### Bazy Danych 2

### Laboratorium 4

Entity Framework

Mateusz Łopaciński

### Kod po wprowadzeniu

#### **Zaimplementowane klasy**

#### **Klasa Product**

namespace MateuszLopacinskiEFProducts

{

internal class Product

{

public int ProductID { get; set; }

public string ProductName { get; set; }

public int UnitsOnStock { get; set; }

}

}

#### **Klasa ShopContext**

using Microsoft.EntityFrameworkCore;

namespace MateuszLopacinskiEFProducts

{

internal class ShopContext : DbContext

{

public DbSet<Product> Products { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

optionsBuilder.UseSqlite("Datasource=ProductsDatabase.db");

}

}

}

#### **Klasa Program**

namespace MateuszLopacinskiEFProducts

{

class Program

{

static void Main()

{

Console.WriteLine("Podaj nazwę produktu");

String prodName = Console.ReadLine();

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

ShopContext shopContext = new ShopContext();

Product product = new Product { ProductName = prodName };

shopContext.Products.Add(product);

shopContext.SaveChanges();

var query = from prod in shopContext.Products

select prod.ProductName;

foreach (var pName in query)

{

Console.WriteLine(pName);

}

}

}

}

#### **Przykład działania**

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Wprowadzenie pojęcia Dostawcy

#### **Zaimplementowane klasy**

#### **Klasa Product**

namespace MateuszLopacinskiEFProducts

{

internal class Product

{

public int ProductID { get; set; }

public string ProductName { get; set; }

public int UnitsOnStock { get; set; }

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

public override string ToString()

{

return $"{ProductName} ({UnitsOnStock} szt.)";

}

}

}

#### **Klasa Supplier**

namespace MateuszLopacinskiEFProducts

{

internal class Supplier

{

public int SupplierID { get; set; }

public string CompanyName { get; set; }

public string Street { get; set; }

public string City { get; set; }

public override string ToString()

{

return CompanyName;

}

}

}

#### **Klasa ShopContext**

using Microsoft.EntityFrameworkCore;

namespace MateuszLopacinskiEFProducts

{

internal class ShopContext : DbContext

{

public DbSet<Product> Products { get; set; }

public DbSet<Supplier> Suppliers { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

optionsBuilder.UseSqlite("Datasource=ProductsDatabase.db");

}

}

}

#### **Klasa Program**

namespace MateuszLopacinskiEFProducts

{

class Program

{

static void Main()

{

ShopContext shopContext = new ShopContext();

Product product = createNewProduct();

Supplier? supplier = null;

bool isCorrectChoice = false;

bool createdNewSupplier = false;

do {

Console.WriteLine("Dodać nowego dostawcę? (tak/nie)");

string choice = Console.ReadLine();

switch (choice)

{

case "tak":

isCorrectChoice = true;

supplier = createNewSupplier();

createdNewSupplier = true;

break;

case "nie":

isCorrectChoice = true;

displayAllSuppliers(shopContext);

supplier = findSupplier(shopContext);

break;

}

} while (!isCorrectChoice);

Console.WriteLine("Dodaję dostawcę do produktu...");

product.Supplier = supplier;

Console.WriteLine("Zapisuję dane do bazy...");

if (createdNewSupplier) shopContext.Suppliers.Add(supplier);

shopContext.Products.Add(product);

shopContext.SaveChanges();

}

private static Product createNewProduct()

{

Console.Write("Podaj nazwę produktu\n>>> ");

string prodName = Console.ReadLine();

Console.Write("Podaj liczbę dostępnych sztuk produktu\n>>> ");

int quantity = Int32.Parse(Console.ReadLine());

Console.WriteLine("Tworzę nowy produkt...");

Product product = new Product

{

ProductName = prodName,

UnitsOnStock = quantity

};

Console.WriteLine($"Stworzono produkt: {product}");

return product;

}

private static Supplier createNewSupplier()

{

Console.Write("\nPodaj nazwę dostawcy\n>>> ");

string companyName = Console.ReadLine();

Console.Write("Podaj miasto\n>>> ");

string city = Console.ReadLine();

