BD-2021: EntityFramework

Vadim Karpinskiy

Zadanie 1: Wprowadzenie do EntityFramework.Core

Stworzyłem projekt o nazwie VKEntityFramework. Następnie dodałem do projektu dwie klasy:

- Product – zawiera pola ProductID, ProductName oraz UnitsOnStock

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

- ProductContext – ta klasa będzie zarządzać obiektami klasy Product: zapisywać i odczytywać je z bazy danych itd. Dana klasa dziedziczy po DbContext.

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

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

Następnie zmieniłem zawartość klasy Program.cs: pytamy użytkownika o nazwę produktu, który chcemy dodać do naszej bazy, a następnie wypisujemy wszystkie zarejestrowane w bazie produkty:

```
namespace VKEntityFramework
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter name of the product: ");
            string prodName = Console.ReadLine();

            Console.WriteLine("Currently registered products in the database: ");
            ProductContext productContext = new ProductContext();
            Product product = new Product { ProductName = prodName };
            productContext.Products.Add(product);
            productContext.SaveChanges();

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

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

Wynik działania programu:

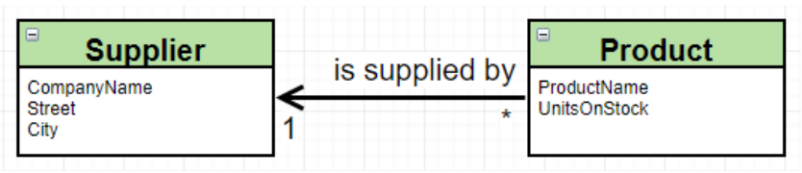
```
Enter name of the product:
Correction fluid
Currently registered products in the database:
Flamaster
Flamaster
Pencil
Highlighter
Eraser
Pen
Correction fluid
```

Diagram z Datagrip:

	ProductID	ProductName	UnitsOnStock
1	1	Flamaster	0
2	2	Flamaster	0
3	3	Pencil	0
4	4	Highlighter	0
5	5	Eraser	0
6	6	Pen	0
7	7	Correction fluid	0
8	8	Paint	0

Zadanie 2

1. Musimy wprowadzić pojęcie Dostawcy oraz zaprezentować następującą relację:



Powyższą relację możemy określić jako “Wiele produktów są dostarczane przez jednego dostawcę”

Na początku tworzymy klasę Supplier:

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

Następnie modyfikujemy klasę Product – dodajemy tam nowe pole Supplier Supplier oraz “SupplierID”, które będzie spełniało rolę klucza obcego:

```
namespace VKEntityFramework
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
        public int SupplierID { get; set; }
        public Supplier Supplier { get; set; }
    }
}
```

Dodajemy do klasy ProductContext kolekcję obiektów klasy Supplier:

```
namespace VKEntityFramework
{
    public class ProductContext: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=Products");
        }
    }
}
```

Dla każdego produktu, który już istnieje w bazie, musimy teraz wprowadzić dostawcę. Robimy to modyfikując funkcję Main:

```
{
    Console.WriteLine("Enter name of the product: ");
    string prodName = Console.ReadLine();
    Product product = new Product { ProductName = prodName };

    Console.WriteLine("Enter company name of the supplier: ");
    string compName = Console.ReadLine();

    ProductContext productContext = new ProductContext();
    Supplier supplier = new Supplier { CompanyName = compName };
    productContext.Suppliers.Add(supplier);
    productContext.SaveChanges();

    var supp1 = (from prod in productContext.Suppliers select prod).FirstOrDefault();

    product.SupplierID = supp1.SupplierID;
    product.Supplier = supp1;
    productContext.Products.Add(product);
    productContext.SaveChanges();

