

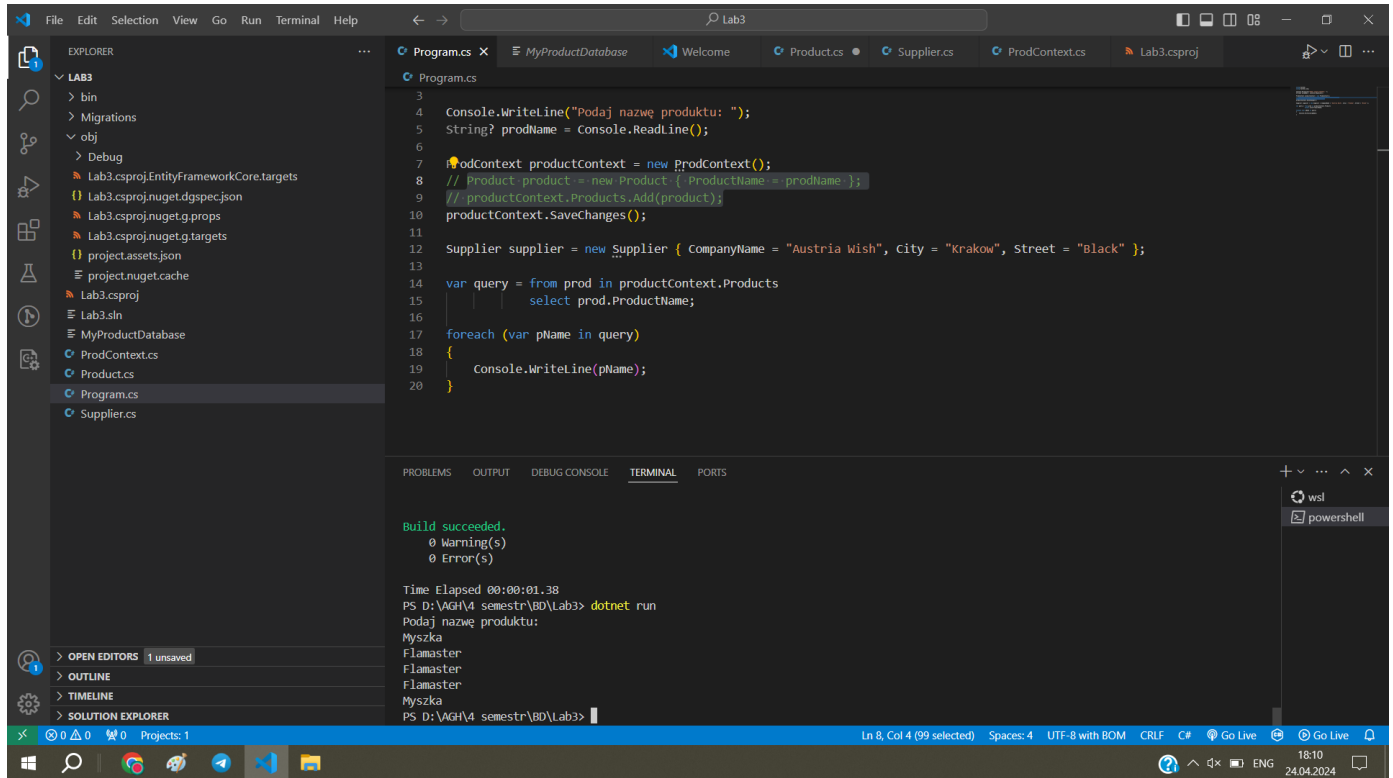
Entity Framework

ćwiczenie 3

Imiona i nazwiska autorów: Stas Kochevenko & Wiktor Dybalski

Wproradzenie

Zostały dodane podstawowe klasy Product, ProdContext oraz Program, baza danych MyProductDatabase.



Kod:

- Product.cs

```
using System.Runtime.Intrinsics.X86;

public class Product
{
    public int ProductID { get; set; }
    public String? ProductName { get; set; }
    public int UnitsInStock { get; set; }
}
```

- ProdContext.cs

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

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

```
    }  
}
```

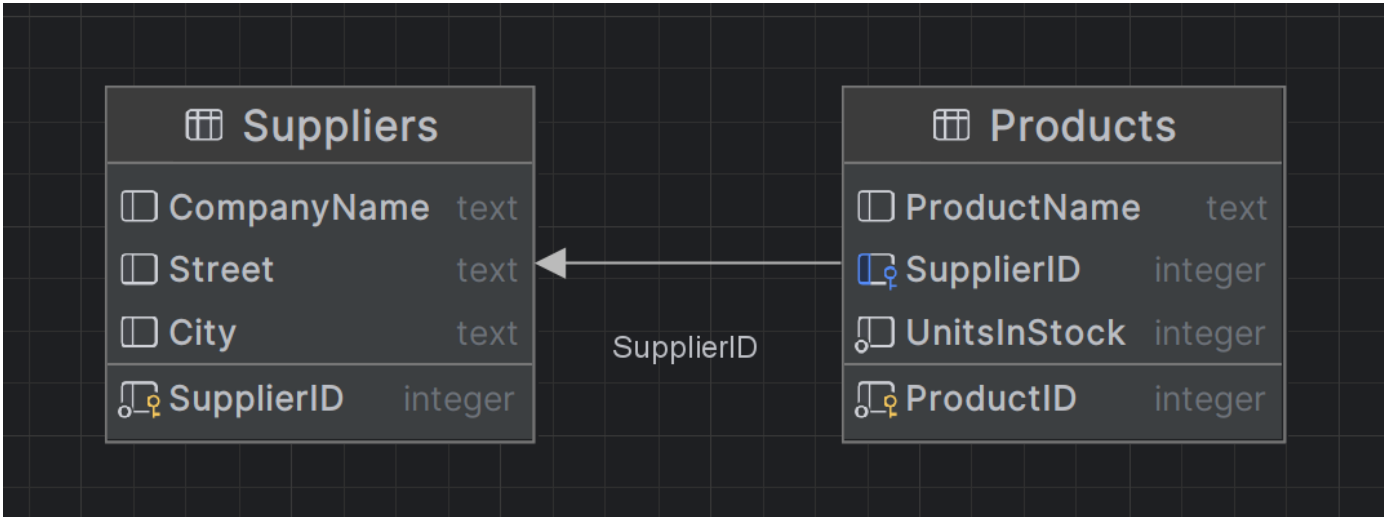
• Program.cs

```
using System;  
using System.Linq;  
  
Console.WriteLine("Podaj nazwę produktu: ");  
String? prodName = Console.ReadLine();  
  
ProdContext productContext = new ProdContext();  
  
Product product = new Product { ProductName = prodName, UnitsInStock = 24, Supplier = supplier };  
productContext.Products.Add(product);  
productContext.SaveChanges();  
  
var query = from prod in productContext.Products  
            select new { prod.ProductID, prod.ProductName };  
  
foreach (var pName in query)  
{  
    Console.WriteLine(pName);  
}
```

Zadanie a - wprowadzenie pojęcia Dostawcy

Została dodana klasa Supplier oraz zaktualizowana odpowiednio klasa Product, żeby mieć połączenie z klasą Supplier.

Schemat zmienionej bazy danych, wygenerowany przez DataGrip:



Testowanie dodania nowego dostawcy i ustawienia dostawcy poprzednio wprowadzonego produktu na dodanego dostawcę. Tabela Products przed wywołaniem metody dodającej:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	<null>	5
5	5	Klawiatura	<null>	54
6	6	Computer	<null>	100

Wynik działania programu w postaci konsolowych komunikatów:





```
Dodać nowego dostawcę? (Tak/Nie)
Tak
Podaj nazwę nowego dostawcy: Lenovo
Podaj nazwę miasta: Amsterdam
Podaj nazwę ulicy: Liberty
Został utworzony dostawca: Lenovo.

Podaj ID produktu do wyszukiwania: 6

Zmienić dostawcę dla produktu na podanego wyżej? (Tak/Nie)
Tak

Dla productu: Computer zmieniono dostawcę na: Lenovo.
Lista produktów:
1 | Laptop: 8 szt. Dostawca: undefined
2 | Telewizor: 27 szt. Dostawca: undefined
3 | Telefon: 18 szt. Dostawca: undefined
4 | Myszka: 5 szt. Dostawca: undefined
5 | Klawiatura: 54 szt. Dostawca: undefined
6 | Computer: 100 szt. Dostawca: Lenovo
Lista dostawców:
1 | Austria Wish
2 | Amour
3 | LG
6 | Lenovo
```