Console.Write("Podaj ulicę\n>>> ");

string street = Console.ReadLine();

Console.WriteLine("Tworzę nowego dostawcę...");

Supplier supplier = new Supplier

{

CompanyName = companyName,

City = city,

Street = street

};

Console.WriteLine($"Stworzono dostawcę: {supplier}");

return supplier;

}

private static Supplier findSupplier(ShopContext shopContext)

{

Console.Write("Wprowadź id dostawcy, który ma zostać przypisany do nowego produktu\n>>>");

int choice = Int32.Parse(Console.ReadLine());

var query = from sup in shopContext.Suppliers

where sup.SupplierID == choice

select sup;

return query.FirstOrDefault();

}

private static void displayAllSuppliers(ShopContext shopContext)

{

Console.WriteLine("Lista wszystkich dostawców");

foreach (Supplier supplier in shopContext.Suppliers)

{

Console.WriteLine($"[{supplier.SupplierID}] {supplier}");

}

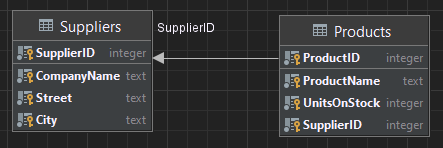
}

}

}

#### **Diagram bazy danych**

Poniżej zamieściłem diagram z programu DataGrip, przedstawiający modelowaną relację.



#### **Przykład działania**

#### **Przykładowe wykonania programu**

* **Z dodawaniem nowego dostawcy**

Obraz zawierający tekst

Opis wygenerowany automatycznie

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

Obraz zawierający tekst

Opis wygenerowany automatycznie

#### **Tabele po kilkukrotnym dodaniu produktów**

* **Produkty**

Obraz zawierający tekst, sprzęt elektroniczny, ekran, zrzut ekranu

Opis wygenerowany automatycznie

* **Dostawcy**

**Obraz zawierający tekst

Opis wygenerowany automatycznie**

### Odwrócenie relacji

#### **Zmienione klasy**

#### **Klasa Product**

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

namespace MateuszLopacinskiEFProducts

{

internal class Product

{

public int ProductID { get; set; }

public string ProductName { get; set; }

public int UnitsOnStock { get; set; }

public override string ToString()

{

return $"{ProductName} ({UnitsOnStock} szt.)";

}

}

}

#### **Klasa Supplier**

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

namespace MateuszLopacinskiEFProducts

{

internal class Supplier

{

public int SupplierID { get; set; }

public string CompanyName { get; set; }

public string Street { get; set; }

public string City { get; set; }

public ICollection<Product> Products { get; set; } = new List<Product>();

public override string ToString()

{

return CompanyName;

}

}

}

#### **Klasa Program**

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

* **Poprzedni kod (fragment)**

static void Main()

{

**…**

Console.WriteLine("Dodaję dostawcę do produktu...");

product.Supplier = supplier;

**…**

}

* **Nowy kod (fragment)**

static void Main()

{

**…**

Console.WriteLine("Dodaję produkt to dostawcy...");

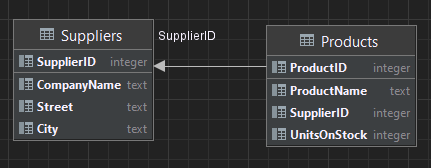
supplier.Products.Add(product);

**…**

}

#### **Diagram bazy danych**

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

****

#### **Przykład działania**

#### **Przykładowe wykonania programu**

* **Z dodawaniem produktu do istniejącego dostawcy**

Obraz zawierający tekst

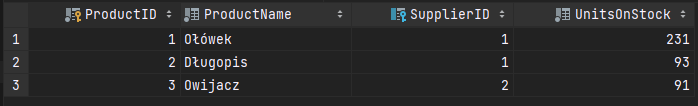
Opis wygenerowany automatycznie

* **Z dodawaniem nowego dostawcy**

#### Obraz zawierający tekst Opis wygenerowany automatycznie

#### **Tabele po dodaniu kilku dostawców i produktów**

* **Produkty**



* **Dostawcy**

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Relacja dwustronna

#### **Zmienione klasy**

#### **Klasa Product**

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

#### **Klasa Supplier**

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

#### **Klasa Program**

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

* **Poprzedni kod (fragment)**

static void Main()

