

1 RELACJA PRODUCTS I SUPPLIERS TYPU JEDEN DO WIELU

DODANIE KLASY SUPPLIER

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

1.1 DODANIE POLA REPREZENTUJĄCEGO DOSTAWCĘ W KLASIE PRODUCT

Dodane pole jest nullifikowalne, ponieważ polecenie mówi, że najpierw należy dodać produkt, a potem dostawcę

```
public Supplier? SuppliedBy { get; set; }
```

1.2 DODANIE SUPPLIERS DO KONTEKSTU

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

1.3 MIGRACJA BAZY I JEJ AKTUALIZACJA

```
dotnet ef migrations add AddSuppliedByColumnInProducts
dotnet ef database update
```

W kolejnych zadaniach migracje i aktualizacje będą pomijane

1.4 DODANIE PRODUKTU DO BAZY

```
ProductContext productContext = new ProductContext();

Console.WriteLine("Podaj nazwę produktu");
string prodName = Console.ReadLine();

Product product = new Product { ProductName = prodName };

productContext.Products.Add(product);

productContext.SaveChanges();
```

Tablica Products po wykonaniu:

ProductID	ProductName	SuppliedBySupplierID	UnitsOnStock
1	1 Zeszyt	<null>	0

1.5 DODANIE DOSTAWCY DO SUPPLIERS

```

ProductContext productContext = new ProductContext();

Console.WriteLine("Podaj nazwę dostawcy");
string companyName = Console.ReadLine();
Console.WriteLine("Podaj miasto");
string city = Console.ReadLine();

Supplier supplier = new Supplier { CompanyName = companyName, City =
city};
productContext.Suppliers.Add(supplier);
productContext.SaveChanges();

```

1.6 POWIĄZANIE DOSTAWCY I PRODUKTU

```

ProductContext productContext = new ProductContext();

var supplier = productContext.Suppliers.Single();

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

foreach (var prod in query)
{
    prod.SuppliedBy = supplier;
}

productContext.SaveChanges();

```

Tablica Products po wykonaniu:

ProductID	ProductName	SuppliedBySupplierID	UnitsOnStock
1	1 Zeszyt	1	0

2 ODWRÓCENIE RELACJI

2.1 MODYFIKACJA KLASY PRODUCT

```

namespace GracjanFilipekEFLab
{

```

```

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

```

2.2 MODYFIKACJA KLASY SUPPLIER

```

namespace GracjanFilipekEFLab
{
    public class Supplier
    {
        public int SupplierID { get; set; }
        public string CompanyName { get; set; }
        public string City { get; set; }
        public ICollection<Product> suppliedProducts { get; } = new
HashSet<Product>();
    }
}

```

2.3 STWORZENIE PRODUKTÓW I DOSTAWCY, POWIĄZANIE ICH ZE SOBĄ I DODANIE DO BAZY

```

ProductContext productContext = new ProductContext();

Supplier supplier = new Supplier { City = "London", CompanyName =
"Gregory's Tea Shop"};

ICollection<Product> products = new List<Product>
{
    new Product { ProductName = "Red tea", UnitsOnStock = 4},
    new Product { ProductName = "Green tea", UnitsOnStock = 3 },
    new Product { ProductName = "White tea", UnitsOnStock = 2 }
};

products.ForEach(p => { supplier.suppliedProducts.Add(p); });

productContext.Suppliers.Add(supplier);
productContext.Products.AddRange(products);

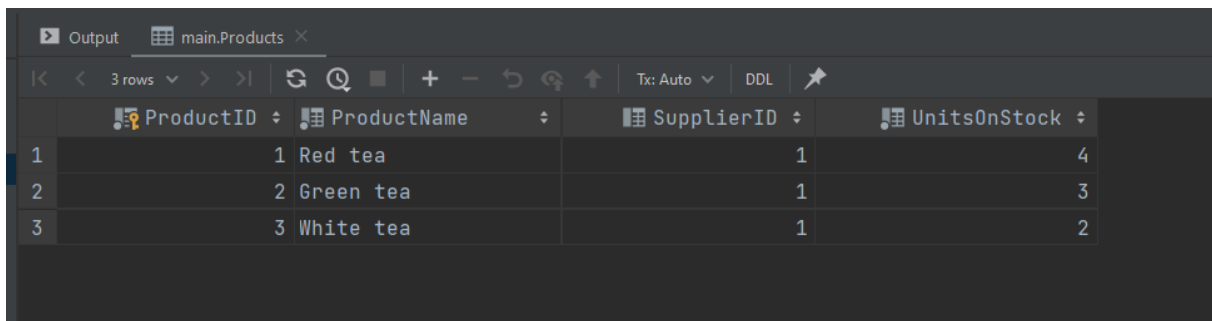
productContext.SaveChanges();

```

Tabela Suppliers po wykonaniu:

main.Suppliers			
SupplierID CompanyName City			
1	Gregory's Tea Shop	London	

Tabela Products po wykonaniu:



	ProductID	ProductName	SupplierID	UnitsOnStock
1	1	Red tea	1	4
2	2	Green tea	1	3
3	3	White tea	1	2

3 RELACJA DWUSTRONNA

3.1 DODANIE POLA SUPPLIEDBY DO KLASY PRODUCT

```
public Supplier SuppliedBy { get; set; }
```

3.2 DODANIE POLA SUPPLIEDPRODUCTS DO KLASY SUPPLIER

```
Public ICollection<Product> suppliedProducts { get; } = new List<Product>();
```

3.3 STWORZENIE DOSTAWCY, PRODUKTÓW I POWIĄZANIE ICH

```
ProductContext productContext = new ProductContext();

Supplier supplier = new Supplier { City = "London", CompanyName =
"Gregory's Tea Shop"};
productContext.Suppliers.Add(supplier);

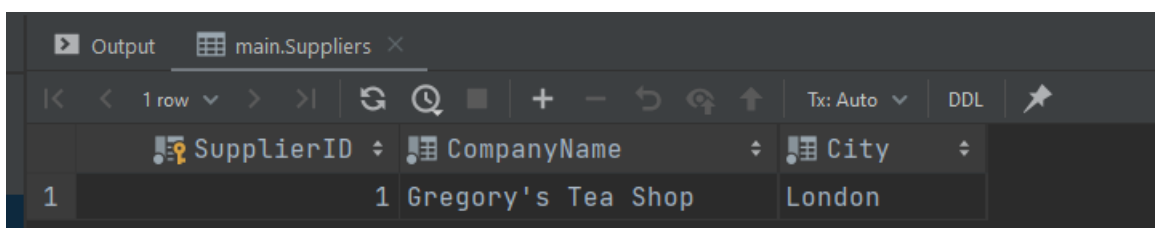
ICollection<Product> products = new List<Product>
{
    new Product { ProductName = "Red tea", UnitsOnStock = 4},
    new Product { ProductName = "Green tea", UnitsOnStock = 3 },
    new Product { ProductName = "White tea", UnitsOnStock = 2 }
};

productContext.Products.AddRange(products);

products.ForEach(p => {
    p.SuppliedBy = supplier;
    supplier.suppliedProducts.Add(p);
});

productContext.SaveChanges();
```

Tabela Suppliers po wykonaniu:



	SupplierID	CompanyName	City
1	1	Gregory's Tea Shop	London

Tabela Products po wykonaniu:

	ProductID	ProductName	SuppliedBySupplierID	UnitsOnStock
1	1	Red tea	1	4
2	2	Green tea	1	3
3	3	White tea	1	2

4

4.1 KOMENTARZ

Na wstępie zaznaczam, że nie zrozumiałem, co miałoby oznaczać „Quantity” w klasie Invoice i dlaczego jest to cecha właśnie klasy Invoice (jak pokazuje diagram), a nie np. relacji „Includes”.

Z tego powodu na potrzeby tego zadania zakładam, że „Quantity” odnosi się do liczby sprzedanych sztuk danego produktu w ramach danej faktury. Z tego też powodu relację many-to-many zamodeluję jako dwie relacje one-to-many (tylko w taki sposób będę w stanie przypisać „Quantity” relacji faktura-produkt, a nie samej fakturze).

4.2 KLASA INVOICE

```
public class Invoice
{
    [Key]
    public int InvoiceNumber { get; set; }
    public ICollection<InvoiceProduct> InvoiceProducts { get; } = new
    HashSet<InvoiceProduct>();
}
```

4.3 KLASA PRODUCT

```
public class Product
{
    public int ProductId { get; set; }
    [Required]
    public string ProductName { get; set; }
    [Required]
    public int UnitsOnStock { get; set; }
    public ICollection<InvoiceProduct> InvoiceProducts { get; } = new
    HashSet<InvoiceProduct>();
}
```

4.4 KLASA INVOICEPRODUCT

```
public class InvoiceProduct
{
    ...
}
```

```

    public int InvoiceId { get; set; }
    public Invoice Invoice { get; set; }

    public int ProductId { get; set; }
    public Product Product { get; set; }

    [Required]
    public int Quantity { get; set; }

    public void Sell()
    {
        if (Product.UnitsOnStock - Quantity < 0)
        {
            throw new ArgumentOutOfRangeException("Cannot sell " + Quantity +
" units, product has only " + Product.UnitsOnStock + " units available");
        }

        Product.UnitsOnStock -= Quantity;
        Product.InvoiceProducts.Add(this);
        Invoice.InvoiceProducts.Add(this);
    }
}

```

4.5 DBCONTEXT

```

public class ProductContext: DbContext
{
    public DbSet<Product> Products { get; set; }
    public DbSet<Invoice> Invoices { get; set; }
    public DbSet<InvoiceProduct> InvoicesProduct { get; set; }

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

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<InvoiceProduct>().HasKey(p => new { p.InvoiceId,
p.ProductId });
    }
}