Sprawdźmy trwałość zmian za pomocą DataGrip:

	 ProductID	 ProductName	 SupplierID	 UnitsInStock
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	<null>	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100

Testowanie ustawienia dostawcy dla poprzednio wprowadzonego produktu. Wynik działania programu w postaci konsolowych komunikatów:





```
Dodać nowego dostawcę? (Tak/Nie)
Nie
Lista dostawców:
1 | Austria Wish
2 | Amour
3 | LG
6 | Lenovo
Podaj ID dostawcy do wyszukiwania: 3

Podaj ID produktu do wyszukiwania: 4

Zmienić dostawcę dla produktu na podanego wyżej? (Tak/Nie)
Tak

Dla produktu: Myszka zmieniono dostawcę na: LG.
Lista produktów:
1 | Laptop: 8 szt. Dostawca: undefined
2 | Telewizor: 27 szt. Dostawca: undefined
3 | Telefon: 18 szt. Dostawca: undefined
4 | Myszka: 5 szt. Dostawca: LG
5 | Klawiatura: 54 szt. Dostawca: undefined
6 | Computer: 100 szt. Dostawca: Lenovo
Lista dostawców:
1 | Austria Wish
2 | Amour
3 | LG
6 | Lenovo
```

Sprawdźmy trwałość zmian za pomocą DataGrip:

	 ProductID ↕	 ProductName ↕	 SupplierID ↕	 UnitsInStock ↕
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	3	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100

Kod:

- Product.cs

```
using System.Runtime.Intrinsics.X86;

using System.Runtime.Intrinsics.X86;

public class Product
{
    public int ProductID { get; set; }
    public String? ProductName { get; set; }
    public int UnitsInStock { get; set; }
    public Supplier? supplier { get; set; } = null;
    public override string ToString()
    {
        string supName = "undefined";
        if (supplier != null)
        {
            supName = supplier.CompanyName;
        }
    }
}
```

```
        return $"{ProductName}: {UnitsInStock} szt. Dostawca: {supName}";
    }
}
```

- Supplier.cs

```
public 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;
    }
}
```

- ProdContext.cs

```
using Microsoft.EntityFrameworkCore;
public class ProdContext : 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=MyProductDatabase");
    }
}
```

- Program.cs

```
class Program
{
    private static Product createNewProduct()
    {
        Console.WriteLine("Podaj nazwę nowego produktu: ");
        string prodName = Console.ReadLine();
        Console.WriteLine("Podaj ilość jednostek danego produktu dostępnych w sklepie: ");
        int quantity = Int32.Parse(Console.ReadLine());

        Product product = new Product { ProductName = prodName, UnitsInStock = quantity };
        Console.WriteLine($"Został utworzony produkt: {product}");

        return product;
    }

    private static Product findProduct(ProdContext productContext)
    {
        Console.WriteLine("Podaj ID produktu do wyszukiwania: ");
        int prodID = Int32.Parse(Console.ReadLine());

        var query = from prod in productContext.Products
                     where prod.ProductID == prodID
                     select prod;

        return query.FirstOrDefault();
    }

    private static void showAllProducts(ProdContext productContext)
    {
        Console.WriteLine("Lista produktów: ");
    }
}
```

```
        foreach (Product product in productContext.Products)
        {
            Console.WriteLine($"{product.ProductID} | {product}");
        }
    }

    private static Supplier createNewSupplier()
    {
        Console.WriteLine("Podaj nazwę nowego dostawcy: ");
        string companyName = Console.ReadLine();
        Console.WriteLine("Podaj nazwę miasta: ");
        string city = Console.ReadLine();
        Console.WriteLine("Podaj nazwę ulicy: ");
        string street = Console.ReadLine();

        Supplier supplier = new Supplier { CompanyName = companyName, City = city, Street = street };
        Console.WriteLine($"Został utworzony dostawca: {supplier}.\n");

        return supplier;
    }

    private static Supplier findSupplier(ProdContext productContext)
    {
        Console.WriteLine("Podaj ID dostawcy do wyszukiwania: ");
        int supplierID = Int32.Parse(Console.ReadLine());

        var query = from supplier in productContext.Suppliers
                    where supplier.SupplierID == supplierID
                    select supplier;

        return query.FirstOrDefault();
    }

    private static void showAllSuppliers(ProdContext productContext)
    {
        Console.WriteLine("Lista dostawców: ");
        foreach (Supplier supplier in productContext.Suppliers)
        {
            Console.WriteLine($"{supplier.SupplierID} | {supplier}");
        }
    }

    static void Main()
    {
        ProdContext productContext = new ProdContext();

        Supplier supplier = null;

        Console.WriteLine("Dodać nowego dostawcę? (Tak/Nie)");
        string choice = Console.ReadLine();
        bool correctAnswer = false;
        do
        {
            switch (choice)
            {
                case "Tak":
                    supplier = createNewSupplier();
                    productContext.Suppliers.Add(supplier);
                    correctAnswer = true;
                    break;
                case "Nie":
                    showAllSuppliers(productContext);
                    supplier = findSupplier(productContext);
                    correctAnswer = true;
                    break;
            }
        } while (!correctAnswer);

        productContext.SaveChanges();
    }
}
```

```

Console.WriteLine("");

Product product = findProduct(productContext);

Console.WriteLine("");

Console.WriteLine("Zmienić dostawcę dla produktu na podanego wyżej? (Tak/Nie)");
choice = Console.ReadLine();
correctAnswer = false;
do
{
    switch (choice)
    {
        case "Tak":
            product.supplier = supplier;
            Console.WriteLine($"Dla produktu: {product.ProductName} zmieniono dostawcę na: {supplier.CompanyName}.");
            productContext.SaveChanges();
            correctAnswer = true;
            break;
        case "Nie":
            correctAnswer = true;
            break;
    }
} while (!correctAnswer);

showAllProducts(productContext);

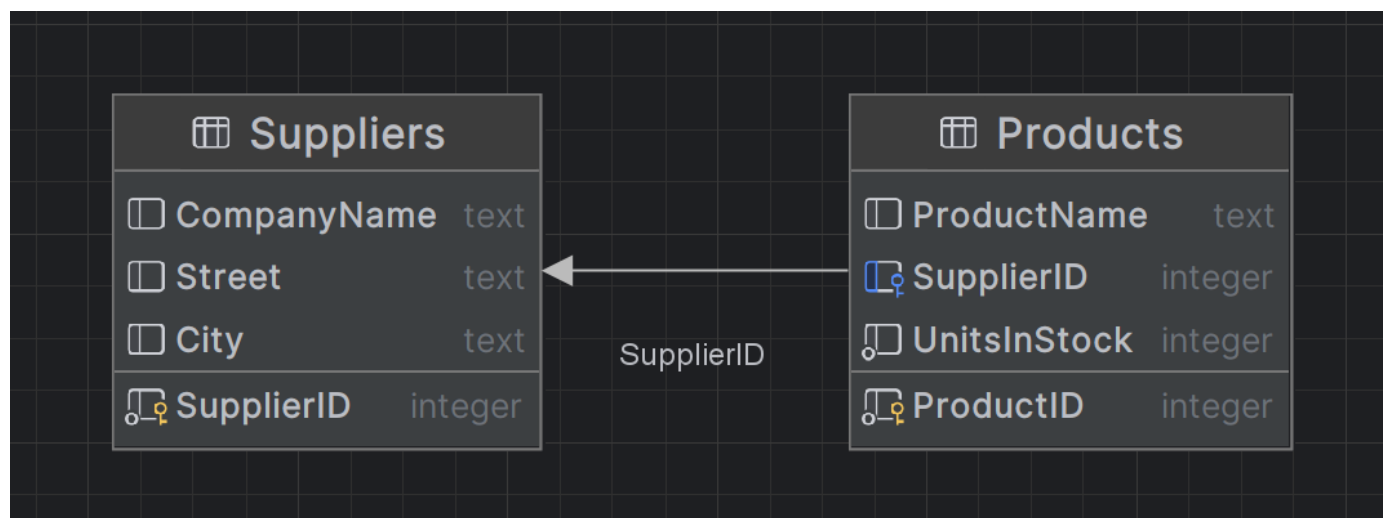
showAllSuppliers(productContext);
}

```

Zadanie b - odwrócenie relacji Supplier -> Product

- Z klasy "Product" został usunięty atrybut "Supplier" zgodnie ze schematem.
- W klasie "Supplier" dodano kolekcję produktów, dostarczanych przez danego dostawcę.
- W kodzie głównego programu został dodany fragment kodu odpowiedzialny za dodanie nowych produktów. W porównaniu do zadania 1, jedyną zmianą było ustawienie produktu do dostawcy, zamiast ustawienia dostawcy do produktu. Niżej będzie podany fragment kodu, który uległ zmianie.
- Klasa ProdContext pozostała bez zmian.