{

**…**

Console.WriteLine("Dodaję produkt to dostawcy...");

supplier.Products.Add(product);

**…**

}

* **Nowy kod (fragment)**

static void Main()

{

**…**

Console.WriteLine("Dodaję produkt to dostawcy...");

product.Supplier = supplier;

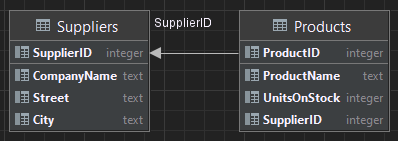
supplier.Products.Add(product);

**…**

}

#### **Diagram bazy danych**

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



#### **Przykład działania**

#### **Przykładowe wykonania programu**

* **Z dodawaniem nowego dostawcy**

Obraz zawierający tekst

Opis wygenerowany automatycznie

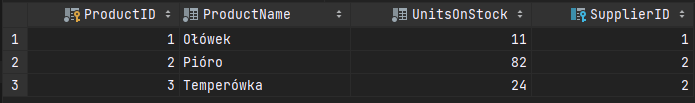
* **Z dodawaniem produktu do istniejącego dostawcy**

Obraz zawierający tekst

Opis wygenerowany automatycznie

#### **Tabele po dodaniu kilku dostawców i produktów**

* **Produkty**



* **Dostawcy**

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Relacja wiele do wielu

#### **Zmienione klasy**

#### **Klasa Product**

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

namespace MateuszLopacinskiEFProducts

{

internal class Product

{

public int ProductID { get; set; }

public string ProductName { get; set; }

public int UnitsInStock { get; set; }

// Navigation properties

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

public override string ToString()

{

return $"{ProductName} ({UnitsInStock} szt. dostępnych)";

}

}

}

#### **Klasa Invoice**

Stworzyłem również poniższą klasę.

using System.ComponentModel.DataAnnotations;

using System.Text;

namespace MateuszLopacinskiEFProducts

{

internal class Invoice

{

[Key]

public int InvoiceNumber { get; set; }

// Navigation properties

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

public override string ToString()

{

StringBuilder sn = new($"Invoice {InvoiceNumber}:");

foreach (InvoiceItem item in InvoiceItems)

{

sn.Append($"\t- {item}");

}

return sn.ToString();

}

}

}

#### **Klasa InvoiceItem**

Klasa pomocnicza, reprezentująca pozycje faktury.

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MateuszLopacinskiEFProducts

{

class InvoiceItem

{

[Key, Column(Order = 0)]

public int InvoiceNumber { get; set; }

[Key, Column(Order = 1)]

public int ProductID { get; set; }

// Navigation properties

public virtual Invoice Invoice { get; set; }

public virtual Product Product { get; set; }

// Additional values

public int Quantity { get; set; }

public override string ToString()

{

return $"{Product} ({Quantity} szt.)";

}

}

}

#### **Klasa ShopContext**

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

using Microsoft.EntityFrameworkCore;

namespace MateuszLopacinskiEFProducts

{

internal class ShopContext : DbContext

{

public DbSet<Product> Products { get; set; }

public DbSet<Invoice> Invoices { get; set; }

public DbSet<InvoiceItem> InvoiceItems { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

optionsBuilder.UseSqlite("Datasource=ProductsDatabase.db");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<InvoiceItem>()

.HasKey(x => new { x.InvoiceNumber, x.ProductID });

}

}

}

#### **Klasa Program**

namespace MateuszLopacinskiEFProducts

{

class Program

{

public const string ADD = "add"; // Add a new product

public const string REMOVE = "remove"; // Remove a product

public const string SELL = "sell"; // Sell products

public const string ALL = "all"; // Display all products

public const string AVAIALBLE = "available"; // Display available products

public const string EXIT = "exit"; // Exit

public static List<string> ALL\_CMDS = new() {

ADD, REMOVE, SELL, ALL, AVAIALBLE, EXIT

};

static void Main()