    Console.WriteLine("Currently registered products with suppliers: ");

    foreach (Product item in productContext.Products)
    {
        Console.WriteLine(item.ProductName + " " + item.Supplier.CompanyName);
    }
}
```

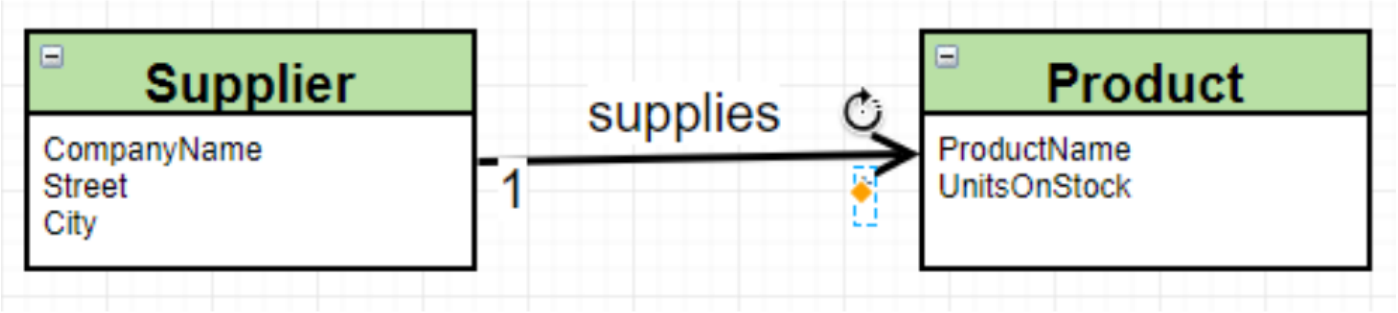
Wynik działania programu:

```
Enter name of the product:
Cap
Enter company name of the supplier:
Criag
Currently registered products with suppliers:
Pencilcase Criag
Pen Criag
Pen Criag
Eraser Criag
Corrector Criag
Sharp Criag
Copybook Criag
Book Criag
Highlighter Criag
Fall Criag
Cap Criag
```

```
sqlite> select * from Products;
1|Pencilcase|1|0
2|Pen|1|0
3|Pen|1|0
4|Eraser|1|0
5|Corrector|1|0
6|Sharp|1|0
```

Czyli relacja jest spełniona: wiele produktów są dostarczane przez pojedynczego dostawcę

2. Następnie musimy odwrócić naszą relację w następujący sposób:



Przywracamy zawartość klasy Product do pierwotnej, a do klasy Supplier dodajemy kolekcję obiektów Product:

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

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

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

    }
}
```

Modyfikujemy również Main:

```
static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();
    Console.WriteLine("Enter company name of the supplier: ");
    string compName = Console.ReadLine();
    Supplier supplier = productContext.Suppliers.FirstOrDefault(s => s.CompanyName == compName);
    if (supplier is null)
    {
        supplier = new Supplier { CompanyName = compName };
        productContext.Suppliers.Add(supplier);
    }

    Console.WriteLine("Enter name of the product: ");
    string prodName = Console.ReadLine();
    Product product = new Product { ProductName = prodName };
    productContext.Products.Add(product);

    supplier.Products.Add(product);

    productContext.SaveChanges();

    Console.WriteLine("Currently registered suppliers and their products: ");

    var data1 = productContext.Suppliers.Include(s => s.Products).ToList();
    foreach (var item in data1)
    {
        Console.WriteLine("Supplier: " + item.CompanyName);
        Console.WriteLine("------------");
        foreach (Product p in item.Products)
        {
            Console.WriteLine(p.ProductName);
        }
        Console.WriteLine("------------");
    }
}
```