Schemat zmienionej bazy danych, wygenerowany przez DataGrip:



Jak widać, pomimo odwrócenia relacji, schemat bazy danych się nie zmienił. Entity Framework dokonał optymalizacji, dzięki czemu uniknęliśmy powielania danych w tabeli Suppliers (ponieważ nie możemy trzymać listy w polu). Oprócz tego takie podejście miało by gorsze konsekwencje: każdy rekord miałby nowy SupplierID, mimo że byłby to ten sam dostawca, jedynie mający przypisany inny ProductID. W takiej sytuacji nie byłibyśmy w stanie odróżnić dostawców.

Testowanie dodania nowych produktów i ustawienia ich dostawcy na nowo stworzonego. Tabela Products przed wywołaniem metody dodającej:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	3	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100

Wynik działania programu w postaci konsolowych komunikatów:

```
Dodać nowego dostawcę? (Tak/Nie)
Tak
Podaj nazwę nowego dostawcy: Samsung
Podaj nazwę miasta: Sietl
Podaj nazwę ulicy: Beach street
Został utworzony dostawca: Samsung.
Dodać nowy produkt? (Tak/Nie)
Tak
Podaj nazwę nowego produktu: Tablet
Podaj ilość jednostek danego produktu dostępnych w sklepie: 8
Został utworzony produkt: Tablet: 8 szt.
Dodać nowy produkt? (Tak/Nie)
Tak
Podaj nazwę nowego produktu: Clock
Podaj ilość jednostek danego produktu dostępnych w sklepie: 4
Został utworzony produkt: Clock: 4 szt.
Dodać nowy produkt? (Tak/Nie)
Nie

Podaj ID produktu do wyszukiwania: 8

Zmienić dostawcę dla produktu na podanego wyżej? (Tak/Nie)
Tak

Dla productu: Clock zmieniono dostawcę na: Samsung.
Lista produktów:
1 | Laptop: 8 szt.
2 | Telewizor: 27 szt.
3 | Telefon: 18 szt.
4 | Myszka: 5 szt.
5 | Klawiatura: 54 szt.
6 | Computer: 100 szt.
7 | Tablet: 8 szt.
8 | Clock: 4 szt.
Lista dostawców:
1 | Austria Wish
2 | Amour
3 | LG
6 | Lenovo
7 | Samsung
```


Sprawdźmy trwałość zmian za pomocą DataGrip. Tabela Product po dodaniu produktów:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	3	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100
7	7	Tablet	<null>	8
8	8	Clock	<null>	4

Tabela Product po wprowadzeniu zmiany dostawcy:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	3	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100
7	7	Tablet	<null>	8
8	8	Clock	7	4

Kod:

- Product.cs

```
using System.Runtime.Intrinsics.X86;

public class Product
{
    public int ProductID { get; set; }
    public String? ProductName { get; set; }
    public int UnitsInStock { get; set; }
    public override string ToString()
    {
        return $"{ProductName}: {UnitsInStock} szt.";
    }
}
```

- Supplier.cs

```
public 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;
    }
}
```

- Program.cs (fragment starego kodu)

```
static void Main()
{
    ...
    product.supplier = supplier;
    ...
}
```

- Program.cs (fragment nowego kodu)

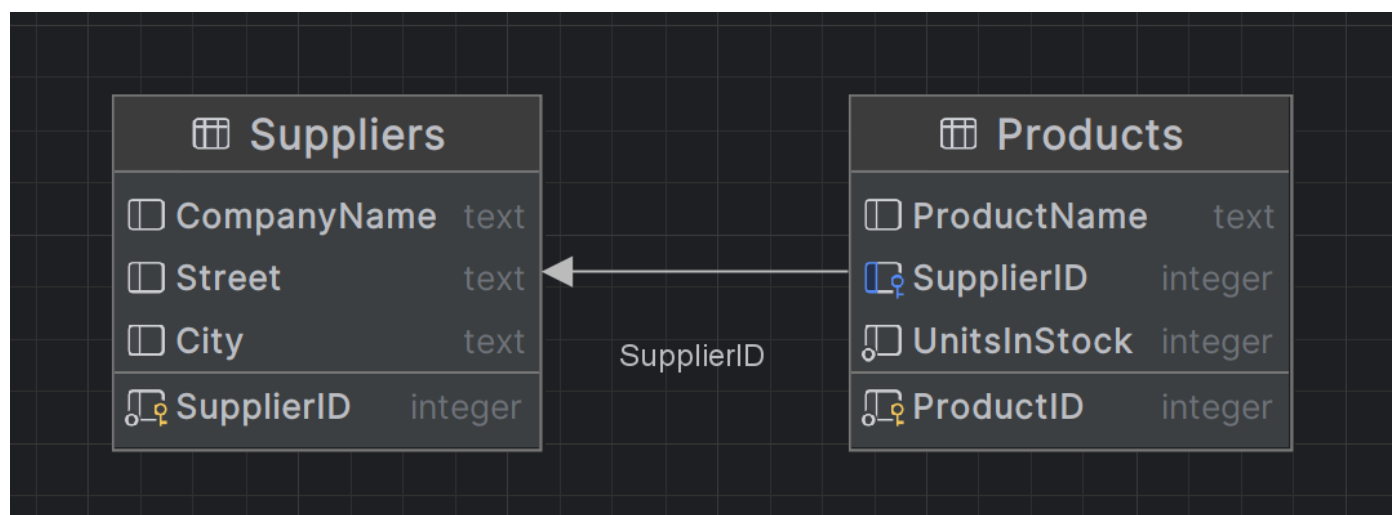
```
static void Main()
{
    ...

    do {
        Console.WriteLine("Dodać nowy produkt? (Tak/Nie)");
        choice = Console.ReadLine();
        correctAnswer = false;
        switch (choice)
        {
            case "Tak":
                product = createNewProduct();
                productContext.Products.Add(product);
                correctAnswer = true;
                break;
            case "Nie":
                correctAnswer = true;
                break;
        }
        Console.WriteLine("");
    } while (!correctAnswer || choice == "Tak");
    ...
    supplier.Products.Add(product);
    ...
}
```

Zadanie c - dwustronna relacja Supplier <----> Product

- Do klasy "Product" został dodany atrybut, określający Dostawcę danego produktu (tak jak w zad. 2).
- Klasa "Supplier" pozostała bez zmian względem implementacji poprzedniego zadania.
- W kodzie jedyną zmianą było dodanie ustawienia dostawcy do produktu. Niżej będzie podany fragment kodu, który uległ zmianie.
- Klasa ProdContext pozostała bez zmian.

Schemat zmienionej bazy danych, wygenerowany przez DataGrip:



Jak widać, pomimo utworzenia dwukierunkowej relacji, schemat bazy danych nadal się nie zmienił. Z tego wynika, że Entity Framework dokonuje optymalizacji, pozostawiając tylko niezbędną relację w bazie danych, natomiast pozwala na utworzenie dwukierunkowej relacji (można również stworzyć ją w przeciwnym kierunku), aby mieć możliwość łatwiejszego manipulowania powiązаныmi między sobą obiektami. W taki sposób Entity Framework unika powielania danych w tabeli Suppliers.