{

using ShopContext shopContext = new();

bool exited = false;

while (!exited)

{

switch (GetCommand())

{

case ADD:

AddNewProduct(shopContext);

break;

case REMOVE:

RemoveProduct(shopContext);

break;

case SELL:

SellProduct(shopContext);

break;

case ALL:

DisplayAllProducts(shopContext);

break;

case AVAIALBLE:

DisplayAvailableProducts(shopContext);

break;

case EXIT:

exited = true;

Console.WriteLine("To dzisiaj na tyle, dzięki za współpracę");

break;

default:

Console.WriteLine("Polecenie nie zostało rozpoznane, spróbuj jeszcze raz.");

break;

}

Console.WriteLine();

}

}

private static string GetCommand()

{

Console.WriteLine("Napisz, co chcesz zrobić. Lista dostępnych komend:");

DisplayAvailableCommands();

Console.Write(">>> ");

return Console.ReadLine();

}

private static void AddNewProduct(ShopContext shopContext)

{

Product product = CreateNewProduct();

Console.WriteLine("Zapisuję produkt do bazy danych...");

shopContext.Products.Add(product);

shopContext.SaveChanges();

}

private static void SellProduct(ShopContext shopContext)

{

List<InvoiceItem> items = ChooseInvoiceItems(shopContext);

Console.WriteLine("Aktualizuję liczbę dostępnych produktów...");

foreach (InvoiceItem item in items)

{

item.Product.UnitsInStock -= item.Quantity;

Console.WriteLine($"Pozostało {item.Product.UnitsInStock} szt. {item.Product.ProductName}");

}

Invoice invoice = CreateInvoice(items);

shopContext.Invoices.Add(invoice);

shopContext.SaveChanges();

}

private static void RemoveProduct(ShopContext shopContext)

{

Console.WriteLine("Lista wszystkich produktów:");

DisplayAllProducts(shopContext);

Console.Write("Podaj id produktu do usunięcia\n>>> ");

int id = Int32.Parse(Console.ReadLine());

var query = from product in shopContext.Products

where product.ProductID == id

select product;

if (query?.Count() > 0)

{

shopContext.Remove(query.First());

shopContext.SaveChanges();

Console.WriteLine("Produkt został pomyślnie usunięty");

} else

{

Console.WriteLine($"Nie można usunąć produktu. Produkt o id równym {id} nie istnieje");

}

}

private static Invoice CreateInvoice(List<InvoiceItem> items)

{

return new Invoice

{

InvoiceItems = items

};

}

private static Product CreateNewProduct()

{

Console.Write("Podaj nazwę produktu\n>>> ");

string prodName = Console.ReadLine();

Console.Write("Podaj liczbę dostępnych sztuk produktu\n>>> ");

int quantity = Int32.Parse(Console.ReadLine());

Console.WriteLine("Tworzę nowy produkt...");

Product product = new()

{

ProductName = prodName,

UnitsInStock = quantity

};

Console.WriteLine($"Stworzono produkt: {product}");

return product;

}

private static List<InvoiceItem> ChooseInvoiceItems(ShopContext shopContext)

{

Console.WriteLine("Z poniższej listy wybierz produkty, które mają zostać dodane do faktury");

Console.WriteLine("- wprowadź id produktu, a po spacji liczbę sprzedawanych sztuk");

Console.WriteLine("- aby zakończyć wybieranie produktów, naciśniej Enter\n");

DisplayAvailableProducts(shopContext);

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

while (true)

{

Console.Write("\n>>> ");

string input = Console.ReadLine();

if (input == String.Empty)

{

Console.WriteLine("Zakończono wybieranie produktów");

break;

} else

{

string[] splitted = input.Split();

int id = Int32.Parse(splitted[0]);

int quantity = Int32.Parse(splitted[1]);

Product product = shopContext.Products.First(p => p.ProductID == id);

int newQuantity = quantity + (addedItems.ContainsKey(product) ? addedItems[product] : 0);

if (newQuantity > product.UnitsInStock)

{

Console.WriteLine($"Nie można dodać {product.ProductName}. Dostępnych jest tylko {product.UnitsInStock} szt.");