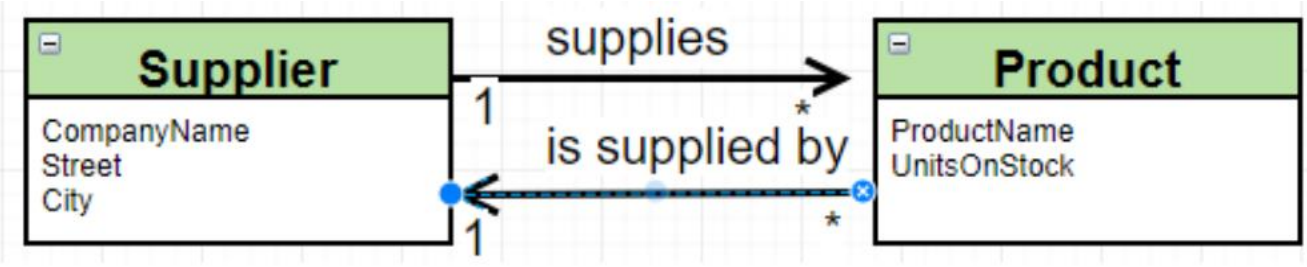
Wyniki działania programu:

```
Enter company name of the supplier:
Valve
Enter name of the product:
Limbo
Currently registered suppliers and their products:
Supplier: RedBarrels
------------
Outlast
Outlast II
Outlast: Whistleblower
------------
Supplier: Valve
------------
Limbo
Alyx
Portal
------------
Supplier: CD Project
------------
Cyberpunk
------------
Supplier: IamTagir
------------
That Level Again
------------
```

```
sqlite> select * from Products;
1|Outlast|1|0
2|Outlast II|1|0
3|Alyx|2|0
4|Cyberpunk|3|0
5|Portal|2|0
6|Outlast: Whistleblower|1|0
7|That Level Again|4|0
8|Limbo|2|0
```

```
sqlite> select * from Suppliers;
1|RedBarrels||
2|Valve||
3|CD Project||
4|IamTagir||
```

3. Teraz zareprezentujemy relację dwustronną:



Aby przedstawić relację takiego typu musimy dodać do klasy Product atrybut Supplier oraz zmodyfikować Main:

```
namespace VKEntityFramework
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int UnitsOnStock { get; set; }
        public Supplier Supplier { get; set; }
    }
}
```

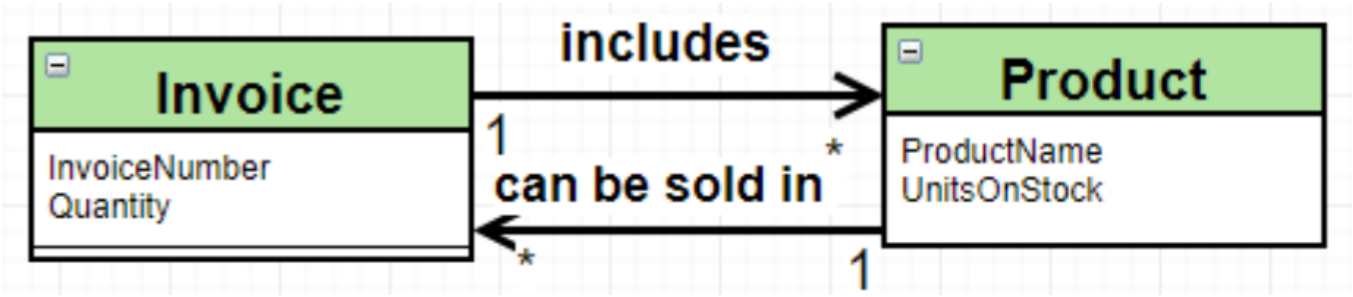
```
{
    ProductContext productContext = new ProductContext();
    Console.WriteLine("Enter company name of the supplier: ");
    string compName = Console.ReadLine();
    Supplier supplier = productContext.Suppliers.FirstOrDefault(s => s.CompanyName == compName);
    if (supplier is null)
    {
        supplier = new Supplier { CompanyName = compName };
        productContext.Suppliers.Add(supplier);
    }

    Console.WriteLine("Enter name of the product: ");
    string prodName = Console.ReadLine();
    Product product = new Product { ProductName = prodName };
    productContext.Products.Add(product);
    product.Supplier = supplier;
    supplier.Products.Add(product);
    productContext.SaveChanges();

    Console.WriteLine("Currently registered suppliers and their products: ");

    var data = productContext.Suppliers.Include(s => s.Products).ToList();
    foreach (var item in data)
    {
        Console.WriteLine("Supplier: " + item.CompanyName);
        Console.WriteLine("------------");
        foreach (Product p in item.Products)
        {
            Console.WriteLine(p.ProductName);
        }
        Console.WriteLine("------------");
    }
    Console.WriteLine("List of suppliers:");
    var data1 = productContext.Products.Include(p => p.Supplier).ToList();
    foreach (var p in data1)
    {
        Console.WriteLine(p.Supplier.CompanyName);
    }
}
```

```
Enter company name of the supplier:
Valve
Enter name of the product:
Cuphead
Currently registered suppliers and their products:
Supplier: RedBarrels
------------
Outlast
Outlast II
Outlast: Whistleblower
------------
Supplier: Valve
------------
Cuphead
Alyx
Portal
Limbo
------------
Supplier: CD Project
------------
Cyberpunk
------------
Supplier: IamTagir
------------
That Level Again
------------
Supplier: Ubisoft
------------
L4D
------------
List of suppliers:
RedBarrels
RedBarrels
Valve
CD Project
Valve
RedBarrels
IamTagir
Valve
Ubisoft
Valve
```