```

4.6 FUNKCJA MAIN

```

static void Main(string[] args)
{
    ProductContext productContext = new ProductContext();

    ICollection<Invoice> invoices = new HashSet<Invoice>
    {
        new Invoice { },
        new Invoice { },
        new Invoice { },
    };

    ICollection<Product> products = new HashSet<Product>
    {

```

```

        new Product { ProductName = "Red tea", UnitsOnStock = 10},
        new Product { ProductName = "Green tea", UnitsOnStock = 10 },
        new Product { ProductName = "White tea", UnitsOnStock = 10 }
    };

    // connect invoices and products
    ICollection<InvoiceProduct> invoiceProducts = new
    HashSet<InvoiceProduct>
    {
        new InvoiceProduct { Invoice = invoices.ElementAt(0), Product =
        products.ElementAt(0), Quantity = 1},
        new InvoiceProduct { Invoice = invoices.ElementAt(0), Product =
        products.ElementAt(1), Quantity = 2},

        new InvoiceProduct { Invoice = invoices.ElementAt(1), Product =
        products.ElementAt(2), Quantity = 3},

        new InvoiceProduct { Invoice = invoices.ElementAt(2), Product =
        products.ElementAt(0), Quantity = 2},
        new InvoiceProduct { Invoice = invoices.ElementAt(2), Product =
        products.ElementAt(1), Quantity = 1},
        new InvoiceProduct { Invoice = invoices.ElementAt(2), Product =
        products.ElementAt(2), Quantity = 2},
    };

    // finalise each invoice
    foreach (var invProd in invoiceProducts)
    {
        invProd.Sell();
    }

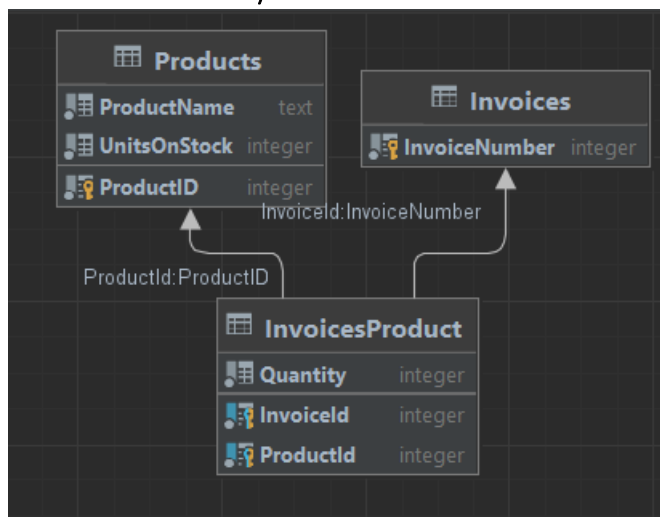
    productContext.Invoices.AddRange(invoices);
    productContext.Products.AddRange(products);

    productContext.SaveChanges();
}

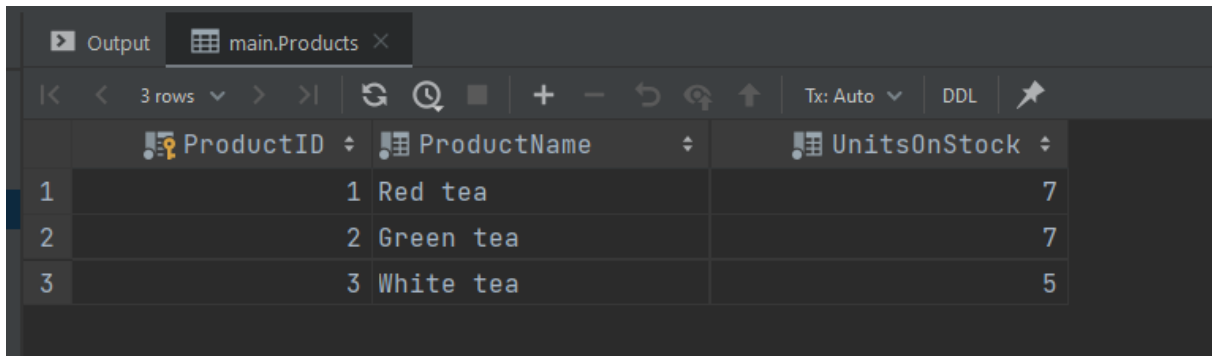
```

