUSTOMER,
            SUPPLIER
        };
    }
}
```

6.1.3. Klasa Supplier

```
namespace MateuszLopacinskiEFProducts
{
    internal class Supplier : Company
    {
        public int SupplierID { get; set; }
        public string BankAccountNumber { get; set; } = String.Empty;

        public override string ToString()
        {
            return $"{base.ToString()} (dostawca)";
        }
    }
}
```

6.1.4. Klasa Customer

```
namespace MateuszLopacinskiEFProducts
{
    internal class Customer : Company
    {
        public int CustomerID { get; set; }
        public int Discount { get; set; } // In %

        public override string ToString()
        {
            return $"{base.ToString()} (klient)";
        }
```

```
    }
}
```

6.1.5. CompanyContext

```
using Microsoft.EntityFrameworkCore;

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

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

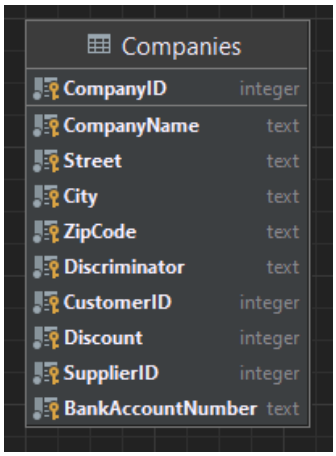
6.1.6. Klasa Program

```
namespace MateuszLopacinskiEFProducts
{
    static class ProgramAction
    {
        public const string ADD = "add";
        public const string DISPLAY = "display";

        public static List<string> ACTIONS = new()
        {
            ADD, DISPLAY
        };
    }
    class Program
    {
        static void Main()
        {
            using CompanyContext companyContext = new();

            string action = Choose("Wybierz, co chcesz zrobić", ProgramAction.ACTIONS);

            switch (action)
            {
                case ProgramAction.ADD:
                    AddCompany(companyContext);
                    break;
                case ProgramAction.DISPLAY:
                    DisplayCompanies(companyContext);
                    break;
            }
        }

        private static void AddCompany(CompanyContext companyContext)
        {
            while (true)
            {
                string type = Choose("Wprowadź typ firmy, którą chcesz dodać",
CompanyType.ALL_TYPES);

                string companyName = Input("Podaj nazwę firmy");
                string street = Input("Podaj ulicę");
                string city = Input("Podaj miasto");
                string postalCode = Input("Podaj kod pocztowy (zip)");

                switch (type)
                {
```

```
                    case CompanyType.CUSTOMER:
                        companyContext.Companies.Add(CreateCustomer(companyName, street,
city, postalCode));
                        companyContext.SaveChanges();
                        return;
                    case CompanyType.SUPPLIER:
                        companyContext.Companies.Add(CreateSupplier(companyName, street,
city, postalCode));
                        companyContext.SaveChanges();
                        return;
                }
            }
        }

        private static Supplier CreateSupplier(string companyName, string street, string
city, string postalCode)
        {
            string bankAccount = Input("Podaj numer konta bankowego");

            return new Supplier
            {
                CompanyName = companyName,
                Street = street,
                City = city,
                ZipCode = postalCode,
                BankAccountNumber = bankAccount
            };
        }

        private static Customer CreateCustomer(string companyName, string street, string
city, string postalCode)
        {
            int discount = int.Parse(Input("Podaj wartość zniżki (%)"));

            return new Customer
            {
                CompanyName = companyName,
                Street = street,
                City = city,
                ZipCode = postalCode,
                Discount = discount
            };
        }

        private static void DisplayCompanies(CompanyContext companyContext)
        {
            List<string> types = new();
            types.AddRange(CompanyType.ALL_TYPES);
            types.Add("all");

            string type = Choose("Wprowadź typ firm, które chcesz wypisać", types);

            switch (type)
            {
                case "all":
                    DisplayAllCompanies(companyContext);
                    break;
                case CompanyType.SUPPLIER:
                    DisplaySuppliers(companyContext);
                    break;
                case CompanyType.CUSTOMER:
                    DisplayCustomers(companyContext);
                    break;
            }
        }

        private static void DisplaySuppliers(CompanyContext companyContext)
        {
            Console.WriteLine("Lista wszystkich dostawców (firm):");
```

```
            foreach (Supplier supplier in companyContext.Suppliers)
            {
                Console.WriteLine(supplier);
            }
        }

        private static void DisplayCustomers(CompanyContext companyContext)
        {
            Console.WriteLine("Lista wszystkich klientów (firm):");
            foreach (Customer customer in companyContext.Customers)
            {
                Console.WriteLine(customer);
            }
        }

        private static void DisplayAllCompanies(CompanyContext companyContext)
        {
            Console.WriteLine("Lista wszystkich klientów (firm):");
            foreach (Company company in companyContext.Companies)
            {
                Console.WriteLine(company);
            }
        }

        private static string Input(string text)
        {
            Console.Write($"{text}\n>>> ");
            string? input = Console.ReadLine();
            if (input == null) return String.Empty;
            return input.Trim();
        }

        public static string Choose(string text, List<string> choices)
        {
            Console.WriteLine(text);
            Console.WriteLine("Możliwy wybór:");
            foreach (string choice in choices) Console.WriteLine($"\t- {choice}");
            Console.WriteLine("Aby wyjść, wpisz 'exit'");

            while (true)
            {
                string choice = Input("");

                foreach (string possibleChoice in choices)
                {
                    if (choice.Equals(possibleChoice)) return choice;
                }

                if (choice.Equals("exit"))
                {
                    Console.WriteLine("To dzisiaj na tyle, dzięki za współpracę");
                    return string.Empty;
                }

                Console.WriteLine("Polecenie nie zostało rozpoznane, spróbuj jeszcze
raz\n");
            }
        }
    }
}
```

