Raport - Entity Framework

Ćwiczenie/zadanie

Autorzy: Seweryn Tasior, Filip Węgrzyn

zad 2

a)

- ProdContext.cs

```
using Microsoft.EntityFrameworkCore;
public class ProdContext : DbContext
{
    public DbSet<Product> Products { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite("Datasource=MyProductDatabase");
    }
}
```

- Product.cs

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

        public int? SupplierID { get; set; }
        public Supplier? Supplier { get; set; }
    }
```

- Supplier.cs

```
    using System.Collections.ObjectModel;

    public class Supplier
    {
        public int SupplierID { get; set; }
        public String? CompanyName { get; set; }
        public String? Street { get; set; }
        public String? City { get; set; }

    }
```

- Program.cs

```
    using System;
    using Microsoft.EntityFrameworkCore;
    ProdContext context = new ProdContext();
    //Tworzenie supplliera
    Supplier supplier = new Supplier { CompanyName = "Okta", Street="Krola Augusta",City="Krakow" };
    //Przypisanie produktu do dostawcy
    context.Suppliers.Add(supplier);
    //Wyszukiwanie produktu
    var query2 = from prod in context.Products
                 where prod.ProductName == "kredki"
                 select prod;
    foreach (Product product in query2)
    {
        Console.WriteLine(product.ProductName);
        product.Supplier = supplier;
    }
    context.SaveChanges();
```

- Następnie zostały wykonane komendy `dotnet build` i `dotnet run`
- Products schema

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsInStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers"
("SupplierID")
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
```

- Suppliers schema

```
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
);
```

- Zawartość tabeli Products

```
sqlite> select * from products
   ...> ;
1|Flamaster|0|
2|Flamaster|0|
3|kredki|0|1
```

- Zawartość tabeli Suppliers

```
sqlite> select * from suppliers
   ...> ;
1|Okta|Krola Augusta|Krakow
```

b)

- Product.cs

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

}
```

- Supplier.cs

```
using System.Collections.ObjectModel;

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; }
}
```

- Program.cs

```
using System;
using Microsoft.EntityFrameworkCore;

ProdContext context = new ProdContext();

//Tworzenie kilka produktów
Product p1 = new Product { ProductName  = "Pilka", UnitsInStock =10 };
Product p2 = new Product { ProductName  = "Deska", UnitsInStock =2 };
Product p3 = new Product { ProductName  = "Kubek", UnitsInStock =4 };

// Dodanie produktów do tabeli Products
context.Products.Add(p1);
context.Products.Add(p2);
context.Products.Add(p3);
context.SaveChanges();

// Wyszukanie dostawcy
var supplier = context.Suppliers
```

```
            .Where(s => s.CompanyName == "Okta")
            .FirstOrDefault();
        if (supplier == null)
            return;
        if (supplier.Products == null)
            supplier.Products = new List<Product>();

        // Dodanie produktów do dostawcy
        supplier.Products.Add(p1);
        supplier.Products.Add(p2);
        supplier.Products.Add(p3);

        context.SaveChanges();
```

- Następnie zostały wykonane komendy `dotnet build` i `dotnet run`
- Products schema

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsInStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers"
("SupplierID")
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
```

- Suppliers schema

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsInStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers"
("SupplierID")
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
```

- Zawartość tabeli Products i Suppliers

```
sqlite> select * from suppliers;
1|Okta|Krola Augusta|Krakow
sqlite> select * from products;
1|Flamaster|0|
2|Flamaster|0|
3|kredki|0|1
4|Pilka|10|1
5|Deska|2|1
6|Kubek|4|1
```

c)

- Product.cs

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

- Supplier.cs

```
    using System.Collections.ObjectModel;

    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; }
    }
```

- Program.cs

```
using System;
using Microsoft.EntityFrameworkCore;

ProdContext context = new ProdContext();

//Tworzenie kilku produktów
Product p3 = new Product { ProductName  = "Hak", UnitsInStock =11 };
Product p4 = new Product { ProductName  = "Skakanka", UnitsInStock =9 };
Product p5 = new Product { ProductName  = "Balon", UnitsInStock =20 };