Testowanie dodania nowych produktów i ustawienia ich dostawcy na nowo stworzonego. Tabela Products przed wywołaniem metody dodającej:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	3	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100
7	7	Tablet	<null>	8
8	8	Clock	7	4

Wynik działania programu w postaci konsolowych komunikatów:

```
Dodać nowego dostawcę? (Tak/Nie)
Nie
Lista dostawców:
1 | Austria Wish
2 | Amour
3 | LG
6 | Lenovo
7 | Samsung
Podaj ID dostawcy do wyszukiwania: 3
Dodać nowy produkt? (Tak/Nie)
Tak
Podaj nazwę nowego produktu: Microwave
Podaj ilość jednostek danego produktu dostępnych w sklepie: 18
Został utworzony produkt: Microwave: 18 szt. Dostawca: undefined
Dodać nowy produkt? (Tak/Nie)
Tak
Podaj nazwę nowego produktu: Vacuum cleaner
Podaj ilość jednostek danego produktu dostępnych w sklepie: 4
Został utworzony produkt: Vacuum cleaner: 4 szt. Dostawca: undefined
```




```
Dodać nowy produkt? (Tak/Nie)
Tak
Podaj nazwę nowego produktu: Headphones
Podaj ilość jednostek danego produktu dostępnych w sklepie: 1
Został utworzony produkt: Headphones: 1 szt. Dostawca: undefined
Dodać nowy produkt? (Tak/Nie)
Nie

Podaj ID produktu do wyszukiwania: 11

Zmienić dostawcę dla produktu na podanego wyżej? (Tak/Nie)
Tak

Dla productu: Headphones zmieniono dostawcę na: LG.
Lista produktów:
1 | Laptop: 8 szt. Dostawca: undefined
2 | Telewizor: 27 szt. Dostawca: undefined
3 | Telefon: 18 szt. Dostawca: undefined
4 | Myszka: 5 szt. Dostawca: LG
5 | Klawiatura: 54 szt. Dostawca: undefined
6 | Computer: 100 szt. Dostawca: Lenovo
7 | Tablet: 8 szt. Dostawca: undefined
8 | Clock: 4 szt. Dostawca: Samsung
9 | Microwave: 18 szt. Dostawca: undefined
10 | Vacuum cleaner: 4 szt. Dostawca: undefined
11 | Headphones: 1 szt. Dostawca: LG
Lista dostawców:
1 | Austria Wish
2 | Amour
3 | LG
6 | Lenovo
7 | Samsung
```

Sprawdźmy trwałość zmian za pomocą DataGrip. Tabela Product po dodaniu produktów:

	 ProductID ▾	 ProductName ▾	 SupplierID ▾	 UnitsInStock ▾
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	3	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100
7	7	Tablet	<null>	8
8	8	Clock	7	4
9	9	Microwave	<null>	18
10	10	Vacuum cleaner	<null>	4
11	11	Headphones	<null>	1

Sprawdźmy trwałość zmian za pomocą DataGrip. Tabela Product po wprowadzeniu zmiany dostawcy:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Laptop	<null>	8
2	2	Telewizor	<null>	27
3	3	Telefon	<null>	18
4	4	Myszka	3	5
5	5	Klawiatura	<null>	54
6	6	Computer	6	100
7	7	Tablet	<null>	8
8	8	Clock	7	4
9	9	Microwave	<null>	18
10	10	Vacuum cleaner	<null>	4
11	11	Headphones	3	1

Kod:

- Product.cs

```
using System.Runtime.Intrinsics.X86;

using System.Runtime.Intrinsics.X86;

public class Product
{
    public int ProductID { get; set; }
    public String? ProductName { get; set; }
    public int UnitsInStock { get; set; }
    public Supplier? supplier { get; set; } = null;
    public override string ToString()
    {
        string supName = "undefined";
        if (supplier != null)
        {
            supName = supplier.CompanyName;
        }
        return $"{ProductName}: {UnitsInStock} szt. Dostawca: {supName}";
    }
}
```

- Program.cs (fragment starego kodu)

```
static void Main()
{
    ...
    supplier.Products.Add(product);
    ...
}
```

- Program.cs (fragment nowego kodu)

```
static void Main()
{
    ...
    supplier.Products.Add(product);
    product.supplier = supplier;
    ...
}
```

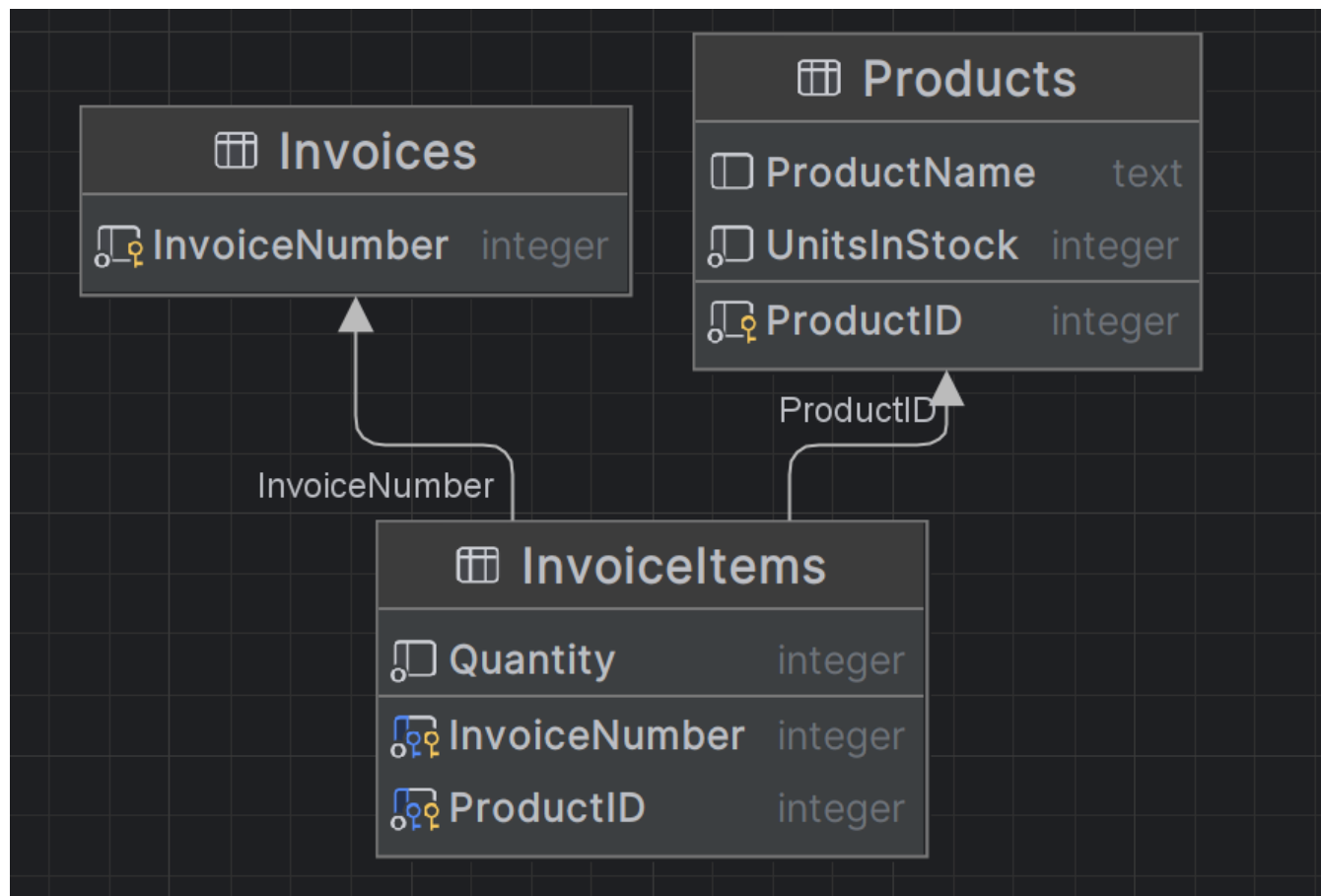
Zadanie d - modelowanie relacji wiele-do-wielu

- Została usunięta klasa "Supplier", skoro nie jest potrzebna w ramach danego zadania.
- Pojawiła się klasa "Invoice" odpowiednio do treści zadania, w której mieści się kolekcja pozycji "InvoiceItem" sprzedanych w ramach danej transakcji.
- Dodaliśmy pomocniczą klasę "InvoiceItem", która reprezentuje pojedynczą pozycję transakcji: numer produktu oraz ilość zakupionych produktów o tym numerze w ramach danej transakcji. Ma ona również navigation properties do klasy "Product" oraz "Invoice".
- W klasie "Product" została dodana dodatkowa metoda wypisywania oraz navigation properties do klasy "InvoiceItem".
- Klasę ProdContext rozszerzyliśmy o zbiory "Invoice" oraz "InvoiceItem", jednocześnie usunęliśmy zbiór "Suppliers". Oprócz tego, wyznaczyliśmy Composite PrimaryKey.
- Kod głównego programu został dopasowany do funkcjonalności, potrzebnych w tym zadaniu. Pojawiła się możliwość wyboru wśród 10 opcji akcji:

```
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT
```

Podaj jedną liczbę z zakresu 1-10

Schemat zmienionej bazy danych, wygenerowany przez DataGrip:



- Stwórmmy kilka produktów

Tabela Products przed wywołaniem metody dodającej:

	ProductID	ProductName	UnitsInStock
1	1	Laptop	8
2	2	Telewizor	27
3	3	Telefon	18
4	4	Myszka	5
5	5	Klawiatura	54

Wynik działania programu w postaci konsolowych komunikatów:

```
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10
1

Podaj nazwę nowego produktu: Computer
Podaj ilość jednostek danego produktu dostępnych w sklepie: 100
Został utworzony produkt: Computer
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10
1

Podaj nazwę nowego produktu: Tablet
Podaj ilość jednostek danego produktu dostępnych w sklepie: 8
Został utworzony produkt: Tablet
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10
1
```

```

1
Podaj nazwę nowego produktu: Clock
Podaj ilość jednostek danego produktu dostępnych w sklepie: 4
Został utworzony produkt: Clock
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

```

Sprawdźmy trwałość zmian za pomocą DataGrip. Tabela Product po dodaniu produktów:

	ProductID	ProductName	UnitsInStock
1	1	Laptop	8
2	2	Telewizor	27
3	3	Telefon	18
4	4	Myszka	5
5	5	Klawiatura	54
6	13	Computer	100
7	14	Tablet	8
8	15	Clock	4

- Następnie dodamy produkty do koszyka i kupimy je w kilku transakcjach

Wynik działania programu w postaci konsolowych komunikatów (pierwsza transakcja):

```

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10
4

Product #1 Laptop: 8 szt.
Product #2 Telewizor: 27 szt.
Product #3 Telefon: 18 szt.
Product #4 Myszka: 5 szt.
Product #5 Klawiatura: 54 szt.
Product #13 Computer: 100 szt.
Product #14 Tablet: 8 szt.
Product #15 Clock: 4 szt.

Podaj ID produktu, który należy dodać do koszyka: 13
Podaj ilość sztuk produktu: 5
Do koszyka został dodany produkt o numerze 13
1. Add new product
2. Show all products

```



```
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT
```

Podaj jedną liczbę z zakresu 1-10

4

Product #1 Laptop: 8 szt.
Product #2 Telewizor: 27 szt.
Product #3 Telefon: 18 szt.
Product #4 Myszka: 5 szt.
Product #5 Klawiatura: 54 szt.
Product #13 Computer: 100 szt.
Product #14 Tablet: 8 szt.
Product #15 Clock: 4 szt.

Podaj ID produktu, który należy dodać do koszyka: 14

Podaj ilość sztuk produktu: 2

Do koszyka został dodany produkt o numerze 14

```
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT
```

Podaj jedną liczbę z zakresu 1-10

4

Product #1 Laptop: 8 szt.
Product #2 Telewizor: 27 szt.
Product #3 Telefon: 18 szt.
Product #4 Myszka: 5 szt.
Product #5 Klawiatura: 54 szt.
Product #13 Computer: 100 szt.
Product #14 Tablet: 8 szt.
Product #15 Clock: 4 szt.

Podaj ID produktu, który należy dodać do koszyka: 5

Podaj ilość sztuk produktu: 3

Do koszyka został dodany produkt o numerze 5

```
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
```

10. EXIT

Podaj jedną liczbę z zakresu 1-10

6

Product #13 Computer: 5 szt.

Product #14 Tablet: 2 szt.

Product #5 Klawiatura: 3 szt.

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10

7

Produkty w koszyku zostały kupione

Wynik działania programu w postaci konsolowych komunikatów (druga transakcja):

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10

4

Product #1 Laptop: 8 szt.

Product #2 Telewizor: 27 szt.

Product #3 Telefon: 18 szt.

Product #4 Myszka: 5 szt.

Product #5 Klawiatura: 51 szt.

Product #13 Computer: 95 szt.

Product #14 Tablet: 6 szt.

Product #15 Clock: 4 szt.

Podaj ID produktu, który należy dodać do koszyka: 12

Podaj ID produktu, który należy dodać do koszyka: 13

Podaj ilość sztuk produktu: 2

Do koszyka został dodany produkt o numerze 13

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10

4

Product #1 Laptop: 8 szt.

Product #2 Telewizor: 27 szt.

Product #3 Telefon: 18 szt.

Product #4 Myszka: 5 szt.

Product #5 Klawiatura: 51 szt.

Product #13 Computer: 95 szt.

Product #14 Tablet: 6 szt.

Product #15 Clock: 4 szt.

Podaj ID produktu, który należy dodać do koszyka: 14

Podaj ilość sztuk produktu: 1

Do koszyka został dodany produkt o numerze 14

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10

7

Produkty w koszyku zostały kupione

Wynik działania programu w postaci konsolowych komunikatów (trzecia transakcja):

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10

4

Product #1 Laptop: 8 szt.
Product #2 Telewizor: 27 szt.
Product #3 Telefon: 18 szt.
Product #4 Myszka: 5 szt.
Product #5 Klawiatura: 51 szt.
Product #13 Computer: 93 szt.
Product #14 Tablet: 5 szt.
Product #15 Clock: 4 szt.

Podaj ID produktu, który należy dodać do koszyka: 13

Podaj ilość sztuk produktu: 1

Do koszyka został dodany produkt o numerze 13

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10

4

Product #1 Laptop: 8 szt.
Product #2 Telewizor: 27 szt.
Product #3 Telefon: 18 szt.
Product #4 Myszka: 5 szt.
Product #5 Klawiatura: 51 szt.
Product #13 Computer: 93 szt.
Product #14 Tablet: 5 szt.
Product #15 Clock: 4 szt.

Podaj ID produktu, który należy dodać do koszyka: 5

Podaj ilość sztuk produktu: 8

Do koszyka został dodany produkt o numerze 5

1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10

7

Produkty w koszyku zostały kupione

Przekonamy się, że liczba produktów w sklepie została zmniejszona za pomocą DataGrip. Tabela Product po zakupieniu produktów:

	ProductID	ProductName	UnitsInStock
1	1	Laptop	8
2	2	Telewizor	27
3	3	Telefon	18
4	4	Myszka	5
5	5	Klawiatura	43
6	13	Computer	92
7	14	Tablet	5
8	15	Clock	4

Tabela Invoice po przeprowadzeniu 3 transakcji:

	InvoiceNumber
1	1
2	2
3	3

Tabela InvoiceItems po przeprowadzeniu 3 transakcji:

	InvoiceNumber	ProductID	Quantity
1	1	5	3
2	1	13	5
3	1	14	2
4	2	13	2
5	2	14	1
6	3	5	8
7	3	13	1

- Produkty, które zostały sprzedane w ramach wybranej transakcji:

```
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT

Podaj jedną liczbę z zakresu 1-10
8

Podaj ID transakcji do wypisywania zakupionych pozycji: 1

Invoice 1:
    1) Product #5 Klawiatura: 3 szt.
    2) Product #13 Computer: 5 szt.
    3) Product #14 Tablet: 2 szt.
```

- Faktury, w ramach których został sprzedany produkt

```
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT
```

Podaj jedną liczbę z zakresu 1-10

9

Product #1 Laptop: 8 szt.

Product #2 Telewizor: 27 szt.

Product #3 Telefon: 18 szt.

Product #4 Myszka: 5 szt.

Product #5 Klawiatura: 43 szt.

Product #13 Computer: 92 szt.

Product #14 Tablet: 5 szt.

Product #15 Clock: 4 szt.

Podaj ID produktu, dla którego należy wyszukać transakcji: 13

Invoice #1

Invoice #2

Invoice #3

```
1. Add new product
2. Show all products
3. Show available products
4. Add product to basket
5. Remove product from basket
6. Show products in basket
7. Buy products in basket
8. Show products sold in transaction
9. Show invoices having sold the product
10. EXIT
```

Podaj jedną liczbę z zakresu 1-10

9

Product #1 Laptop: 8 szt.

Product #2 Telewizor: 27 szt.

Product #3 Telefon: 18 szt.

Product #4 Myszka: 5 szt.

Product #5 Klawiatura: 43 szt.

Product #13 Computer: 92 szt.

Product #14 Tablet: 5 szt.

Product #15 Clock: 4 szt.