4.7 EFEKTY

4.7.1 Schemat bazy

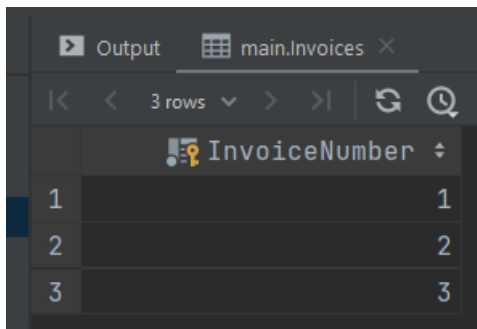


4.7.2 Zawartość tabeli



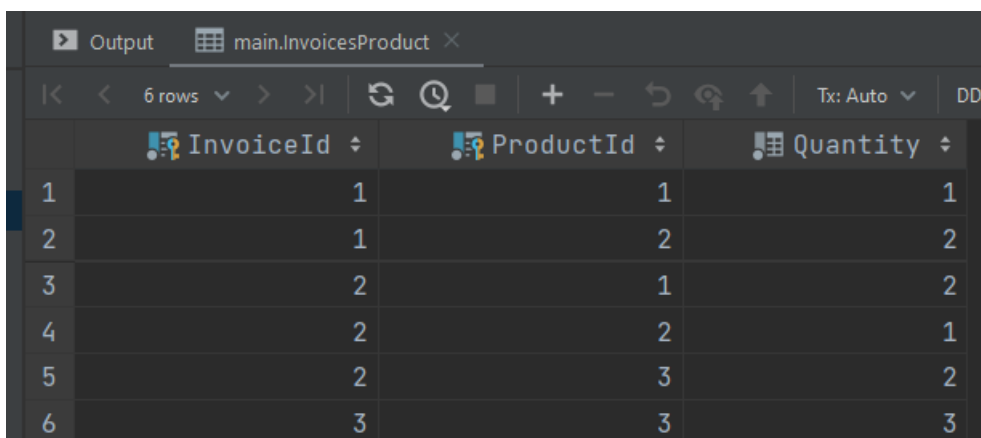
Output main.Products

	ProductID	ProductName	UnitsOnStock
1	1	Red tea	7
2	2	Green tea	7
3	3	White tea	5



Output main.Invoices

	InvoiceNumber
1	1
2	2
3	3



Output main.InvoicesProduct

	InvoiceId	ProductId	Quantity
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1
5	2	3	2
6	3	3	3

5 MAPOWANIE DZIEDZICZENIA: TABLE-PER-HIERARCHY

Polecenia zadań 5 i 6 nie są dla mnie jasne. Jedynymi “strategiami mapowania dziedziczenia”, o jakich znalazłem informacje są Table-Per-Hierarchy, Table-Per-Type oraz Table-Per-Concrete-Type.

Polecenie zadania 5 podpunkt b mówi o trzech różnych strategiach mapowania (zakładam, że mowa o tych wyżej wymienionych), natomiast zadanie 6. każe zamodelować dziedziczenie strategią Table-Per-Type. Z tego powodu połączyłem oba te zadania i pokażę hierarchię zrealizowaną za pomocą tych trzech strategii w trzech kolejnych punktach.

5.1 KONTEKST

```
public DbSet<Company> Companies { get; set; }
public DbSet<Supplier> Suppliers { get; set; }
public DbSet<Customer> Customers { get; set; }
```

5.2 KLASA COMPANY

```
public class Company
{
    [Key]
    public string CompanyName { get; set; }
    [Required]
    public string Street { get; set; }
    [Required]
    public string City { get; set; }
    [Required]
    public string ZipCode { get; set; }

    public override string ToString()
    {
        return JsonSerializer.Serialize(this);
    }
}
```

5.3 KLASA CUSTOMER

```
public class Customer: Company
{
    public float Discount { get; set; } = 0;
```

5.4 KLASA SUPPLIER

```
public class Supplier: Company
{
    [Required]
    [MinLength(8)]
    [MaxLength(20)]
    public string BankAccountNumber { get; set; }
```

5.5 KLASA PROGRAM

Klasa ta jest używana we wszystkich trzech kolejnych zadaniach

```
    static void Main(string[] args)
    {
        //AddToDatabase();

        //QueryFromDatabase();
    }

    static void AddToDatabase()
    {
        ProductContext productContext = new ProductContext();

        ICollection<Customer> customers = new HashSet<Customer>
        {
            new Customer { City = "London", CompanyName = "Customer1",
Discount = 0.2f, Street = "St. James", ZipCode = "00-000"},