Na koniec zaprezentuję schematy danych tabel:

```
SQLite version 3.35.4 2021-04-02 15:20:15
Enter ".help" for usage hints.
sqlite> .tables
Products              Suppliers             __EFMigrationsHistory
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "SupplierID" INTEGER NULL,
    "UnitsOnStock" INTEGER NOT NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers" ("SupplierID") ON DELETE RESTRICT
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
sqlite> .schema Suppliers
CREATE TABLE IF NOT EXISTS "Suppliers" (
    "SupplierID" INTEGER NOT NULL CONSTRAINT "PK_Suppliers" PRIMARY KEY AUTOINCREMENT,
    "CompanyName" TEXT NULL,
    "Street" TEXT NULL,
    "City" TEXT NULL
);
sqlite>
```

4. Zamodelujemy relację wiele-do-wielu według poniższego schematu:



Na początku tworzymy klasę Invoice i dodajemy do niej 3 pola: InvoiceID, InvoiceNumber oraz Quantity. Dla modelowania powyższej relacji stworzymy pomocniczą tabelę łączącą InvoiceProduct:

```
namespace VKEntityFramework
{
    public class InvoiceProduct
    {
        public int ProductID { get; set; }
        public Product Product { get; set; }

        public int InvoiceID { get; set; }
        public Invoice Invoice { get; set; }
    }
}
```

Teraz zmodyfikujemy zarówno klasę Invoice, jak i Product. Do tych dwóch klas dodajemy kolekcję obiektów typu InvoiceProduct:

```
public class Product
{
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public int UnitsOnStock { get; set; }
    public Supplier Supplier { get; set; }
    public ICollection<InvoiceProduct> InvoiceProducts { get; set; }

    public Product()
    {
        InvoiceProducts = new Collection<InvoiceProduct>();
    }
}
```

```
public class Invoice
{
    public int InvoiceID { get; set; }
    public int InvoiceNumber { get; set; }
    public int Quantity { get; set; }
    public ICollection<InvoiceProduct> InvoiceProducts { get; set; }

    public Invoice()
    {
        InvoiceProducts = new Collection<InvoiceProduct>();
    }
}
```

Musimy również dodać tabele Invoice oraz InvoiceProduct do naszej bazy. Co więcej, musimy zbudować sam moduł relacji pomiędzy nimi. Zrobimy to, odpowiednie modyfikując klasę ProductContext:

```
public class ProductContext:DbContext
{
    public DbSet<Product> Products { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Invoice> Invoices { get; set; }
    public DbSet<InvoiceProduct> InvoiceProducts { get; set; }

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

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<InvoiceProduct>().HasKey(ip => new { ip.ProductID, ip.InvoiceID });
    }
}
```