} else

{

Console.WriteLine($"Dodano {quantity} szt. {product.ProductName} do faktury. Razem na fakturze jest {newQuantity} szt.");

if (addedItems.ContainsKey(product)) addedItems.Remove(product);

addedItems.Add(product, newQuantity);

}

}

}

List<InvoiceItem> items = new();

foreach (KeyValuePair<Product, int> item in addedItems)

{

items.Add(new InvoiceItem { Product = item.Key, Quantity = item.Value });

}

return items;

}

private static void DisplayAvailableCommands()

{

foreach (var cmd in Program.ALL\_CMDS)

{

Console.WriteLine($"\t- {cmd},");

}

}

private static void DisplayAllProducts(ShopContext shopContext)

{

var query = from product in shopContext.Products

select product;

foreach (var product in query)

{

Console.WriteLine($"[{product.ProductID}] {product.ProductName} (dostępne: {product.UnitsInStock})");

}

if (query.Count() == 0)

{

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

}

}

private static void DisplayAvailableProducts(ShopContext shopContext)

{

var query = from product in shopContext.Products

where product.UnitsInStock > 0

select product;

foreach (var product in query)

{

Console.WriteLine($"[{product.ProductID}] {product.ProductName} (dostępne: {product.UnitsInStock})");

}

if (query?.Count() == 0)

{

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

}

}

}

}

#### **Diagram bazy danych**

Obraz zawierający tekst

Opis wygenerowany automatycznie

#### **Przykład działania**

#### **Przykładowe wykonania programu**

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

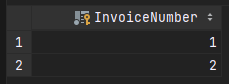
#### **Tabele po dodaniu / usunięciu / sprzedaniu kilku produktów**

* **Products**

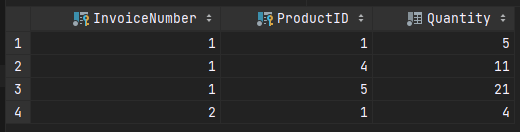
Obraz zawierający tekst

Opis wygenerowany automatycznie

* **Invoices**



* **InvoiceItems**

****

### Dziedziczenie Table-Per-Hierarchy

#### **Wykorzystane klasy**

#### **Klasa Company**

namespace MateuszLopacinskiEFProducts

{

internal abstract class Company

{

public int CompanyID { get; set; }

public string CompanyName { get; set; } = String.Empty;

public string Street { get; set; } = String.Empty;

public string City { get; set; } = String.Empty;

public string ZipCode { get; set; } = String.Empty;

public override string ToString()

{

return $"[{CompanyID}] {CompanyName}";

}

}

}

#### **Klasa CompanyType (jako enum na stringach)**

namespace MateuszLopacinskiEFProducts

{

internal static class CompanyType

{

public const string CUSTOMER = "customer";

public const string SUPPLIER = "supplier";

public static List<string> ALL\_TYPES = new()

{

CUSTOMER,

SUPPLIER

};

}

}

#### **Klasa Supplier**

namespace MateuszLopacinskiEFProducts

{

internal class Supplier : Company

{

public int SupplierID { get; set; }

public string BankAccountNumber { get; set; } = String.Empty;

public override string ToString()

{

return $"{base.ToString()} (dostawca)";

}

}

}

#### **Klasa Customer**

namespace MateuszLopacinskiEFProducts

{

internal class Customer : Company

{

public int CustomerID { get; set; }

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

public override string ToString()

{

return $"{base.ToString()} (klient)";

}

}

}

#### **CompanyContext**

using Microsoft.EntityFrameworkCore;

namespace MateuszLopacinskiEFProducts

{

internal class CompanyContext : DbContext

{

public DbSet<Company>? Companies { get; set; }

public DbSet<Supplier>? Suppliers { get; set; }

public DbSet<Customer>? Customers { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

base.OnConfiguring(optionsBuilder);

optionsBuilder.UseSqlite("Datasource=CompaniesDatabase.db");

}

}

}

#### **Klasa Program**

namespace MateuszLopacinskiEFProducts

{

static class ProgramAction

{

public const string ADD = "add";

public const string DISPLAY = "display";

public static List<string> ACTIONS = new()

