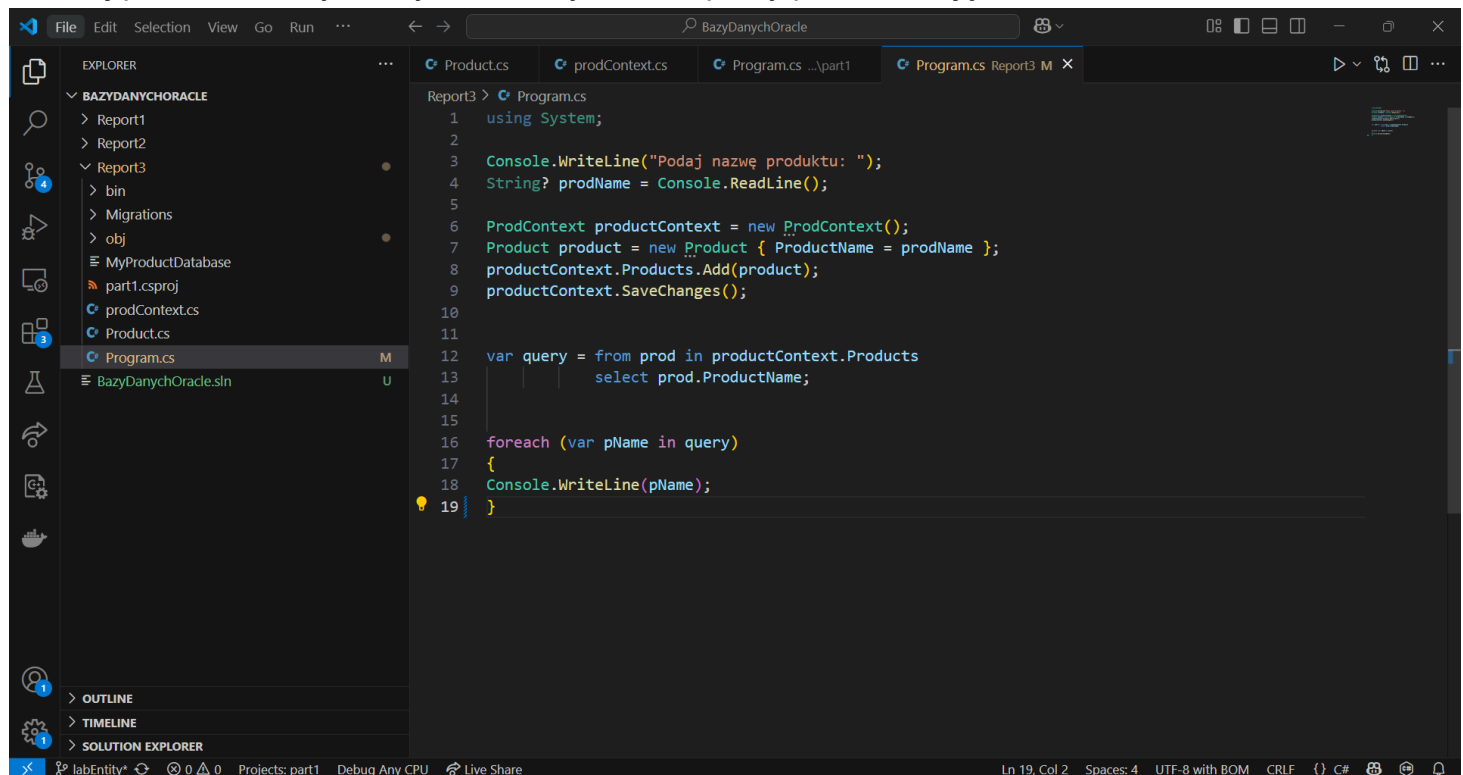


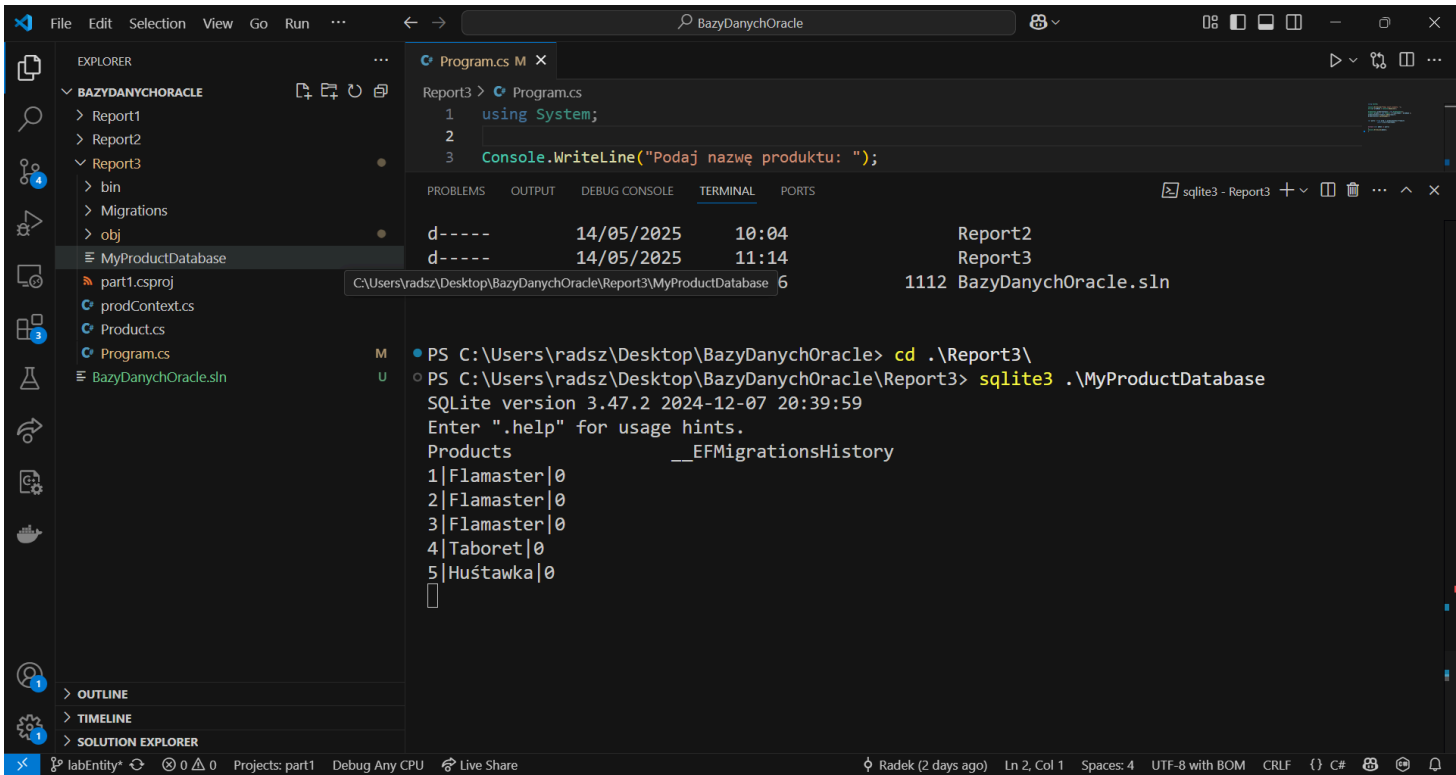