// Dodanie produktów do tabeli Products
context.Products.Add(p3);
context.Products.Add(p4);
context.Products.Add(p5);
context.SaveChanges();

// Wyszukanie dostawcy
var supplier = context.Suppliers
    .Where(s => s.CompanyName == "Okta")
    .FirstOrDefault();
if (supplier == null)
    return;
if (supplier.Products == null)
    supplier.Products = new List<Product>();

// Dodanie produktów do dostawcy
supplier.Products.Add(p3);
supplier.Products.Add(p4);
supplier.Products.Add(p5);

// Dodanie suppliera do produktów
p3.Supplier = supplier;
p4.Supplier = supplier;
p5.Supplier = supplier;

context.SaveChanges();
```

- Następnie zostały wykonane komendy `dotnet build` i `dotnet run`
- Products schema

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsInStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers"
("SupplierID")
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
```

- Suppliers schema

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsInStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers"
("SupplierID")
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
```

- Zawartość tabeli Products i Suppliers

```
sqlite> select * from suppliers;
1|Okta|Krola Augusta|Krakow
sqlite> select * from products;
1|Flamaster|0|
2|Flamaster|0|
3|kredki|0|1
4|Pilka|10|1
5|Deska|2|1
6|Kubek|4|1
7|Hak|11|1
8|Skakanka|9|1
9|Balon|20|1
```

d)

- Product.cs

```
using System.Collections.ObjectModel;
public class Product
{
    public Product(){
        this.Invoices = new HashSet<Invoice>();
    }
    public int ProductID { get; set; }
    public String? ProductName { get; set; }
    public int UnitsInStock { get; set; }

    public int? SupplierID { get; set; }
    public Supplier? Supplier { get; set; }

    public virtual ICollection<Invoice>? Invoices { get; set; }
}
```

- Invoice.cs

```
using System.Collections.ObjectModel;

public class Invoice
{
    public Invoice(){
        this.Products = new HashSet<Product>();
    }
    public int InvoiceID { get; set; }
    public int? InvoiceNumber { get; set; }
    public int? Quantity { get; set; }

    public virtual ICollection<Product>? Products { get; set; }
}
```

- Program.cs

```
using System;
using Microsoft.EntityFrameworkCore;

if (File.Exists("MyProductDatabase"))
{
    File.Delete("MyProductDatabase");
    Console.WriteLine("Stara baza danych usunięta.");
}

ProdContext context = new ProdContext();
context.Database.EnsureCreated();

// Tworzenie kilku produktów
Product p1 = new Product { ProductName = "Wiosla", UnitsInStock = 8 };
Product p2 = new Product { ProductName = "Szklanka", UnitsInStock = 21 };
Product p3 = new Product { ProductName = "Karton", UnitsInStock = 23 };
Product p4 = new Product { ProductName = "Zeszyt", UnitsInStock = 12 };

// Dodanie produktów do tabeli Products
context.Products.AddRange(p1, p2, p3, p4);
context.SaveChanges();

// Tworzenie kilku faktur
Invoice i1 = new Invoice { InvoiceNumber = 342, Quantity = 3 };
Invoice i2 = new Invoice { InvoiceNumber = 654, Quantity = 5 };
Invoice i3 = new Invoice { InvoiceNumber = 111, Quantity = 6 };

i1.Products = new List<Product>();
i2.Products = new List<Product>();
i3.Products = new List<Product>();

p1.Invoices = new List<Invoice>();
p2.Invoices = new List<Invoice>();
p3.Invoices = new List<Invoice>();
p4.Invoices = new List<Invoice>();

// Dodanie faktur do tabeli Invoices
context.Invoices.AddRange(i1, i2, i3);
context.SaveChanges();

// Sprzedaż produktów - przypisanie ich do faktur
// Faktura 1
i1.Products.Add(p1);
i1.Products.Add(p2);
p1.Invoices.Add(i1);
p2.Invoices.Add(i1);
```

```
// Faktura 2
i2.Products.Add(p3);
p3.Invoices.Add(i2);

// Faktura 3
i3.Products.Add(p3);
i3.Products.Add(p4);
p3.Invoices.Add(i3);
p4.Invoices.Add(i3);

