
Imiona i nazwiska autorów:

Antoni Dulewicz, Marcin Serafin, Wojciech Wietrzny

Część samodzielna: **a. Dodanie dostawcy i ustawienie dostawcy produktu na niego:**

Supplier:

```
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;
    }
}
```

Product:

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

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

ProdContext:

```
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:

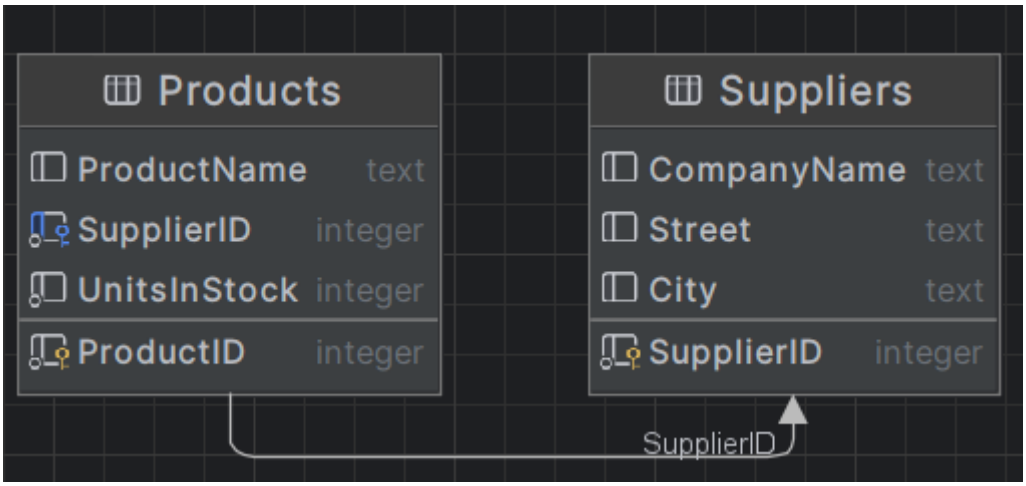
```
ProdContext prodContext = new ProdContext();

Product product = new Product { ProductName = "Flamaster", UnitsInStock = 10 };

Supplier supplier = new Supplier { CompanyName = "MyCompany", City = "Cracow",
Street = "Kawiory" };

product.Supplier = supplier;

prodContext.Suppliers.Add(supplier);
prodContext.Products.Add(product);
prodContext.SaveChanges();
```

Suupliers:

	SupplierID	City	CompanyName	Street
1	1	Cracow	MyCompany	Kawiory

Products:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Flamaster	1	10

Relacja: 

b. Odwrocenie relacji:

Product:

```
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:

```
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:

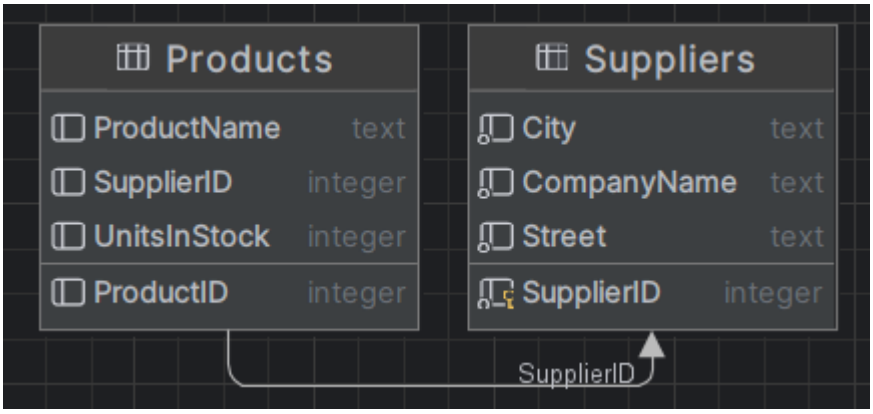
```
ProdContext prodContext = new ProdContext();

Product product = new Product { ProductName = "Zeszyt", UnitsInStock = 10 };