```

```

        new Customer { City = "Paris", CompanyName = "Customer2",
Discount = 0.0f, Street = "Trafalgar", ZipCode = "11-222"},
        new Customer { City = "Budapest", CompanyName = "Customer3",
Discount = 0.1f, Street = "Rouge st.", ZipCode = "33-444"},
    };

    ICollection<Supplier> suppliers = new HashSet<Supplier>
    {
        new Supplier { City = "Phoenix", CompanyName = "Supplier1",
BankAccountNumber = "0123456789012345", Street = "St. Pierre", ZipCode = "55-
6666"},
        new Supplier { City = "Beijing", CompanyName = "Supplier2",
BankAccountNumber = "3453453463242352", Street = "Tianan", ZipCode = "77-888"},
        new Supplier { City = "Nan Madol", CompanyName = "Supplier3",
BankAccountNumber = "4673529174837284", Street = "Uloka", ZipCode = "99-999"},
    };

    productContext.Customers.AddRange(customers);
    productContext.Suppliers.AddRange(suppliers);

    productContext.SaveChanges();
}

static void QueryryFromDatabase()
{
    ProductContext productContext = new ProductContext();

    var comp_query = from comp in productContext.Companies
                     select comp;

    Console.WriteLine("Companies:");
    foreach (var company in comp_query)
    {
        Console.WriteLine(company);
    }
    Console.WriteLine();

    var supp_query = from supp in productContext.Suppliers
                     select supp;

    Console.WriteLine("Suppliers:");
    foreach (var supplier in supp_query)
    {
        Console.WriteLine(supplier);
    }
    Console.WriteLine();

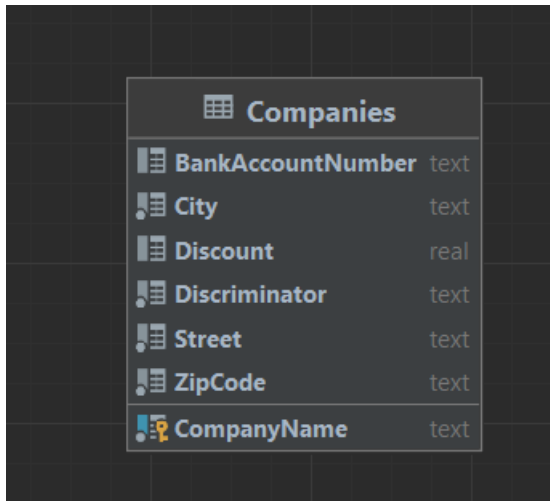
    var cust_query = from cust in productContext.Customers
                     select cust;

    Console.WriteLine("Customers");
    foreach (var cust in cust_query)
    {
        Console.WriteLine(cust);
    }
}

```

5.6 EFEKT

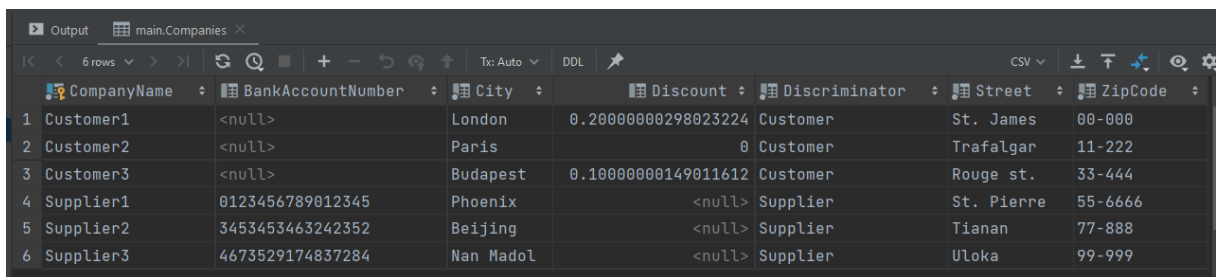
Po wykonaniu funkcji main w bazie istnieje tylko jedna tablica:



The screenshot shows a database schema diagram for a table named 'Companies'. The table has the following columns and data types:

Column Name	Data Type
BankAccountNumber	text
City	text
Discount	real
Discriminator	text
Street	text
ZipCode	text
CompanyName	text


Zawartość tablicy:



The screenshot shows a database table viewer displaying the contents of the 'main.Companies' table. The table has 6 rows and 7 columns: CompanyName, BankAccountNumber, City, Discount, Discriminator, Street, and ZipCode.

	CompanyName	BankAccountNumber	City	Discount	Discriminator	Street	ZipCode
1	Customer1	<null>	London	0.200000000298023224	Customer	St. James	00-000
2	Customer2	<null>	Paris	0	Customer	Trafalgar	11-222
3	Customer3	<null>	Budapest	0.100000000149011612	Customer	Rouge st.	33-444
4	Supplier1	0123456789012345	Phoenix	<null>	Supplier	St. Pierre	55-6666
5	Supplier2	3453453463242352	Beijing	<null>	Supplier	Tianan	77-888
6	Supplier3	4673529174837284	Nan Madol	<null>	Supplier	Uloka	99-999