{

ADD, DISPLAY

};

}

class Program

{

static void Main()

{

using CompanyContext companyContext = new();

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

switch (action)

{

case ProgramAction.ADD:

AddCompany(companyContext);

break;

case ProgramAction.DISPLAY:

DisplayCompanies(companyContext);

break;

}

}

private static void AddCompany(CompanyContext companyContext)

{

while (true)

{

string type = Choose("Wprowadź typ firmy, którą chcesz dodać", CompanyType.ALL\_TYPES);

string companyName = Input("Podaj nazwę firmy");

string street = Input("Podaj ulicę");

string city = Input("Podaj miasto");

string postalCode = Input("Podaj kod pocztowy (zip)");

switch (type)

{

case CompanyType.CUSTOMER:

companyContext.Companies.Add(CreateCustomer(companyName, street, city, postalCode));

companyContext.SaveChanges();

return;

case CompanyType.SUPPLIER:

companyContext.Companies.Add(CreateSupplier(companyName, street, city, postalCode));

companyContext.SaveChanges();

return;

}

}

}

private static Supplier CreateSupplier(string companyName, string street, string city, string postalCode)

{

string bankAccount = Input("Podaj numer konta bankowego");

return new Supplier

{

CompanyName = companyName,

Street = street,

City = city,

ZipCode = postalCode,

BankAccountNumber = bankAccount

};

}

private static Customer CreateCustomer(string companyName, string street, string city, string postalCode)

{

int discount = int.Parse(Input("Podaj wartość zniżki (%)"));

return new Customer

{

CompanyName = companyName,

Street = street,

City = city,

ZipCode = postalCode,

Discount = discount

};

}

private static void DisplayCompanies(CompanyContext companyContext)

{

List<string> types = new();

types.AddRange(CompanyType.ALL\_TYPES);

types.Add("all");

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

switch (type)

{

case "all":

DisplayAllCompanies(companyContext);

break;

case CompanyType.SUPPLIER:

DisplaySuppliers(companyContext);

break;

case CompanyType.CUSTOMER:

DisplayCustomers(companyContext);

break;

}

}

private static void DisplaySuppliers(CompanyContext companyContext)

{

Console.WriteLine("Lista wszystkich dostawców (firm):");

foreach (Supplier supplier in companyContext.Suppliers)

{

Console.WriteLine(supplier);

}

}

private static void DisplayCustomers(CompanyContext companyContext)

{

Console.WriteLine("Lista wszystkich klientów (firm):");

foreach (Customer customer in companyContext.Customers)

{

Console.WriteLine(customer);

}

}

private static void DisplayAllCompanies(CompanyContext companyContext)

{

Console.WriteLine("Lista wszystkich klientów (firm):");

foreach (Company company in companyContext.Companies)

{

Console.WriteLine(company);

}

}

private static string Input(string text)

{

Console.Write($"{text}\n>>> ");

string? input = Console.ReadLine();

if (input == null) return String.Empty;

return input.Trim();

}

public static string Choose(string text, List<string> choices)

{

Console.WriteLine(text);

Console.WriteLine("Możliwy wybór:");

foreach (string choice in choices) Console.WriteLine($"\t- {choice}");

Console.WriteLine("Aby wyjść, wpisz 'exit'");

while (true)

{

string choice = Input("");

foreach (string possibleChoice in choices)

{

if (choice.Equals(possibleChoice)) return choice;

}

if (choice.Equals("exit"))

{

Console.WriteLine("To dzisiaj na tyle, dzięki za współpracę");

return string.Empty;