Na koniec modyfikujemy Main i sprawdzamy wyniki:

```
class Program
{
    static void Main(string[] args)
    {
        // SUPPLIER
        ProductContext productContext = new ProductContext();
        Console.WriteLine("Enter company name of the supplier: ");
        string compName = Console.ReadLine();
        Supplier supplier = productContext.Suppliers.FirstOrDefault(s => s.CompanyName == compName);
        if (supplier is null)
        {
            supplier = new Supplier { CompanyName = compName };
            productContext.Suppliers.Add(supplier);
        }
        // ------------------------------------------------------------- //

        // PRODUCT
        Console.WriteLine("Enter name of the product: ");
        string prodName = Console.ReadLine();
        Product product = new Product { ProductName = prodName };
        productContext.Products.Add(product);
        // ------------------------------------------------------------- //

        // INVOICE
        Console.WriteLine("Enter invoice number: ");
        int invoiceNumber = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Enter quantity of the product: ");
        int quantity = Convert.ToInt32(Console.ReadLine());
        Invoice invoice = productContext.Invoices.FirstOrDefault(i => i.InvoiceNumber == invoiceNumber);
        if (invoice is null)
        {
            invoice = new Invoice { InvoiceNumber = invoiceNumber, Quantity = quantity };
            productContext.Invoices.Add(invoice);
        }
        // ------------------------------------------------------------- //

        // Connecting tables with each other
        product.Supplier = supplier;
        supplier.Products.Add(product);

        InvoiceProduct invoiceProduct = new InvoiceProduct();
        invoiceProduct.Invoice = invoice;
        invoiceProduct.Product = product;

        invoice.InvoiceProducts.Add(invoiceProduct);
        product.InvoiceProducts.Add(invoiceProduct);

        productContext.InvoiceProducts.Add(invoiceProduct);
        productContext.SaveChanges();
        // ------------------------------------------------------------- //

        Console.WriteLine("Products included into invoice number " + invoiceNumber);
```

```
        var products = productContext.InvoiceProducts
            .Include(ip => ip.Product)
            .Where(ip => ip.Invoice.InvoiceNumber == invoiceNumber)
            .Select(ip => ip.Product.ProductName).ToList();

        foreach (var p in products)
        {
            Console.WriteLine(p);
        }

        Console.WriteLine("Invoices that include product with name " + prodName);

        var invoices = productContext.InvoiceProducts
            .Include(ip => ip.Invoice)
            .Where(ip => ip.Product.ProductName == prodName)
            .Select(ip => ip.Invoice.InvoiceNumber).ToList();

        foreach (var i in invoices)
        {
            Console.WriteLine(i);
        }
    }
}
```

Sprawdzamy wyniki:

```
Enter company name of the supplier:
House
Enter name of the product:
Sneakers
Enter invoice number:
1
Enter quantity of the product:
6
Products included into invoice number 1
T-shirt
Sneakers
Jeans
Sneakers
Invoices that include product with name Sneakers
1
3
4
1
```