// Zapisanie relacji
context.SaveChanges();

// Wyświetlenie produktów na fakturach
Console.WriteLine("\nZawartość faktur:");

var invoices = context.Invoices
    .Include(i => i.Products)
    .ToList();

foreach (var invoice in invoices)
{
    Console.WriteLine($"Faktura nr {invoice.InvoiceNumber} (ID: {invoice.InvoiceID}):");
    if (invoice.Products != null)
    {
        foreach (var product in invoice.Products)
        {
            Console.WriteLine($" - {product.ProductName} (na stanie: {product.UnitsInStock})");
        }
    }
    Console.WriteLine();
}

// Wyświetlenie faktur dla produktów
Console.WriteLine("\nFaktury dla poszczególnych produktów:");

var products = context.Products
    .Include(p => p.Invoices)
    .ToList();

foreach (var product in products)
{
    Console.WriteLine($"Produkt: {product.ProductName}");
    if (product.Invoices != null && product.Invoices.Any())
    {
        Console.WriteLine("Sprzedany na fakturach:");
        foreach (var inv in product.Invoices)
        {
            Console.WriteLine($" - Faktura nr {inv.InvoiceNumber}");
        }
    }
    else
    {
        Console.WriteLine("Nie został jeszcze sprzedany.");
    }
    Console.WriteLine();
}

context.SaveChanges();
```

- Następnie zostały wykonane komendy `dotnet build` i `dotnet run`. Wynik poniżej:

```
PS C:\Users\Seweryn\Desktop\agh-computer\sem4\databases-labs\lab-3> dotnet run
Stara baza danych usunieta.

Zawartosc faktur:
Faktura nr 342 (ID: 1):
 - Wiosla (na stanie: 8)
 - Szklanka (na stanie: 21)

Faktura nr 654 (ID: 2):
 - Karton (na stanie: 23)

Faktura nr 111 (ID: 3):
 - Karton (na stanie: 23)
 - Zeszyt (na stanie: 12)


Faktury dla poszczególnych produktów:
Produkt: Wiosla
Sprzedany na fakturach:
 - Faktura nr 342

Produkt: Szklanka
Sprzedany na fakturach:
 - Faktura nr 342

Produkt: Karton
Sprzedany na fakturach:
 - Faktura nr 654
 - Faktura nr 111

Produkt: Zeszyt
Sprzedany na fakturach:
 - Faktura nr 111
```

- Products i Invoices schemas

```
sqlite> .schema Products
CREATE TABLE IF NOT EXISTS "Products" (
    "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
    "ProductName" TEXT NULL,
    "UnitsInStock" INTEGER NOT NULL,
    "SupplierID" INTEGER NULL,
    CONSTRAINT "FK_Products_Suppliers_SupplierID" FOREIGN KEY ("SupplierID") REFERENCES "Suppliers
" ("SupplierID")
);
CREATE INDEX "IX_Products_SupplierID" ON "Products" ("SupplierID");
sqlite> .schema Invoices
CREATE TABLE IF NOT EXISTS "Invoices" (
    "InvoiceID" INTEGER NOT NULL CONSTRAINT "PK_Invoices" PRIMARY KEY AUTOINCREMENT,
    "InvoiceNumber" INTEGER NULL,
    "Quantity" INTEGER NULL
);
```

- InvoiceProduct schema

```
sqlite> .schema InvoiceProduct
CREATE TABLE IF NOT EXISTS "InvoiceProduct" (
    "InvoicesInvoiceID" INTEGER NOT NULL,
    "ProductsProductID" INTEGER NOT NULL,
    CONSTRAINT "PK_InvoiceProduct" PRIMARY KEY ("InvoicesInvoiceID", "ProductsProductID"
),
    CONSTRAINT "FK_InvoiceProduct_Invoices_InvoicesInvoiceID" FOREIGN KEY ("InvoicesInvo
iceID") REFERENCES "Invoices" ("InvoiceID") ON DELETE CASCADE,
    CONSTRAINT "FK_InvoiceProduct_Products_ProductsProductID" FOREIGN KEY ("ProductsProd
uctID") REFERENCES "Products" ("ProductID") ON DELETE CASCADE
);
CREATE INDEX "IX_InvoiceProduct_ProductsProductID" ON "InvoiceProduct" ("ProductsProduct
ID");
```

- Zawartość tabeli Products i Invoices