Supplier supplier = new Supplier { CompanyName = "OtherCompany", City = "Warsaw",
Street = "Zielona" };

supplier.Products.Add(product);

prodContext.Suppliers.Add(supplier);
prodContext.Products.Add(product);
prodContext.SaveChanges();
```

Suppliers:

	SupplierID	City	CompanyName	Street
1	1	Cracow	MyCompany	Kawiory
2	2	Warsaw	OtherCompany	Zielona

Products:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Flamaster	1	10
2	2	Zeszyt	2	10

Relacja: 

c. Relacja dwustronna:

Do Product spowrotem dodajemy:

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

Supplier zostawiamy tak jak jest

Program:

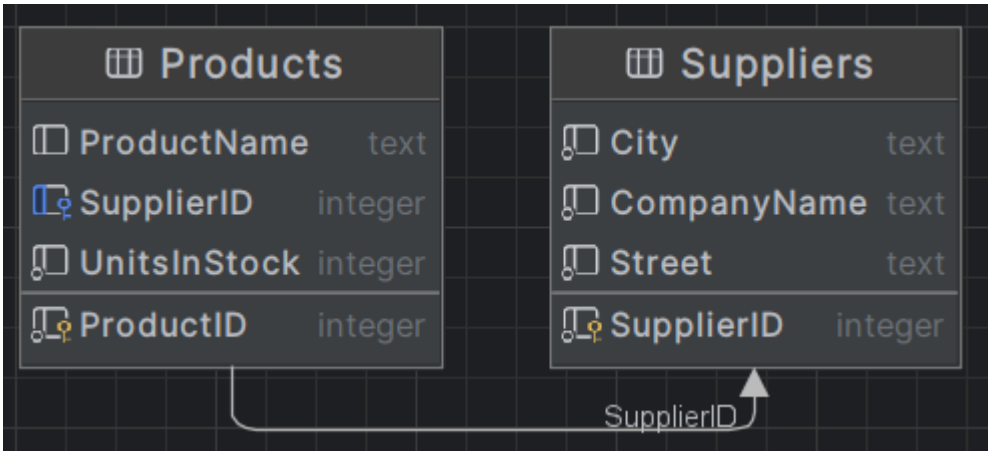
```
ProdContext prodContext = new ProdContext();

Product product1 = new Product { ProductName = "Olowek", UnitsInStock = 15 };
Product product2 = new Product { ProductName = "Dlugopis", UnitsInStock = 20 };

Supplier supplier = new Supplier { CompanyName = "NewCompany", City = "Gdansk",
Street = "Brudna" };

supplier.Products.Add(product1);
supplier.Products.Add(product2);
product1.Supplier = supplier;
product2.Supplier = supplier;

prodContext.Suppliers.Add(supplier);
prodContext.Products.Add(product1);
prodContext.Products.Add(product2);

prodContext.SaveChanges();
```

Suppliers:

	SupplierID	City	CompanyName	Street
1	1	Cracow	MyCompany	Kawiory
2	2	Warsaw	OtherCompany	Zielona
3	3	Gdansk	NewCompany	Brudna

Products:

	ProductID	ProductName	SupplierID	UnitsInStock
1	1	Flamaster	1	10
2	2	Zeszyt	2	10
3	3	Olowek	3	15
4	4	Dlugopis	3	20

Relacja: 

d. Invoice:

Klasa pomocnicza, ProductInvoice:

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

public class ProductInvoice
{
    [Key]
    public int ProductInvoiceID { get; set; }
    public int Quantity { get; set; }

    [ForeignKey("ProductID")]
    public Product? Product { get; set; } = null;

    [ForeignKey("InvoiceNumber")]
    public Invoice? Invoice { get; set; } = null;

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

Invoice:

```
using System.ComponentModel.DataAnnotations;

public class Invoice
{
    [Key]
    public int InvoiceNumber { get; set; }
    public ICollection<ProductInvoice> ProductInvoices { get; set; } = new
List<ProductInvoice>();

    public override string ToString(){
        string result = "-------------------\n";
        result += "Faktura nr " + InvoiceNumber + ":\n";
        int total = 0;
        result += "-------------------\n";
        foreach (var prodInvoice in ProductInvoices)
        {
            result += prodInvoice.Product.ProductName + ": " +
prodInvoice.Quantity + " szt.\n";
            total += prodInvoice.Quantity;
        }
        result += "-------------------\n";
        result += "Total quantity: " + total + " szt.\n";
        result += "-------------------\n";

        return result;
    }

}
```

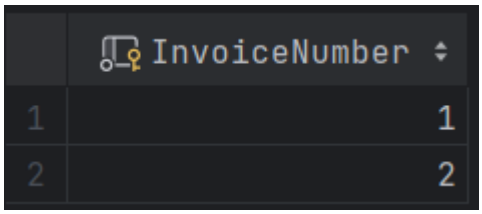
Program

```
ProdContext prodContext = new ProdContext();

Invoice invoice = new Invoice {};


var product1 = prodContext.Products.FirstOrDefault(p => p.ProductID == 1);
var productInvoice = new ProductInvoice { Product = product1, Quantity = 5 };
product1.ProductInvoices.Add(productInvoice);
invoice.ProductInvoices.Add(productInvoice);

prodContext.ProductInvoice.Add(productInvoice);

var product3 = prodContext.Products.FirstOrDefault(p => p.ProductID == 3);
productInvoice = new ProductInvoice { Product = product3, Quantity = 5 };
product3.ProductInvoices.Add(productInvoice);
invoice.ProductInvoices.Add(productInvoice);
```

```
prodContext.ProductInvoice.Add(productInvoice);

var product4 = prodContext.Products.FirstOrDefault(p => p.ProductID == 4);
productInvoice = new ProductInvoice { Product = product4, Quantity = 5 };
product4.ProductInvoices.Add(productInvoice);
invoice.ProductInvoices.Add(productInvoice);

prodContext.ProductInvoice.Add(productInvoice);

prodContext.Invoices.Add(invoice);


var newInvoice = new Invoice();

var product5 = prodContext.Products.FirstOrDefault(p => p.ProductID == 4);
var productInvoice = new ProductInvoice { Product = product5, Quantity = 15 };

product5.ProductInvoices.Add(productInvoice);
newInvoice.ProductInvoices.Add(productInvoice);

prodContext.ProductInvoices.Add(productInvoice);

var product6 = prodContext.Products.FirstOrDefault(p => p.ProductID == 2);
productInvoice = new ProductInvoice { Product = product6, Quantity = 5 };

product6.ProductInvoices.Add(productInvoice);
newInvoice.ProductInvoices.Add(productInvoice);

prodContext.ProductInvoices.Add(productInvoice);

prodContext.Invoices.Add(newInvoice);

prodContext.SaveChanges();
```

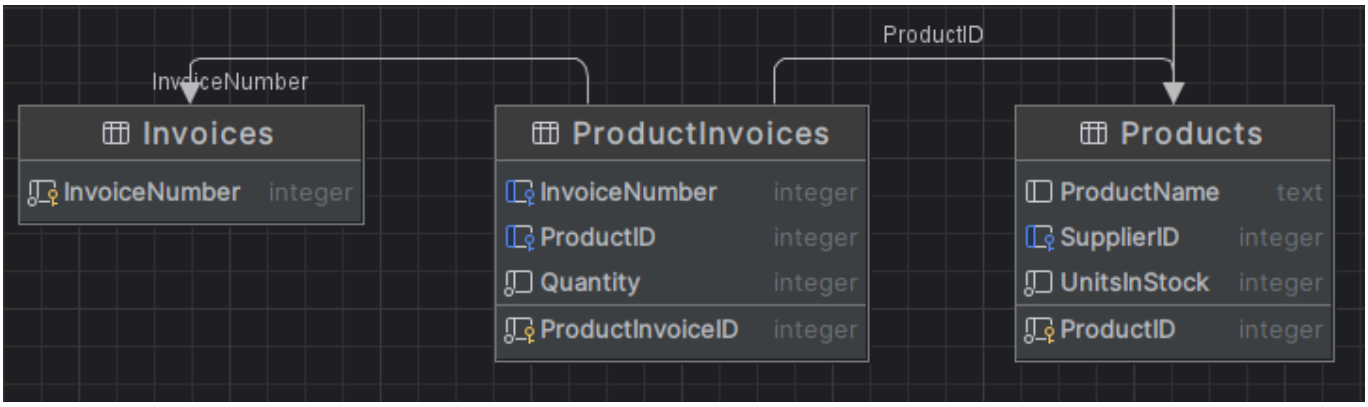
ProductInvoice:

	ProductInvoiceID	InvoiceNumber	ProductID	Quantity
1	2	1	1	10
2	3	1	3	5
3	4	1	4	5
4	5	2	4	15
5	6	2	2	5

Invoice: 

	InvoiceNumber
1	1
2	2

Relacja:



Produkty sprzedane w ramach danej faktury/transakcji:

```
using System.Linq;
using Microsoft.EntityFrameworkCore;
ProdContext prodContext = new ProdContext();

var myInvoice = prodContext.Invoices.Where(inv => inv.InvoiceNumber ==
1).FirstOrDefault();

var prodPerInvoice = prodContext.Invoices.Where(inv =>inv.InvoiceNumber == 1)
    .Include(inv => inv.ProductInvoices)
    .ThenInclude(pi => pi.Product)
    .ToList();

Console.WriteLine(myInvoice);

prodContext.SaveChanges();
```

```
-------------------
Faktura nr 1:
-------------------
Flamaster: 10 szt.
Olowek: 5 szt.
Dlugopis: 5 szt.
-------------------
Total quantity: 20 szt.
-------------------

PS C:\Users\anton\OneDrive\Pulpit\entity\ADULEWICZEFLAB>
```

Faktury w ramach ktory zostal sprzedany dany produkt:

```
using System.Linq;
using Microsoft.EntityFrameworkCore;
ProdContext prodContext = new ProdContext();

var prodToSearch = prodContext.Products
    .Where(p => p.ProductID == 4)
    .Include(p => p.ProductInvoices)
    .ThenInclude(pi => pi.Invoice)
    .FirstOrDefault();


Console.WriteLine(prodToSearch + " -> id: " + prodToSearch.ProductID);
Console.WriteLine("Invoices:");
foreach (var prodInvoice in prodToSearch.ProductInvoices)
{
    Console.WriteLine("Faktura: " + prodInvoice.Invoice.InvoiceNumber + " -> " +
prodInvoice.Quantity + " szt.");

}

prodContext.SaveChanges();
```

```
Dlugopis: 20 szt. -> id: 4
Invoices:
Faktura: 1 -> 5 szt.
Faktura: 2 -> 15 szt.
PS C:\Users\anton\OneDrive\Pulpit\entity\ADULEWICZEFLAB>
```

e. Table-Per-Hierarchy:

Klasa Company:

```
public abstract class Company
{
    public int CompanyId { get; set; }
    public String CompanyName { get; set; } = String.Empty;
    public String City { get; set; } = String.Empty;
    public String Street { get; set; } = String.Empty;
    public String ZipCode { get; set; } = String.Empty;

    public override string ToString()
    {
        return $"Company ({CompanyId}) : {CompanyName} located in {City}";
    }

}
```

Klasa Supplier:

```
public class Supplier : Company
{
    public int SupplierID { get; set; }
    public String bankAccountNumber { get; set; } = String.Empty;

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

Klasa Customer:

```
public class Customer : Company
{
    public int CustomerId { get; set; }
    public double discount { get; set; }

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

CompanyContext:

```
using Microsoft.EntityFrameworkCore;
public 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=MyCompanyDatabase");
    }

}
```

Program:

```
using System;
using System.Linq;