5.7 POBRANIE I WYPISANIE DANYCH



The screenshot shows a Visual Studio debug console window with the following JSON output:

```
Konsola debugowania programu Microsoft Visual Studio

Companies:
{"CompanyName":"Customer1","Street":"St. James","City":"London","ZipCode":"00-000"}
{"CompanyName":"Customer2","Street":"Trafalgar","City":"Paris","ZipCode":"11-222"}
{"CompanyName":"Customer3","Street":"Rouge st.","City":"Budapest","ZipCode":"33-444"}
{"CompanyName":"Supplier1","Street":"St. Pierre","City":"Phoenix","ZipCode":"55-6666"}
{"CompanyName":"Supplier2","Street":"Tianan","City":"Beijing","ZipCode":"77-888"}
{"CompanyName":"Supplier3","Street":"Uloka","City":"Nan Madol","ZipCode":"99-999"}

Suppliers:
{"CompanyName":"Supplier1","Street":"St. Pierre","City":"Phoenix","ZipCode":"55-6666"}
{"CompanyName":"Supplier2","Street":"Tianan","City":"Beijing","ZipCode":"77-888"}
{"CompanyName":"Supplier3","Street":"Uloka","City":"Nan Madol","ZipCode":"99-999"}

Customers
{"CompanyName":"Customer1","Street":"St. James","City":"London","ZipCode":"00-000"}
{"CompanyName":"Customer2","Street":"Trafalgar","City":"Paris","ZipCode":"11-222"}
{"CompanyName":"Customer3","Street":"Rouge st.","City":"Budapest","ZipCode":"33-444"}
```

6 MAPOWANIE DZIEDZICZENIA: TABLE-PER-TYPE

6.1 KONTEKST

Kod kontekstu nie zmienił się względem poprzedniego podejścia.

6.2 KLASY

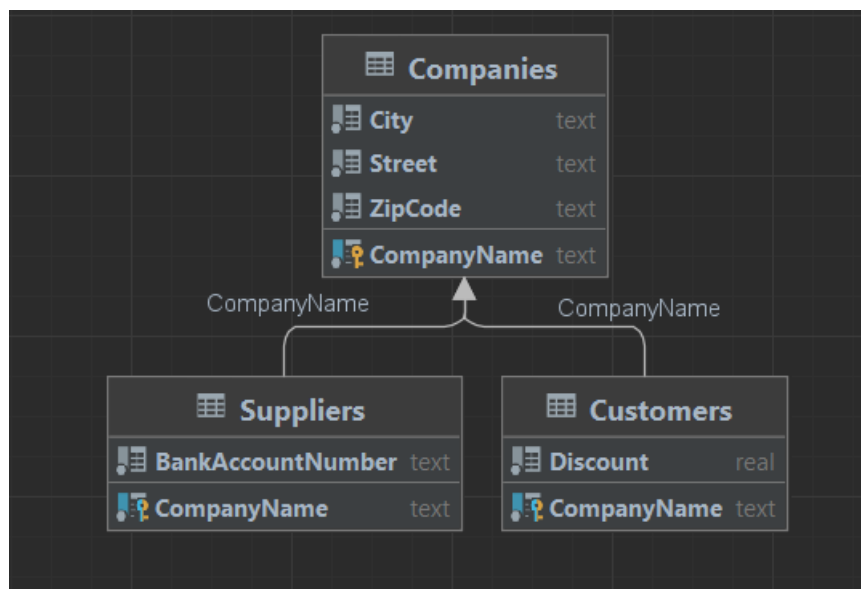
Aby zastosować tę strategię, wprowadzamy anotację `[Table(„nazwa_tabeli”)]` przed nazwą klas dziedziczących. Reszta kodu pozostaje taka sama.

6.3 KLASA PROGRAM

Kod funkcji nie zmienił się względem poprzedniego podejścia.

6.4 EFEKT

Po wykonaniu w bazie istnieją 3 tablice, osobna dla każdej klasy.



Schemat tabel w bazie danych dokładnie odzwierciedla hierarchię dziedziczenia schematu z polecenia.

Zawartości tablic:

Output		main.Suppliers	
	CompanyName	BankAccountNumber	
1	Supplier1	0123456789012345	
2	Supplier2	3453453463242352	
3	Supplier3	4673529174837284	

Output		main.Customers	
	CompanyName	Discount	
1	Customer1	0.20000000298023224	
2	Customer2	0	
3	Customer3	0.10000000149011612	

Output		main.Companies			
	CompanyName	City	Street	ZipCode	
1	Customer1	London	St. James	00-000	
2	Customer2	Paris	Trafalgar	11-222	
3	Customer3	Budapest	Rouge st.	33-444	
4	Supplier1	Phoenix	St. Pierre	55-6666	
5	Supplier2	Beijing	Tianan	77-888	
6	Supplier3	Nan Madol	Uloka	99-999	

6.5 POBIERANIE I WYPISYWANIE DANYCH

```
Konsola debugowania programu Microsoft Visual Studio

Companies:
{"CompanyName":"Customer1","Street":"St. James","City":"London","ZipCode":"00-000"}
{"CompanyName":"Customer2","Street":"Trafalgar","City":"Paris","ZipCode":"11-222"}
{"CompanyName":"Customer3","Street":"Rouge st.","City":"Budapest","ZipCode":"33-444"}
{"CompanyName":"Supplier1","Street":"St. Pierre","City":"Phoenix","ZipCode":"55-6666"}
{"CompanyName":"Supplier2","Street":"Tianan","City":"Beijing","ZipCode":"77-888"}
{"CompanyName":"Supplier3","Street":"Uloka","City":"Nan Madol","ZipCode":"99-999"}

Suppliers:
{"CompanyName":"Supplier1","Street":"St. Pierre","City":"Phoenix","ZipCode":"55-6666"}
{"CompanyName":"Supplier2","Street":"Tianan","City":"Beijing","ZipCode":"77-888"}
{"CompanyName":"Supplier3","Street":"Uloka","City":"Nan Madol","ZipCode":"99-999"}

Customers
{"CompanyName":"Customer1","Street":"St. James","City":"London","ZipCode":"00-000"}
{"CompanyName":"Customer2","Street":"Trafalgar","City":"Paris","ZipCode":"11-222"}
{"CompanyName":"Customer3","Street":"Rouge st.","City":"Budapest","ZipCode":"33-444"}
```