Podaj ID produktu, dla którego należy wyszukać transakcji: 14

Invoice #1

Invoice #2

Kod:

- Product.cs

```
using System.Runtime.Intrinsics.X86;

public 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 $"Product #{ProductID} {ProductName}";
    }

    public void printProductInStock()
    {
        Console.WriteLine($"Product #{ProductID} {ProductName}: {UnitsInStock} szt.");
    }
}
```

- ProdContext.cs

```
using Microsoft.EntityFrameworkCore;
public class ProdContext : 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=MyProductDatabase");
    }

    // Set composite primary key for InvoiceItems
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<InvoiceItem>()
            .HasKey(x => new { x.InvoiceNumber, x.ProductID });
    }
}
```

- Invoice.cs

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

public class Invoice
{
    // Primary key
    [Key]
    public int InvoiceNumber { get; set; }
    public ICollection<InvoiceItem> InvoiceItems { get; set; }
    public override string ToString()
    {
        StringBuilder sb = new($"Invoice {InvoiceNumber}:");
        int i = 0;
        foreach (InvoiceItem item in InvoiceItems)
```

```

        {
            i++;
            sb.Append($"{n\t{i}} {item}");
        }
        return sb.ToString();
    }
}

```

- InvoiceItem.cs (klasa pomocnicza)

```

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

public class InvoiceItem
{
    // Composite primary key
    [Key, Column(Order = 0)]
    public int InvoiceNumber { get; set; }

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

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

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

```

- Program.cs

```

using System;
using System.Data.Common;
using System.Linq;
using System.Runtime.CompilerServices;
using Microsoft.EntityFrameworkCore.Diagnostics;
using Microsoft.EntityFrameworkCore;

class Program
{
    private static void createNewProduct(ProdContext productContext)
    {
        Console.WriteLine("Podaj nazwę nowego produktu: ");
        string prodName = Console.ReadLine();
        Console.WriteLine("Podaj ilość jednostek danego produktu dostępnych w sklepie: ");
        int quantity = Int32.Parse(Console.ReadLine());

        Product product = new Product { ProductName = prodName, UnitsInStock = quantity };
        Console.WriteLine($"Został utworzony produkt: {product.ProductName}");

        productContext.Products.Add(product);
        productContext.SaveChanges();
    }

    private static void addProductToBasket(ProdContext productContext, List<InvoiceItem> basketItems)
    {
        int prodID, prodQuantity;
        bool correctQuantity = false;
        showAvailableProducts(productContext);
        do
        {

```



```
        Console.WriteLine("\nPodaaj ID produktu, który należy dodać do koszyka: ");
        prodID = Int32.Parse(Console.ReadLine());
        Console.WriteLine();
    } while (!productAvailable(productContext, prodID));

    do
    {
        Console.WriteLine();
        Console.WriteLine("Podaaj ilość sztuk produktu: ");
        prodQuantity = Int32.Parse(Console.ReadLine());

        var query = from product in productContext.Products
                     where product.ProductID == prodID && product.UnitsInStock >= prodQuantity
                     select product;
        if ((query?.Count() > 0))
        {
            correctQuantity = true;
        }
        else
        {
            Console.WriteLine("W sklepie nie ma takiej ilości produktów");
        }
    } while (!correctQuantity);
    InvoiceItem item = new InvoiceItem { ProductID = prodID, Quantity = prodQuantity };
    basketItems.Add(item);
    Console.WriteLine($"Do koszyka został dodany produkt o numerze {prodID}");
    productContext.SaveChanges();
}

private static void removeProductFromBasket(ProdContext productContext, List<InvoiceItem> basketItems)
{
    int prodID;
    showProductsInBasket(productContext, basketItems);
    do
    {
        Console.WriteLine("\nPodaaj ID produktu, który należy usunąć z koszyka: ");
        prodID = Int32.Parse(Console.ReadLine());
        Console.WriteLine();
    } while (!productExists(productContext, prodID));

    InvoiceItem item_to_remove = null;
    foreach (InvoiceItem item in basketItems)
    {
        if (item.ProductID == prodID) item_to_remove = item;
    }
    if (item_to_remove != null)
    {
        basketItems.Remove(item_to_remove);
        Console.WriteLine($"Z koszyka został usunięty produkt o numerze {prodID}");
    }
    else { Console.WriteLine($"W koszyku nie znaleziono produktu o numerze {prodID}"); }
}

private static void buyProductsInBasket(ProdContext prodContext, List<InvoiceItem> basketItems)
{
    Invoice invoice = CreateInvoice(basketItems);
    prodContext.Invoices.Add(invoice);
    foreach (InvoiceItem item in basketItems)
    {
        var product = prodContext.Products.FirstOrDefault(prod => prod.ProductID == item.ProductID);
        product.UnitsInStock -= item.Quantity;
    }
    prodContext.SaveChanges();
    Console.WriteLine("Produkty w koszyku zostały kupione");
}

private static bool productAvailable(ProdContext productContext, int UserProductID)
{
    var query = from product in productContext.Products
                 where product.ProductID == UserProductID && product.UnitsInStock > 0
```

```
        select product;
    return (query?.Count() > 0);
}

private static bool productExists(ProdContext productContext, int UserProductID)
{
    var query = from product in productContext.Products
        where product.ProductID == UserProductID
        select product;
    return (query?.Count() > 0);
}

private static void showAllProducts(ProdContext prodContext)
{
    var query = from product in prodContext.Products
        select product;
    foreach (var product in query)
    {
        product.printProductInStock();
    }
    if (query.Count() == 0)
    {
        Console.WriteLine("Nie znaleziono produktów w bazie danych");
    }
}

private static void showAvailableProducts(ProdContext prodContext)
{
    var query = from product in prodContext.Products
        where product.UnitsInStock > 0
        select product;
    foreach (var product in query)
    {
        product.printProductInStock();
    }
    if (query?.Count() == 0)
    {
        Console.WriteLine("Nie znaleziono dostępnych do zakupu produktów w bazie danych");
    }
}

private static void showProductsInBasket(ProdContext prodContext, List<InvoiceItem> basketItems)
{
    foreach (InvoiceItem item in basketItems)
    {
        var product = prodContext.Products
            .Include(ii => ii.InvoiceItems)
            .FirstOrDefault(ii => ii.ProductID == item.ProductID);

        Console.WriteLine($"{product}: {item.Quantity} szt.");
    }
    if (basketItems.Count() == 0)
    {
        Console.WriteLine("Nie znaleziono produktów w koszyku");
    }
}

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

private static void showSoldInTransaction(ProdContext productContext)
{
    int invoiceNumber;

    Console.Write("Podaj ID transakcji do wypisywania zakupionych pozycji: ");
    invoiceNumber = Int32.Parse(Console.ReadLine());
    Console.WriteLine();
}
```

```
var invoice = productContext.Invoices
    .Include(inv => inv.InvoiceItems)
    .ThenInclude(item => item.Product)
    .FirstOrDefault(inv => inv.InvoiceNumber == invoiceNumber);

if (invoice != null)
{
    Console.WriteLine(invoice);
}
else
{
    Console.WriteLine("Nie znaleziono transakcji o takim ID");
}
}

private static void showInvoicesIncludeProduct(ProdContext productContext)
{
    int prodID;
    showAllProducts(productContext);
    do
    {
        Console.Write("Podaj ID produktu, dla którego należy wyszukać transakcji: ");
        prodID = Int32.Parse(Console.ReadLine());
        Console.WriteLine();
    } while (!productExists(productContext, prodID));

    var query = from item in productContext.InvoiceItems.Include(ii => ii.Invoice)
        where item.ProductID == prodID
        select item;

    foreach (var item in query)
    {
        Console.WriteLine("Invoice #{0}", item.InvoiceNumber);
    }
    if (query?.Count() == 0)
    {
        Console.WriteLine("Nie znaleziono transakcji, w których by występował dany produkt");
    }
}

private static void showOptions()
{
    Console.WriteLine("1. Add new product");
    Console.WriteLine("2. Show all products");
    Console.WriteLine("3. Show available products");
    Console.WriteLine("4. Add product to basket");
    Console.WriteLine("5. Remove product from basket");
    Console.WriteLine("6. Show products in basket");
    Console.WriteLine("7. Buy products in basket");
    Console.WriteLine("8. Show products sold in transaction");
    Console.WriteLine("9. Show invoices having sold the product");
    Console.WriteLine("10. EXIT");
}

private static int getUserChoice()
{
    int choice = 0;
    string input;
    do
    {
        Console.WriteLine();
        showOptions();
        Console.WriteLine("\nPodaj jedną liczbę z zakresu 1-10");
        input = Console.ReadLine();
        int.TryParse(input, out choice);
    } while (choice < 0 || choice > 10);
    Console.WriteLine();
    return choice;
}