CompanyContext context = new CompanyContext();
```

```
Supplier supplier = new Supplier {CompanyName = "POLMEX", City = "Cracow", Street
= "Mickiewicza", ZipCode = "32-090", bankAccountNumber = "1234567890"};
Customer customer = new Customer {CompanyName = "Microsoft", City = "Warsaw",
Street = "Marszałkowska", ZipCode = "01-100", discount = 0.5};

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

context.SaveChanges();
```

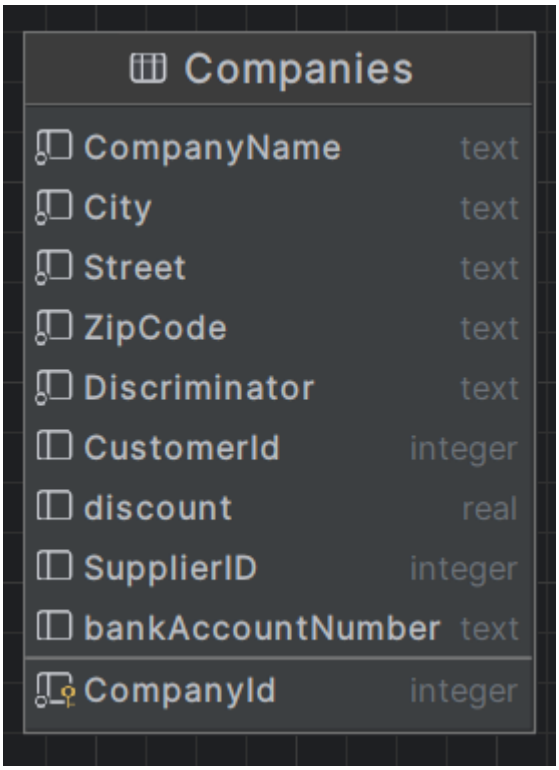


Tabela:

	CompanyId	CompanyName	City	Street	ZipCode	Discriminator	CustomerId	discount	SupplierID	bankAccountNumber
1	1	Microsoft	Warsaw	Marszałkowska	01-100	Customer	0	0.5	<null>	<null>
2	2	POLMEX	Cracow	Mickiewicza	32-090	Supplier	<null>	<null>	0	1234567890

Pobranie firm:

```
using System;

CompanyContext context = new CompanyContext();

var myCustomer = context.Customers.FirstOrDefault();

var mySupplier = context.Suppliers.FirstOrDefault();

Console.WriteLine(myCustomer.ToString());
Console.WriteLine(mySupplier.ToString());

context.SaveChanges();
```

```
Customer: Company (1) : Microsoft located in Warsaw
Supplier: Company (2) : POLMEX located in Cracow
PS C:\Users\anton\OneDrive\Pulpit\entity\COMPEF>
```

f. Table-Per-Type:

Klasa Company:

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

[Table("Companies")]
public abstract class Company
{
    public int CompanyId { get; set; }
    public String CompanyName { get; set; } = String.Empty;
    public String City { get; set; } = String.Empty;
    public String Street { get; set; } = String.Empty;
    public String ZipCode { get; set; } = String.Empty;

    public override string ToString()
    {
        return $"Company ({CompanyId}) : {CompanyName} located in {City}";
    }

}
```

Klasa Supplier:

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

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

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

Klasa Customer

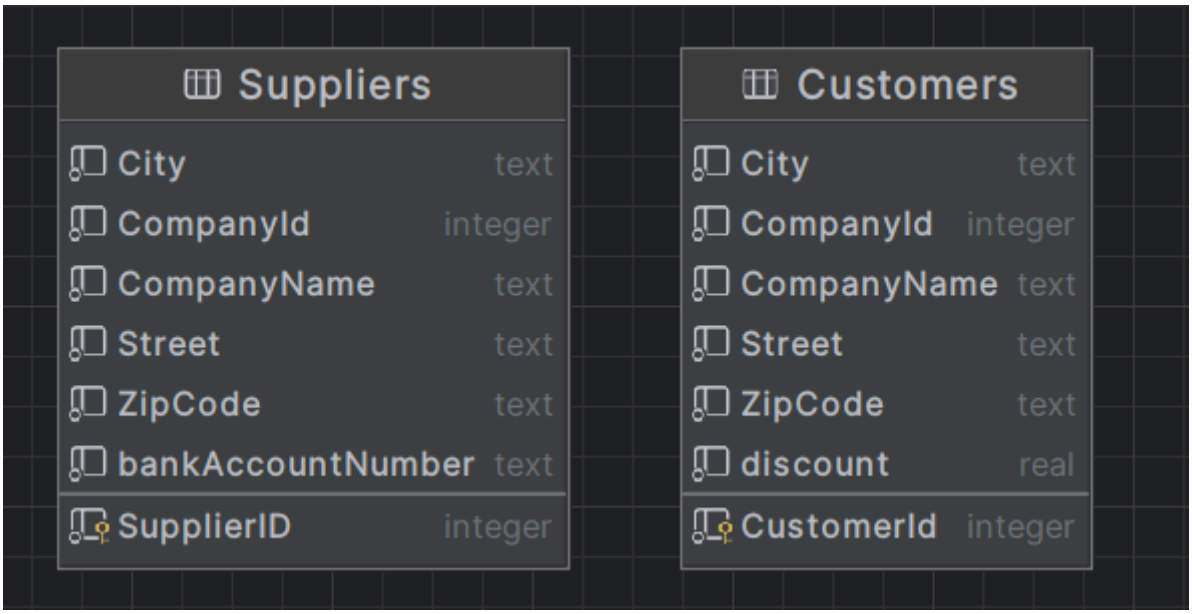
```
using System.ComponentModel.DataAnnotations.Schema;

[Table("Customers")]
public class Customer : Company
{
    public int CustomerId { get; set; }
    public double discount { get; set; }

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

CompanyContext:

```
using Microsoft.EntityFrameworkCore;
public class CompanyContext: DbContext{
    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Customer> Customers { get; set; }

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

}
```

Program:

```
using System;

CompanyContext context = new CompanyContext();

Supplier supplier = new Supplier {CompanyName = "POLMEX", City = "Cracow", Street
= "Mickiewicza", ZipCode = "32-090", bankAccountNumber = "1234567890"};
Customer customer = new Customer {CompanyName = "Microsoft", City = "Warsaw",
Street = "Marszałkowska", ZipCode = "01-100", discount = 0.5};

context.Suppliers.Add(supplier);
context.Customers.Add(customer);

context.SaveChanges();
```

Tabele: 

Pobranie firm:

```
using System;

CompanyContext context = new CompanyContext();

var getSupp = context.Suppliers.Find(1);
var getCust = context.Customers.Find(1);

if(getSupp != null && getCust != null){
    Console.WriteLine(getSupp.ToString());
    Console.WriteLine(getCust.ToString());
}

context.SaveChanges();
```

```
Supplier: Company (0) : POLMEX located in Cracow
Customer: Company (0) : Microsoft located in Warsaw
PS C:\Users\anton\OneDrive\Pulpit\entity\COMPEF>
```

Jak widać w przypadku TPT indeksy obu firm są takie same co wynik z posiadami dwie tabele równolegle dziedziczące po Company.

g. TPH vs TPT: W przypadku dziedziczenia per hirerarchy otrzymujemy jedną wspólną tabelę do której wstawiamy i przechowujemy zarówno obiekty Supplier jak i Customer. W przypadku TPT otrzymujemy tyle tabel ile różnych obiektów dziedziczących po Company - w tym przypadku 2. Oba podejścia mają swoje plusy i minusy.

W przypadku **TPH**:

+ Mniejsza ilość tabel + Mniejsza ilość zapytań - W przypadku dużych różnic między obiektami dziedziczącymi po bazowej klasie, tabela może być bardzo rozbudowana i zawierać wiele pustych kolumn - Mniejsza czytelność

W przypadku **TPT**:

+ Większa czytelność + Brak pustych kolumn + Większa kontrola nad strukturą bazy danych - Większa ilość tabel - Większa ilość zapytań