7 MAPOWANIE DZIEDZICZENIA: TABLE-PER-CONCRETE-TYPE

7.1 NADPISUJEMY METODĘ ONMODELCREATION Z UŻYCIEM FLUENTAPI W KONTEKŚCIE PRODUCTCONTEXT W NASTĘPUJĄCY SPOSÓB:

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

7.2 KONTEKST

Nadpisujemy metodę OnModelCreating i używamy odpowiedniej metody dla encji Company

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

7.3 KLASY

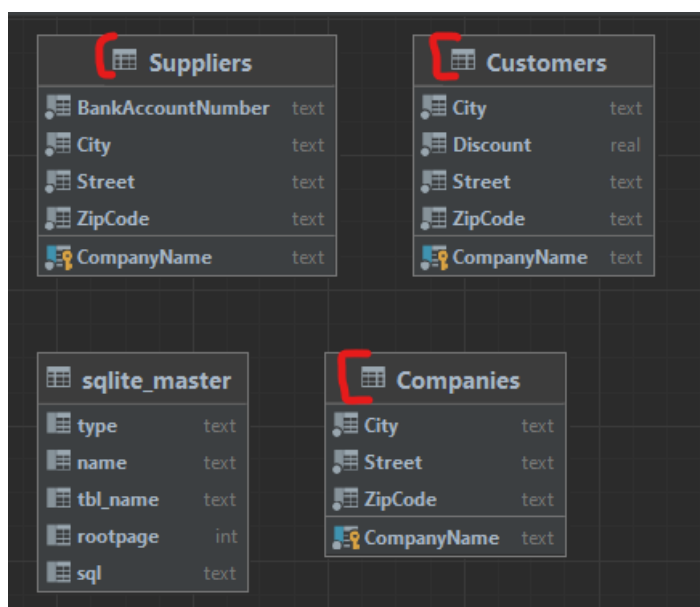
Usuwamy anotacje [Table(„nazwa_tabeli”)] z klas Customer i Supplier

7.4 KLASA PROGRAM

Bez z mian w porównaniu z poprzednim

7.5 EFEKT

Schemat po wykonaniu:



Podejście Table-Per-Concrete-Type powoduje, że tablice Suppliers i Customers mają wszystkie kolumny, które posiada tablica Companies. Istnieje też osobna tablica Companies, zawierająca obiekty, które nie są klasy Customer ani Supplier (polecenie ani diagram UML nie sugerowały, że klasa ta powinna być abstrakcyjna, więc jest możliwość stworzenia jej instancji). Gdyby uczynić klasę Company abstrakcyjną, wtedy podejście Table-Per-Concrete-Type nie wygenerowałoby tablicy Companies.

Zawartości tablic:

main.Customers					
CompanyName	City	Discount	Street	ZipCode	
1 Customer1	London	0.20000000298023224	St. James	00-000	
2 Customer2	Paris		0 Trafalgar	11-222	
3 Customer3	Budapest	0.10000000149011612	Rouge st.	33-444	

main.Suppliers					
CompanyName	BankAccountNumber	City	Street	ZipCode	
1 Supplier1	0123456789012345	Phoenix	St. Pierre	55-6666	
2 Supplier2	3453453463242352	Beijing	Tianan	77-888	
3 Supplier3	4673529174837284	Nan Madol	Uloka	99-999	