static void Main()
{
    ProdContext productContext = new ProdContext();
```

```

List<InvoiceItem> basketItems = new List<InvoiceItem>();

bool exited = false;

while (!exited)
{
    switch (getUserChoice())
    {
        case 1: // Add new product
            createNewProduct(productContext);
            break;
        case 2: // Show all products
            showAllProducts(productContext);
            break;
        case 3: // Show available products
            showAvailableProducts(productContext);
            break;
        case 4: // Add product to basket
            addProductToBasket(productContext, basketItems);
            break;
        case 5: // Remove product from basket
            removeProductFromBasket(productContext, basketItems);
            break;
        case 6: // Show products in basket
            showProductsInBasket(productContext, basketItems);
            break;
        case 7: // Buy products in basket
            buyProductsInBasket(productContext, basketItems);
            basketItems = new List<InvoiceItem>();
            break;
        case 8: // Show products sold in transaction
            showSoldInTransaction(productContext);
            break;
        case 9: // Show invoices having sold the product
            showInvoicesIncludeProduct(productContext);
            break;
        case 10: // EXIT
            exited = true;
            break;
    }
}
}
}

```

Zadanie e - Table-Per-Hierarchy:

Zgodnie ze strategią Table-Per-Hierarchy, tworzymy jedną tabelę Company, która przechowuje wszystkie typy klas Company.cs:

```

namespace zad5;

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}";
    }
}

```

Supplier.cs:

```
namespace zad5;

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

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

Customer.cs:

```
namespace zad5;

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

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

CompanyContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace zad5;

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");
    }
}
```

CreateCompany()

```
private static Company? CreateCompany()
{
    var companyType = "";
    do
    {
        Console.Write("Podaj typ nowej firmy (Supplier/Customer): ");
        companyType = Console.ReadLine();
    }
    while (companyType?.Trim().ToLower() != "supplier" && companyType?.Trim().ToLower() != "customer");

    Console.Write("Podaj nazwę nowej firmy: ");
    var companyName = Console.ReadLine();
}
```

```

Console.Write("Podaj ulicę nowej firmy: ");
var companyStreet = Console.ReadLine();

Console.Write("Podaj miasto nowej firmy: ");
var companyCity = Console.ReadLine();

Console.Write("Podaj kod pocztowy nowej firmy: ");
var companyZipCode = Console.ReadLine();

switch (companyType?.Trim().ToLower())
{
    case "supplier":
        Console.Write("Podaj numer konta bankowego nowej firmy: ");
        String? companyBankAccountNumber = Console.ReadLine();
        return new Supplier
        {
            CompanyName = companyName,
            City = companyCity,
            Street = companyStreet,
            ZipCode = companyZipCode,
            BankAccountNumber = companyBankAccountNumber
        };
    case "customer":
        Console.Write("Podaj zniżkę nowej firmy: ");
        int companyDiscount = int.Parse(Console.ReadLine());
        return new Customer
        {
            CompanyName = companyName,
            City = companyCity,
            Street = companyStreet,
            ZipCode = companyZipCode,
            Discount = companyDiscount,
        };
    default:
        Console.WriteLine("Nie podano typu firmy!");
        return null;
}
}

```

FindCompany()

```

private static Company? FindCompany(CompanyContext companyContext)
{
    Console.Write("Podaj ID firmy do wyszukiwania: ");
    var companyId = int.Parse(Console.ReadLine());

    var query = from comp in companyContext.Companies
                where comp.CompanyID == companyId
                select comp;

    return query.FirstOrDefault();
}

```

ShowAllSuppliers()

```

private static void ShowAllSuppliers(CompanyContext companyContext)
{
    Console.WriteLine("Lista dostawców: ");

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

```

ShowAllCustomers()

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

ShowAllCompanies()

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

RemoveCompany

```
private static void RemoveCompany(CompanyContext companyContext)
{
    Console.WriteLine("Podaj ID firmy do usunięcia: ");
    var companyId = int.Parse(Console.ReadLine());
    var company = companyContext.Companies.FirstOrDefault(comp => comp.CompanyID == companyId);

    if (company == null)
    {
        Console.WriteLine("Nie znaleziono firmy o podanym ID.");
        return;
    }

    companyContext.Companies.Remove(company);
    companyContext.SaveChanges();
    Console.WriteLine("Firma została usunięta.");
}
```

main programu Program.cs:

```
private static void Main()
{
    var companyContext = new CompanyContext();

    Company? company = null;
    var correctAnswer = false;
    String? choice;
    do
    {
        Console.WriteLine("Dodać nową firmę? (Tak/Nie)");
        choice = Console.ReadLine();
        switch (choice)
        {
            case "Tak":
                company = CreateCompany();
                if (company == null)
                {
                    return;
                }

                companyContext.Companies.Add(company);
            }
        }
    } while (!correctAnswer);
}
```

```

        companyContext.SaveChanges();
        correctAnswer = true;
        break;
    case "Nie":
        correctAnswer = true;
        companyContext.SaveChanges();
        break;
    }
} while (!correctAnswer || choice == "Tak");

ShowAllCompanies(companyContext);
ShowAllSuppliers(companyContext);
ShowAllCustomers(companyContext);
}

```

Po upewnieniu się, że wszystko działa poprawnie, dodajemy przykładowego Supliera, a następnie Customera

```

Podaj typ nowej firmy (Supplier/Customer): Supplier
Podaj nazwę nowej firmy: LG
Podaj ulicę nowej firmy: Kawiorzy
Podaj miasto nowej firmy: Kraków
Podaj kod pocztowy nowej firmy: 23-233
Podaj numer konta bankowego nowej firmy: 23465657253
Dodać nową firmę? (Tak/Nie)

```

Widok bazy danych po wykonaniu:

```
select * from Customers
```

CompanyID	CompanyName	Street	City	ZipCode	Discriminator	CustomerID	Discount	SupplierID	BankAccountNumber
1	LG	Kawiorzy	Kraków	23-233	Supplier	<null>	<null>	0	23465657253

Następnie dodajemy tą samą metodą wiele Supplierów oraz Customerów:

```

Podaj typ nowej firmy (Supplier/Customer): Customer
Podaj nazwę nowej firmy: Kowalski2
Podaj ulicę nowej firmy: Zielona
Podaj miasto nowej firmy: Warszawa
Podaj kod pocztowy nowej firmy: 33-444
Podaj zniżkę nowej firmy: 33
Dodać nową firmę? (Tak/Nie)

```

...

```

Lista wszystkich firm:
[1] LG (dostawca)
[2] Kowalski (klient)
[3] Samsung (dostawca)
[4] Kowalski2 (klient)
Lista dostawców:
[1] LG (dostawca)
[3] Samsung (dostawca)
Lista klientów:
[2] Kowalski (klient)
[4] Kowalski2 (klient)

```


Widok bazy danych:

	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	CustomerID	Discount	SupplierID	BankAccountNumber
1	1	LG	Kawior	Kraków	23-233	Supplier	<null>	<null>	0	23465657253
2	2	Kowalski	Mazowiecka	Wrocław	22-222	Customer	0	23	<null>	<null>
3	3	Samsung	Kwiatowa	Lublin	11-111	Supplier	<null>	<null>	0	32364646425
4	4	Kowalski2	Zielona	Warszawa	33-444	Customer	0	33	<null>	<null>