}

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

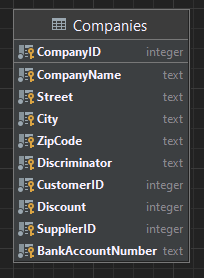
}

}

}

}

#### **Diagram bazy danych**



#### **Przykład działania**

#### **Dodawanie klienta**

Obraz zawierający tekst

Opis wygenerowany automatycznie

#### **Dodawanie dostawcy**

Obraz zawierający tekst

Opis wygenerowany automatycznie

#### **Wypisywanie klientów**

Obraz zawierający tekst

Opis wygenerowany automatycznie

#### **Wypisywanie dostawców**

Obraz zawierający tekst

Opis wygenerowany automatycznie

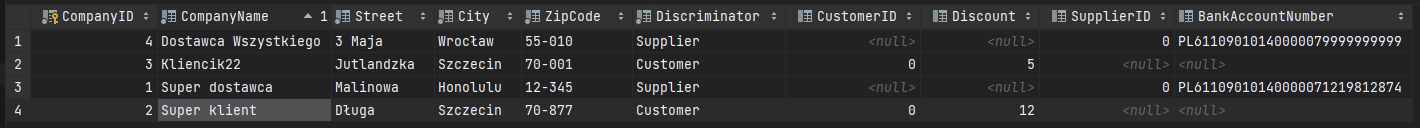
#### **Wypisywanie wszystkich firm**

Obraz zawierający tekst

Opis wygenerowany automatycznie

#### **Utworzona tabela**

* **Companies**



### Dziedziczenie Table-Per-Type

#### **Zmienione klasy**

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

#### **Klasa Customer**

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

using System.ComponentModel.DataAnnotations.Schema;

namespace MateuszLopacinskiEFProducts

{

[Table("Customers")]

internal class Customer : Company

{

public int CompanyID { get; set; }

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

public override string ToString()

{

return $"{base.ToString()} (klient)";

}

}

}

#### **Klasa Supplier**

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

using System.ComponentModel.DataAnnotations.Schema;

namespace MateuszLopacinskiEFProducts

{

[Table("Suppliers")]

internal class Supplier : Company

{

public int CompanyID { get; set; }

public string BankAccountNumber { get; set; } = String.Empty;

public override string ToString()

{

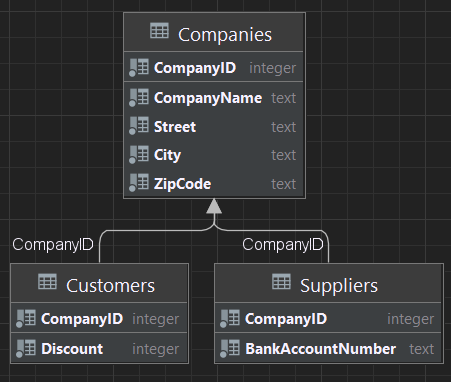
return $"{base.ToString()} (dostawca)";

}

}

}

#### **Diagram bazy danych**



#### **Przykład działania**

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

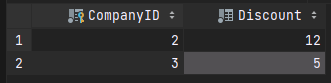
#### **Utworzone tabele**

* **Companies**

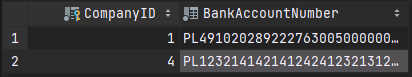
Obraz zawierający tekst

Opis wygenerowany automatycznie

* **Customers**

****

* **Suppliers**



### Porównanie dziedziczenia Table-Per-Hierarchy i Table-Per-Type

#### **Table-Per-Hierarchy**

#### **Charakterystyka**

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

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

#### **Zalety**

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

#### **Wady**

- W przypadku wielu klas dziedziczących z tej samej klasy, jedna tabela nie jest dobrym rozwiązaniem, ponieważ będzie zawierała bardzo dużo wartości **null** (marnowanie miejsca),

- Grupowanie danych, w przypadku wielu klas dziedziczących, zmniejsza przejrzystość schematu bazy danych.

#### **Table-Per-Type**

#### **Charakterystyka**

- Tworzone jest kilka tabel (osobne tabele dla każdej z klas, zarówno tej, z której dziedziczą klasy, jak i klas dziedziczących),

- Tabele klas dziedziczących są łączone z tabelą klasy, z której dziedziczą, przy pomocy relacji **1 do 1**.

#### **Zalety**

- Takie podejście nie wymaga trzymania pustych wartości w tabelach (**null**), dzięki czemu zapisywane są tylko wartości, stanowiące dane,

- W przypadku wielu klas dziedziczących z jednej klasy, takie podejście pozwala na zwiększenie czytelności schematu bazy danych.

#### **Wady**

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