```
sqlite> select * from products;
1|Wiosla|8|
2|Szklanka|21|
3|Karton|23|
4|Zeszyt|12|
sqlite> select * from invoices
   ...> ;
1|342|3
2|654|5
3|111|6
```

- Zawartość tabeli InvoiceProduct

```
sqlite> select * from invoiceproduct
   ...> ;
1|1
1|2
2|3
3|3
3|4
```

e)

- ProdContext.cs

```
using Microsoft.EntityFrameworkCore;
public class ProdContext : DbContext
{
    public DbSet<Company> Companies { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Invoice> Invoices { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=MyProductDatabase");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Company>()
            .HasDiscriminator<string>("CompanyType")
            .HasValue<Supplier>("Supplier")
            .HasValue<Customer>("Customer");
    }
}
```

- Company.cs

```
    using System.Collections.ObjectModel;

    public abstract class Company
    {
        public int CompanyID { get; set; }
        public string? CompanyName { get; set; }
        public string? Street { get; set; }
        public string? City { get; set; }
        public string? ZipCode { get; set; }

        // Pole do identyfikacji typu
        public string CompanyType { get; set; }
    }
```

- Customer.cs

```
    using System.Collections.ObjectModel;

    public class Customer : Company
    {
        public int? Discount { get; set; }

        public Customer()
        {
            CompanyType = "Customer";
        }
    }
```

- Supplier.cs

```
using System.Collections.ObjectModel;
 public class Supplier : Company
 {
     public string? BankAccountNumber { get; set; }

     public Supplier()
     {
         CompanyType = "Supplier";
     }
 }
```

- Program.cs

```
 if (File.Exists("MyProductDatabase"))
 {
     File.Delete("MyProductDatabase");
     Console.WriteLine("Stara baza danych usunięta.");
 }

 // Inicjalizacja
 ProdContext context = new ProdContext();
 context.Database.EnsureCreated();
 Console.WriteLine("Utworzono bazę danych z hierarchią dziedziczenia.");

 // Tworzenie dostawcy
 Supplier supplier = new Supplier
 {
     CompanyName = "Biuromat",
     Street = "Długa 15",
     City = "Warszawa",
     ZipCode = "01-234",
     BankAccountNumber = "PL12345678901234567890123456"
 };
 Supplier supplier2 = new Supplier
 {
     CompanyName = "WilkoPol",
     Street = "Polna 15",
     City = "Koszalin",
     ZipCode = "23-41",
     BankAccountNumber = "PL2134343434113"
 };
 context.Suppliers.Add(supplier);
 context.Suppliers.Add(supplier2);

 // Tworzenie klienta
 Customer customer = new Customer
 {
     CompanyName = "AGH",
     Street = "Al. Mickiewicza 30",
     City = "Kraków",
     ZipCode = "30-059",
     Discount = 10
 };
 Customer customer2 = new Customer
 {
     CompanyName = "Deloitte",
     Street = "Al. Jana Pawła II 22",
     City = "Kraków",
     ZipCode = "32-259",
     Discount = 12
 };
 context.Customers.Add(customer);
 context.Customers.Add(customer2);
 context.SaveChanges();

 // Wyświetlenie wszystkich firm
 Console.WriteLine("\nWszystkie firmy:");
 var companies = context.Companies.ToList();
 foreach (var company in companies)
 {
     Console.WriteLine($"ID: {company.CompanyID}, Nazwa: {company.CompanyName}, Typ: {company.CompanyType}, Miasto: {company.City}");
 }
```

- Następnie zostały wykonane komendy `dotnet build` i `dotnet run`. Wynik poniżej:

```
PS C:\Users\Seweryn\Desktop\agh-computer\sem4\databases-labs\lab-3> dotnet run
Stara baza danych usunieta.
Utworzono baze danych z hierarchia dziedziczenia.