Przykładowe działanie FindCompany:

```
Lista wszystkich firm:
[1] LG (dostawca)
[2] Kowalski (klient)

Lista dostawców:
[1] LG (dostawca)

Lista klientów:
[2] Kowalski (klient)

Wyszukać firmę? (Tak/Nie)
Tak
Podaj ID firmy do wyszukiwania: 2
[2] Kowalski (klient)

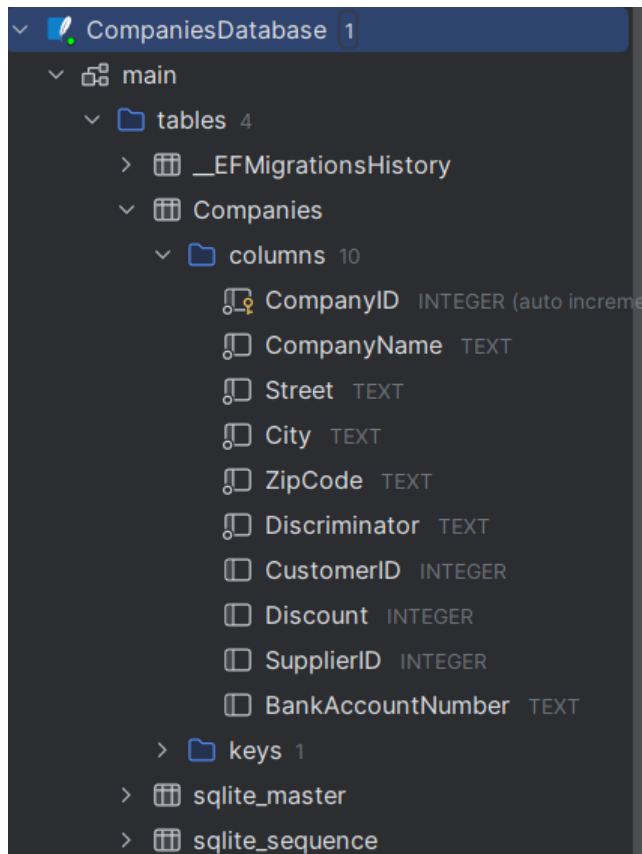
Wyszukać firmę? (Tak/Nie)
Tak
Podaj ID firmy do wyszukiwania: 1
[1] LG (dostawca)
```

Wynik wyszukiwania wszystkich Supplierów:

	CompanyID	CompanyName	Street	City	ZipCode	Discriminator	CustomerID	Discount
1	1	LG	Krakowska	Kraków	23-333	Supplier	<null>	<null>
2	3	Samsung	Kwiatowa	Lublin	11-111	Supplier	<null>	<null>

Schemat bazy danych składa się tylko z tabeli Companies:

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



Jak widać jest to zwykła tabela przechowująca różne typy, rozróżniająca je za pomocą pola Discriminator.

Zadanie f - Table-Per-Type:

Implementacja tego zadania wygląda identycznie. Zmianie uległy jedynie dwie klasy: Customer oraz Supplier. Do nich dodaliśmy taką adnotację przed nazwą klasy: [Table("nazwa_klasy")]

Supplier.cs:

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

namespace zad5;

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

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

Customer.cs:

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

namespace zad5;

[Table("Customers")]
internal class Customer : Company
{
    public int CustomerID { get; set; }
```

```
public int Discount { get; set; } // In %

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

Następnie możemy wykonać wszystkie operacje tak jak w poprzednim zadaniu:

Dodajemy Suppliera oraz Customera:

```
Dodać nową firmę? (Tak/Nie)
Tak
Podaj typ nowej firmy (Supplier/Customer): Supplier
Podaj nazwę nowej firmy: Apple
Podaj ulicę nowej firmy: Krakowska
Podaj miasto nowej firmy: Kraków
Podaj kod pocztowy nowej firmy: 23-333
Podaj numer konta bankowego nowej firmy: 5364564564
Dodać nową firmę? (Tak/Nie)
Tak
Podaj typ nowej firmy (Supplier/Customer): Customer
Podaj nazwę nowej firmy: Kowalski3
Podaj ulicę nowej firmy: Kawiory
Podaj miasto nowej firmy: Kraków
Podaj kod pocztowy nowej firmy: 33-444
Podaj zniżkę nowej firmy: 33
Dodać nową firmę? (Tak/Nie)
Nie
Lista wszystkich firm:
[1] Apple (dostawca)
[2] Kowalski3 (klient)
Lista dostawców:
[1] Apple (dostawca)
Lista klientów:
[2] Kowalski3 (klient)
```

Wynik bazy danych po wykonaniu 'select * from Companies':

Output main.Companies

	CompanyID	CompanyName	Street	City	ZipCode
1	1	Apple	Krakowska	Kraków	23-333
2	2	Kowalski3	Kawiory	Kraków	33-444

Wynik bazy danych po wykonaniu 'select * from Customers':

	CompanyID	CustomerID	Discount
1	2	0	33

Wynik bazy danych po wykonaniu 'select * from Supplier':

CompanyID	SupplierID	BankAccountNumber
1	0	5364564564

Wynik konsolowego wyszukiwania:

```
Lista wszystkich firm:
[1] Apple (dostawca)
[2] Kowalski3 (klient)
[3] Samsung (dostawca)

Lista dostawców:
[1] Apple (dostawca)
[3] Samsung (dostawca)

Lista klientów:
[2] Kowalski3 (klient)

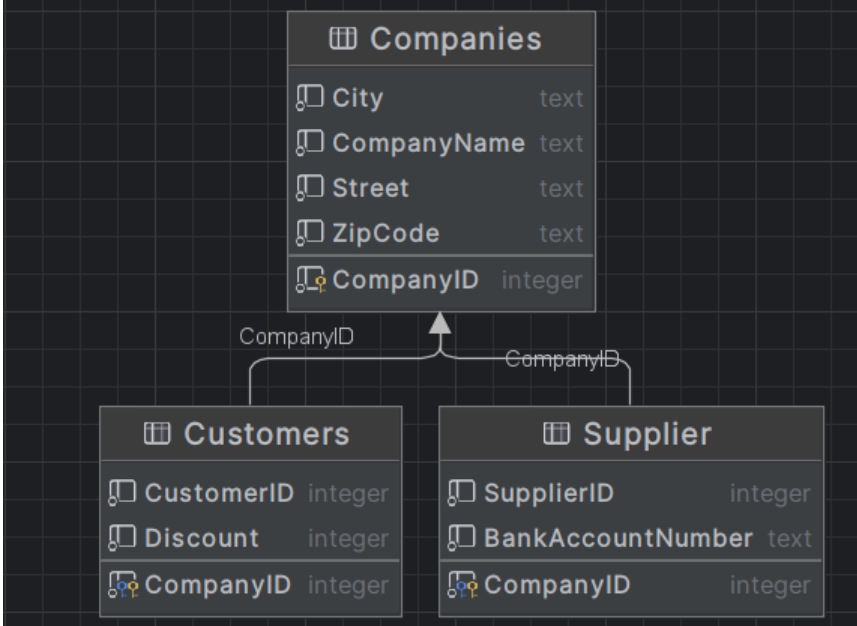
Wyszukać firmę? (Tak/Nie)
Tak
Podaj ID firmy do wyszukiwania: 3
[3] Samsung (dostawca)
```

Szkielet bazy danych:

CompaniesDatabase 1

- main
 - tables 6
 - > _EFMigrationsHistory
 - > Companies
 - > Customers
 - > sqlite_master
 - > sqlite_sequence
 - > Supplier

Schemat bazy danych:



Zadanie g - Podsumowanie:

Opis strategii dziedziczenia: Table-Per-Hierarchy:

- Tworzona jest jedna duża tabela przechowująca wszystkie wspólne pola klas dziedziczących po niej jak i pola charakterystyczne dla poszczególnych typów dziedziczących
- Jeśli dany typ nie ma takiego pola, w tym miejscu wpisywane jest null. Pole discriminator wskazuje na typ klasy dziedziczącej do rozróżnienia obiektów

Zalety:

- Zmniejszenie wykonywania operacji join w porównaniu do operacji na tabelach wykorzystujących Table-Per-Type co może prowadzić do zwiększenia wydajności zapytań

Wady:

- Marnowanie wolnej pamięci na dużą ilość pustych komórek - null (szczególnie zauważalne jeśli kilka klas dziedziczy z jednej głównej klasy)
- Zwiększenie liczby komórek o wartości null (charakterystycznych dla typów dziedziczących) prowadzi do tego, że baza danych staje się nieczytelny

Opis strategii dziedziczenia: Table-Per-Type:

- Tworzone są kilka tabel zarówno dla klas dziedziczących, jak i dla klasy po której dziedziczą te klasy. Każda klasa dziedzicząca zawiera indywidualne atrybuty, natomiast klasa bazowa zawiera wyłącznie pola wspólne dla wszystkich klas dziedziczących
- Tabele klas dziedziczących są łączone z tabelą klasy, z której dziedziczą, przy pomocy relacji 1 do 1

Zalety:

- Nie przechowujemy już pustych pól - null, dzięki czemu wszystkie nasze wartości stanowią dane, które zostały wstępnie wprowadzone do bazy
- W przypadku wielu klas dziedziczących pozbywając się dużej ilości nulli zwiększamy czytelność bazy danych

Wady:

- Konieczne jest wykonywanie wielu operacji join (joinujemy za każdym razem klasę bazową z klasą dziedziczącą jeśli chcemy wyciągnąć bardziej szczegółowe dane o konkretnym typie klasy)