main.Companies				
CompanyName	City	Street	ZipCode	
0 rows				

7.6 FUNKCJA DO POBRANIA I WYPISANIA DANYCH

```
static void QueryFromDatabase()
{
    ProductContext productContext = new ProductContext();

    var supp_query = from supp in productContext.Suppliers
                     select supp;

    foreach (var supplier in supp_query)
    {
        Console.WriteLine(supplier);
    }

    var cust_query = from cust in productContext.Customers
                     select cust;

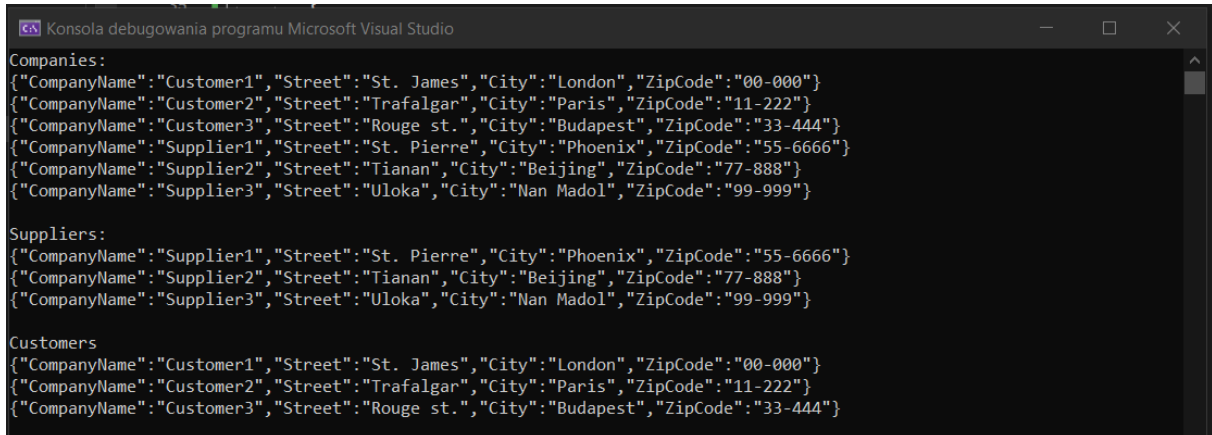
    foreach (var cust in cust_query)
```

```

    {
        Console.WriteLine(cust);
    }
}

```

7.7 POBRANIE I WYPISANIE DANYCH



```

Konsola debugowania programu Microsoft Visual Studio

Companies:
{"CompanyName":"Customer1","Street":"St. James","City":"London","ZipCode":"00-000"}
{"CompanyName":"Customer2","Street":"Trafalgar","City":"Paris","ZipCode":"11-222"}
{"CompanyName":"Customer3","Street":"Rouge st.","City":"Budapest","ZipCode":"33-444"}
{"CompanyName":"Supplier1","Street":"St. Pierre","City":"Phoenix","ZipCode":"55-6666"}
{"CompanyName":"Supplier2","Street":"Tianan","City":"Beijing","ZipCode":"77-888"}
{"CompanyName":"Supplier3","Street":"Uloka","City":"Nan Madol","ZipCode":"99-999"}

Suppliers:
{"CompanyName":"Supplier1","Street":"St. Pierre","City":"Phoenix","ZipCode":"55-6666"}
{"CompanyName":"Supplier2","Street":"Tianan","City":"Beijing","ZipCode":"77-888"}
{"CompanyName":"Supplier3","Street":"Uloka","City":"Nan Madol","ZipCode":"99-999"}

Customers
{"CompanyName":"Customer1","Street":"St. James","City":"London","ZipCode":"00-000"}
{"CompanyName":"Customer2","Street":"Trafalgar","City":"Paris","ZipCode":"11-222"}
{"CompanyName":"Customer3","Street":"Rouge st.","City":"Budapest","ZipCode":"33-444"}

```

8 PORÓWNANIE MAPOWAŃ

8.1 TABLE-PER-HIERARCHY

Jest to domyślna strategia w Entity Frameworku. Polega ona na, jak widać w punkcie 5., na umieszczeniu wszystkich obiektów w jednej tablicy. Typy obiektów możemy rozróżniać po kolumnie Discriminator (nazwę tę można zmienić w razie potrzeby). Zaletą tego podejścia jest stosunkowo łatwy konceptualnie model powstałej bazy. Wadą jest to, że powstała baza jest nieznormalizowana, a do tego tablica zawierająca obiekty z hierarchii może zawierać bardzo wiele wartości null.

8.2 TABLE-PER-TYPE

W tym podejściu tworzona jest osobna tabela dla każdej z klas (także abstrakcyjnych). Tabela każdej klasy zawiera tylko kolumny (pola) zadeklarowane przez siebie, a tabele klas dziedziczących odnoszą się do swojej „diedziczonej części” przez klucz obcy do odpowiedniej encji w tabeli „nadrzędnej”.

Zaletą tego podejścia jest to, że powstała w ten sposób baza danych będzie znormalizowana i nie będą występować wartości null w kolumnach, których poszczególne encje nie potrzebowały (bo tutaj takie kolumny nie istnieją). Dodatkowo zmodyfikowanie takiej bazy sprowadzi się do dodania kolejnej tablicy z kluczami obcymi to tabeli nadrzędnej.

8.3 TABLE-PER-CONCRETE-TYPE

Podejście to jest podobne do table-per-type, lecz nie są tworzone tabele dla klas abstrakcyjnych. Jeśli zatem mamy klasy dziedziczące po klasie abstrakcyjnej, to właściwości tej klasy będą „wbudowane” niezależnie w każdą klasę dziedziczącą (o ile one same nie są abstrakcyjne). Pomiedzy takimi klasami o wspólnym abstrakcyjnym przodku nie ma żadnej relacji (mają jedynie podobne kolumny).