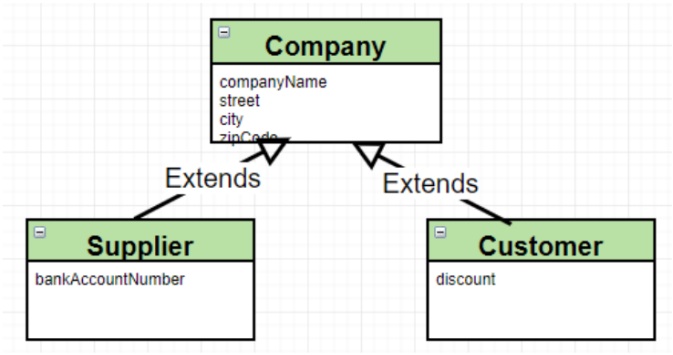
```
sqlite> .tables
InvoiceProducts        Products               __EFMigrationsHistory
Invoices               Suppliers
sqlite> .schema InvoiceProducts
CREATE TABLE IF NOT EXISTS "InvoiceProducts" (
    "ProductID" INTEGER NOT NULL,
    "InvoiceID" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "PK_InvoiceProducts" PRIMARY KEY ("ProductID", "InvoiceID"),
    CONSTRAINT "FK_InvoiceProducts_Invoices_InvoiceID" FOREIGN KEY ("InvoiceID") REFERENCES "Invoices" ("InvoiceID") ON DELETE CASCADE,
    CONSTRAINT "FK_InvoiceProducts_Products_ProductID" FOREIGN KEY ("ProductID") REFERENCES "Products" ("ProductID") ON DELETE CASCADE,
    CONSTRAINT "FK_InvoiceProducts_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers" ("SupplierID") ON DELETE RESTRICT
);
CREATE INDEX "IX_InvoiceProducts_InvoiceID" ON "InvoiceProducts" ("InvoiceID");
CREATE INDEX "IX_InvoiceProducts_SupplierID" ON "InvoiceProducts" ("SupplierID");
sqlite> .schema Invoices
CREATE TABLE IF NOT EXISTS "Invoices" (
    "InvoiceID" INTEGER NOT NULL CONSTRAINT "PK_Invoices" PRIMARY KEY AUTOINCREMENT,
    "InvoiceNumber" INTEGER NOT NULL,
    "Quantity" INTEGER NOT NULL
);
```

```
sqlite> select * from InvoiceProducts;
1|1|
2|1|
3|2|
4|3|
5|4|
6|1|
7|1|
sqlite> select * from Invoices;
1|1|5
2|2|2
3|3|2
4|4|1
```

```
sqlite> select * from Products
   ...> ;
1|T-shirt|1|0
2|Sneakers|2|0
3|Cape|3|0
4|Sneakers|4|0
5|Sneakers|3|0
6|Jeans|2|0
7|Sneakers|1|0
sqlite> select * from Suppliers;
1|House||
2|Cropp||
3|Reserved||
4|H&M||
```

5. Dziedziczenie

Wprowadzimy do modelu poniższą hierarchię dziedziczenia używając strategii Table-Per-Hierarchy:



Tworzymy klasę Company, po której będą dziedziczyć klasy Supplier oraz Customer:

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

Do klas dziedziczących dodajemy ich własne pola oraz zmieniamy ProductContext:

```
namespace VKEntityFramework
{
    public class Supplier : Company
    {
        public long BankAccountNumber { get; set; }
    }
}
```

```
namespace VKEntityFramework
{
    class Customer : Company
    {
        public double Discount { get; set; }
    }
}
```

```
namespace VKEntityFramework
{
    public class ProductContext:DbContext
    {
        public DbSet<Company> Companies { get; set; }

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

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Customer>();
            modelBuilder.Entity<Supplier>();
        }
    }
}
```

Teraz możemy dodać do bazy kilka firm obu rodzajów:

```
class Program
{
    static void Main(string[] args)
    {
        Customer customer = new Customer
        {
            CompanyName = "Mount Massive",
            Street = "Healthy str.",
            City = "Lake Country",
            ZipCode = 13,
            Discount = 3.141592
        };

        Supplier supplier = new Supplier
        {
            CompanyName = "Ericsson",
            Street = "Wall",
            City = "New York",
            ZipCode = 322,
            BankAccountNumber = 50019797131312
        };

        ProductContext productContext = new ProductContext();
        productContext.Companies.Add(supplier);
        productContext.Companies.Add(customer);
        productContext.SaveChanges();
    }
}
```