Wszystkie firmy:
ID: 1, Nazwa: AGH, Typ: Customer, Miasto: Kraków
ID: 2, Nazwa: Deloitte, Typ: Customer, Miasto: Kraków
ID: 3, Nazwa: Biuromat, Typ: Supplier, Miasto: Warszawa
ID: 4, Nazwa: WilkoPol, Typ: Supplier, Miasto: Koszalin
```

- Companies schema

```
sqlite> .schema Companies
CREATE TABLE IF NOT EXISTS "Companies" (
    "CompanyID" INTEGER NOT NULL CONSTRAINT "PK_Companies" PRIMARY KEY AUTOINCREMENT,
    "CompanyName" TEXT NULL,
    "Street" TEXT NULL,
    "City" TEXT NULL,
    "ZipCode" TEXT NULL,
    "CompanyType" TEXT NOT NULL,
    "Discount" INTEGER NULL,
    "BankAccountNumber" TEXT NULL
);
```

- Zawartość tabeli Companies

```
sqlite> select * from companies
   ...> ;
1|AGH|Al. Mickiewicza 30|Kraków|30-059|Customer|10|
2|Deloitte|Al. Jana Pawla II 22|Kraków|32-259|Customer|12|
3|Biuromat|Dluga 15|Warszawa|01-234|Supplier||PL12345678901234567890123456
4|WilkoPol|Polna 15|Koszalin|23-41|Supplier||PL2134343434113
sqlite>
```

f)

- ProdContext.cs

```
using Microsoft.EntityFrameworkCore;
public class ProdContext : DbContext
{
    ...

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Company>().ToTable("Companies");
        modelBuilder.Entity<Supplier>().ToTable("Suppliers");
        modelBuilder.Entity<Customer>().ToTable("Customers");
    }
}
```

- Company.cs, Customer.cs i Supplier.cs takie same
- Program.cs

```
    if (File.Exists("MyProductDatabase"))
    {
        File.Delete("MyProductDatabase");
        Console.WriteLine("Stara baza danych usunięta.");
    }

    // Inicjalizacja
    ProdContext context = new ProdContext();
    context.Database.EnsureCreated();
    Console.WriteLine("Utworzono bazę danych z hierarchią dziedziczenia.");

    // Tworzenie dostawcy
    Supplier supplier = new Supplier
    {
        CompanyName = "Biuromat",
        Street = "Długa 15",
        City = "Warszawa",
        ZipCode = "01-234",
        BankAccountNumber = "PL12345678901234567890123456"
    };
    Supplier supplier2 = new Supplier
    {
        CompanyName = "WilkoPol",
        Street = "Polna 15",
        City = "Koszalin",
        ZipCode = "23-41",
```

```
            BankAccountNumber = "PL2134343434113"
        };
        context.Suppliers.Add(supplier);
        context.Suppliers.Add(supplier2);

        // Tworzenie klienta
        Customer customer = new Customer
        {
            CompanyName = "AGH",
            Street = "Al. Mickiewicza 30",
            City = "Kraków",
            ZipCode = "30-059",
            Discount = 10
        };
        Customer customer2 = new Customer
        {
            CompanyName = "Deloitte",
            Street = "Al. Jana Pawła II 22",
            City = "Kraków",
            ZipCode = "32-259",
            Discount = 12
        };
        context.Customers.Add(customer);
        context.Customers.Add(customer2);
        context.SaveChanges();

        // Wyświetlenie wszystkich firm
        Console.WriteLine("\nWszystkie firmy:");
        var companies = context.Companies.ToList();
        foreach (var company in companies)
        {
            Console.WriteLine($"ID: {company.CompanyID}, Nazwa: {company.CompanyName}, Typ: {company.CompanyType}, Miasto: {company.City}");
        }
```

- Następnie zostały wykonane komendy `dotnet build` i `dotnet run`. Wynik poniżej:

```
Stara baza danych usunieta.