6.2. Diagram bazy danych



6.3. Przykład działania

6.3.1. Dodawanie klienta

```
Wybierz, co chcesz zrobić
Możliwy wybór:
        - add
        - display
Aby wyjść, wpisz 'exit'

>>> add
Wprowadź typ firmy, którą chcesz dodać
Możliwy wybór:
        - customer
        - supplier
Aby wyjść, wpisz 'exit'

>>> customer
Podaj nazwę firmy
>>> Kliencik22
Podaj ulicę
>>> Jutlandzka
Podaj miasto
>>> Szczecin
Podaj kod pocztowy (zip)
>>> 70-001
Podaj wartość zniżki (%)
>>> 5
```

6.3.2. Dodawanie dostawcy

```
Wybierz, co chcesz zrobić
Możliwy wybór:
        - add
        - display
Aby wyjść, wpisz 'exit'

>>> add
Wprowadź typ firmy, którą chcesz dodać
Możliwy wybór:
        - customer
        - supplier
Aby wyjść, wpisz 'exit'

>>> supplier
Podaj nazwę firmy
>>> Dostawca Wszystkiego
Podaj ulicę
>>> 3 Maja
Podaj miasto
>>> Wrocław
Podaj kod pocztowy (zip)
>>> 55-010
Podaj numer konta bankowego
>>> PL61109010140000079999999999
```

6.3.3. Wypisywanie klientów

```
Wybierz, co chcesz zrobić
Możliwy wybór:
        - add
        - display
Aby wyjść, wpisz 'exit'

>>> display
Wprowadź typ firm, które chcesz wypisać
Możliwy wybór:
        - customer
        - supplier
        - all
Aby wyjść, wpisz 'exit'

>>> customer
Lista wszystkich klientów (firm):
[2] Super klient (klient)
[3] Kliencik22 (klient)
```

6.3.4. Wypisywanie dostawców

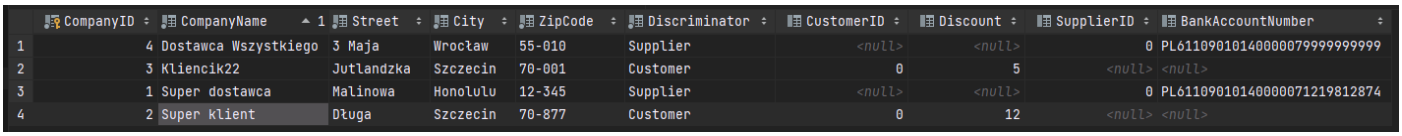
```
Wybierz, co chcesz zrobić
Możliwy wybór:
        - add
        - display
Aby wyjść, wpisz 'exit'

>>> display
Wprowadź typ firm, które chcesz wypisać
Możliwy wybór:
        - customer
        - supplier
        - all
Aby wyjść, wpisz 'exit'

>>> supplier
Lista wszystkich dostawców (firm):
[1] Super dostawca (dostawca)
[4] Dostawca Wszystkiego (dostawca)
```

6.3.5. Wypisywanie wszystkich firm

```
Wybierz, co chcesz zrobić
Możliwy wybór:
        - add
        - display
Aby wyjść, wpisz 'exit'

>>> display
Wprowadź typ firm, które chcesz wypisać
Możliwy wybór:
        - customer
        - supplier
        - all
Aby wyjść, wpisz 'exit'

>>> all
Lista wszystkich klientów (firm):
[1] Super dostawca (dostawca)
[2] Super klient (klient)
[3] Kliencik22 (klient)
[4] Dostawca Wszystkiego (dostawca)
```

6.3.6. Utworzona tabela

- **Companies**



	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	CustomerID	Discount	SupplierID	BankAccountNumber
1	4	Dostawca Wszystkiego	3 Maja	Wrocław	55-010	Supplier	<null>	<null>	0	PL61109010140000079999999999
2	3	Kliencik22	Jutlandzka	Szczecin	70-001	Customer	0	5	<null>	<null>
3	1	Super dostawca	Malinowa	Honolulu	12-345	Supplier	<null>	<null>	0	PL61109010140000071219812874
4	2	Super klient	Długa	Szczecin	70-877	Customer	0	12	<null>	<null>

7. Dziedziczenie Table-Per-Type

7.1. Zmienione klasy

Przejście na dziedziczenie **Table-Per-Type** wymaga zmodyfikowania tylko 2 klas. Wystarczy dodać adnotacje nad deklaracjami tych klas.

7.1.1. Klasa Customer

Dodany został jedynie atrybut `[Table("Customers")]` nad deklaracją klasy.

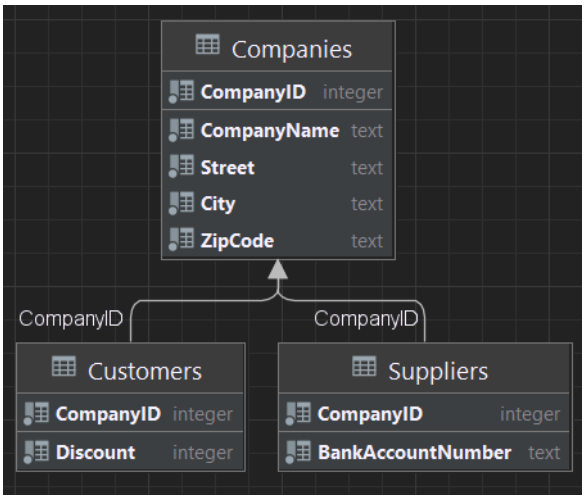
```
using System.ComponentModel.DataAnnotations.Schema;

namespace MateuszLopacinskiEFProducts
{
    [Table("Customers")]
    internal class Customer : Company
    {
        public int CompanyID { get; set; }
        public int Discount { get; set; } // In %

        public override string ToString()
        {
            return $"{base.ToString()} (klient)";
        }
    }
}
```

7.1.2. Klasa Supplier

Dodany został jedynie atrybut `[Table("Suppliers")]` nad deklaracją klasy.

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

namespace MateuszLopacinskiEFProducts
{
    [Table("Suppliers")]
    internal class Supplier : Company
    {
        public int CompanyID { get; set; }
        public string BankAccountNumber { get; set; } = String.Empty;

        public override string ToString()
        {
            return $"{base.ToString()} (dostawca)";
        }
    }
}
```

7.2. Diagram bazy danych



7.3. Przykład działania

Działanie nie różni się niczym od przedstawionego powyżej (różnice w schemacie bazy danych są maskowane przez Entity Framework i działanie programu jest takie samo). Z tego powodu nie umieszczam poniżej przykładów, ponieważ byłyby one takie same jak w podpunkcie **6.3.**.

7.3.1. Utworzone tabele

- **Companies**

	CompanyID	CompanyName	Street	City	ZipCode
1	1	Super dostawca	Malinowa	Honolulu	12-345
2	2	Super Klient	Długa	Szczecin	70-877
3	3	Kliencik22	Jutlandzka	Szczecin	70-001
4	4	Dostawca Wszystkiego	3 Maja	Wrocław	55-010

- **Customers**

	CompanyID	Discount
1	2	12
2	3	5

- **Suppliers**

	CompanyID	BankAccountNumber
1	1	PL49102028922276300500000…
2	4	PL12321414214124241232131…

8. Porównanie dziedziczenia Table-Per-Hierarchy i Table-Per-Type

8.1. Table-Per-Hierarchy

8.1.1. Charakterystyka

- Tworzona jest jedna tabela, która zawiera wspólne dla klas dziedziczących dane oraz dane, charakteryzujące każdą z klas dziedziczących z osobna,

- W przypadku, gdy klasa dziedzicząca posiada atrybut, którego nie ma w klasie, z której dziedziczy, dodawana jest osobna kolumna, w której dla pozostałych klas wpisane są wartości **null**, a dla tej klasy, odpowiednie wartości tego parametru.

8.1.2. Zalety

- Takie podejście do modelowania pozwala na zmniejszenie liczby wykonywanych operacji **join** na tabelach, w porównaniu do modelowania z wykorzystaniem **Table-Per-Type** (gdzie tworzone są osobne tabele dla każdego z typów).

8.1.3. Wady

- W przypadku wielu klas dziedziczących z tej samej klasy, jedna tabela nie jest dobrym rozwiązaniem, ponieważ będzie zawierała bardzo dużo wartości **null** (marnowanie miejsca),
- Grupowanie danych, w przypadku wielu klas dziedziczących, zmniejsza przejrzystość schematu bazy danych.

8.2. Table-Per-Type

8.2.1. Charakterystyka

- Tworzone jest kilka tabel (osobne tabele dla każdej z klas, zarówno tej, z której dziedziczą klasy, jak i klas dziedziczących),
- Tabele klas dziedziczących są łączone z tabelą klasy, z której dziedziczą, przy pomocy relacji **1 do 1**.

8.2.2. Zalety

- Takie podejście nie wymaga trzymania pustych wartości w tabelach (**null**), dzięki czemu zapisywane są tylko wartości, stanowiące dane,
- W przypadku wielu klas dziedziczących z jednej klasy, takie podejście pozwala na zwiększenie czytelności schematu bazy danych.

8.2.3. Wady

- Konieczne jest wykonywanie wielu operacji **join** (łączenie tabel klas dziedziczących z tabelą klasy nadrzędnej – tej, z której klasy dziedziczą).