=== Wszyscy dostawcy ===
ID: 3, Nazwa: Artykuly Biurowe XYZ, Miasto: Kraków, Nr konta: PL61109010140000071219812874
ID: 4, Nazwa: HurtPol, Miasto: Warszawa, Nr konta: PL27114020040000300201355387

=== Wszyscy klienci ===
ID: 1, Nazwa: Uniwersytet AGH, Miasto: Kraków, Rabat: 15%
ID: 2, Nazwa: TechStart SA, Miasto: Gdansk, Rabat: 8%

=== Wszystkie firmy ===
ID: 1, Nazwa: Uniwersytet AGH, Typ: Customer, Miasto: Kraków
ID: 2, Nazwa: TechStart SA, Typ: Customer, Miasto: Gdansk
ID: 3, Nazwa: Artykuly Biurowe XYZ, Typ: Supplier, Miasto: Kraków
ID: 4, Nazwa: HurtPol, Typ: Supplier, Miasto: Warszawa
```

- Companies, Suppliers and Customers schema

```
sqlite> .schema Customers
CREATE TABLE IF NOT EXISTS "Customers" (
    "CompanyID" INTEGER NOT NULL CONSTRAINT "PK_Customers" PRIMARY KEY AUTOINCREMENT,
    "Discount" INTEGER NULL,
    CONSTRAINT "FK_Customers_Companies_CompanyID" FOREIGN KEY ("CompanyID") REFERENCES "
Companies" ("CompanyID") ON DELETE CASCADE
);
sqlite> .schema Suppliers
CREATE TABLE IF NOT EXISTS "Suppliers" (
    "CompanyID" INTEGER NOT NULL CONSTRAINT "PK_Suppliers" PRIMARY KEY AUTOINCREMENT,
    "BankAccountNumber" TEXT NULL,
    CONSTRAINT "FK_Suppliers_Companies_CompanyID" FOREIGN KEY ("CompanyID") REFERENCES "
Companies" ("CompanyID") ON DELETE CASCADE
);
sqlite> .schema Companies
CREATE TABLE IF NOT EXISTS "Companies" (
    "CompanyID" INTEGER NOT NULL CONSTRAINT "PK_Companies" PRIMARY KEY AUTOINCREMENT,
    "CompanyName" TEXT NULL,
    "Street" TEXT NULL,
    "City" TEXT NULL,
    "ZipCode" TEXT NULL,
    "CompanyType" TEXT NOT NULL
);
```

- Zawartość tabel Companies, Suppliers and Customers

```
sqlite> SELECt * from companies
   ...> ;
1|Uniwersytet AGH|Al. Mickiewicza 30|Kraków|30-059|Customer
2|TechStart SA|Nowa 7|Gdansk|80-864|Customer
3|Artykuly Biurowe XYZ|Przemyslowa 10|Kraków|30-001|Supplier
4|HurtPol|Skladowa 5|Warszawa|00-950|Supplier
sqlite> SELECt * from suppliers
   ...> ;
3|PL61109010140000071219812874
4|PL27114020040000300201355387
sqlite> SELECt * from customers;
1|15
2|8
sqlite>
```

g)

Table-Per-Hierarchy (TPH)

W tej strukturze danych wszystkie dane są mapowane do jedenej tabeli, która korzysta z dyskriminatora `ComapnyType` do rozróżnienia typów. Pola specyficzne dla podklas mogą zaierać wartości NULL dla obiektów innych typów

Table-Per-Type (TPT)

Każda klasa jest mapowana do oddzielnej tabeli, natomiast tabela bazowa zawiera wspólne kolumny dla wszystkich typów. Tabele pochodne zawierają tylko kolumny specyficzne dla danego typu

Porównanie

- **Wydajność** TPH ma lepszą wydajność odczytu i zapisu, ponieważ wymaga dostępu do jednej tylko tabeli oraz nie wyamga JOIN-ów
- **Normalizacja** TPH jest mniej znormalizowana niż TPT. Łatwiejsze nakładanie ograniczeń integralnościowych w TPT.

Jakiej używać?

TPH najlepiej używać:

- Gdy hierarchia dziedziczenia jest stosunkowo stabilna
- Gdy podtypy mają niewiele unikalnych właściwości
- Gdy wydajność odczytu/zapisu jest priorytetem
- W przypadku aplikacji z dużą liczbą operacji CRUD

TPT najlepiej używać:

- Gdy hierarchia jest złożona z wieloma podtypami
- Gdy podtypy mają wiele unikalnych właściwości
- Gdy integralność danych i normalizacja są priorytetem