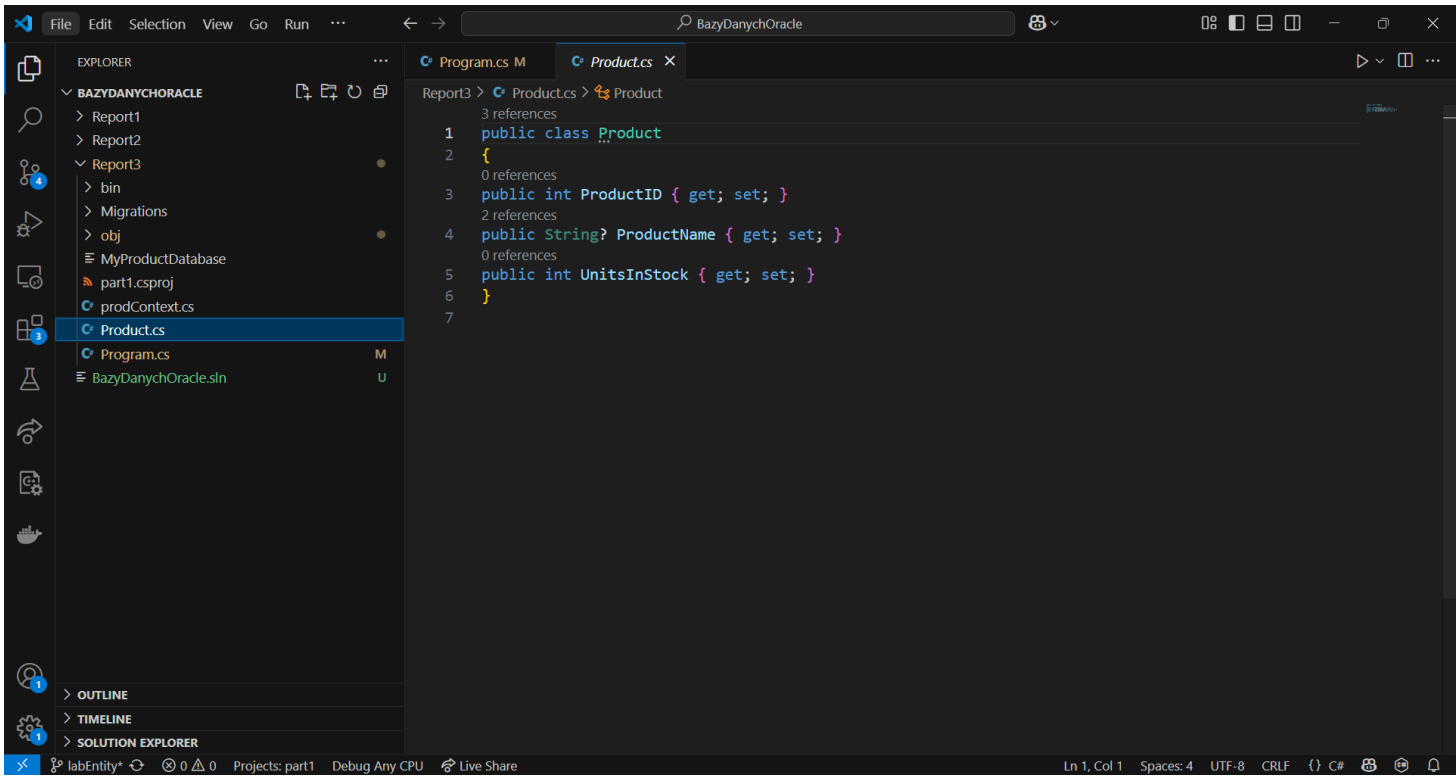
Sprawozdanie Entity Framework

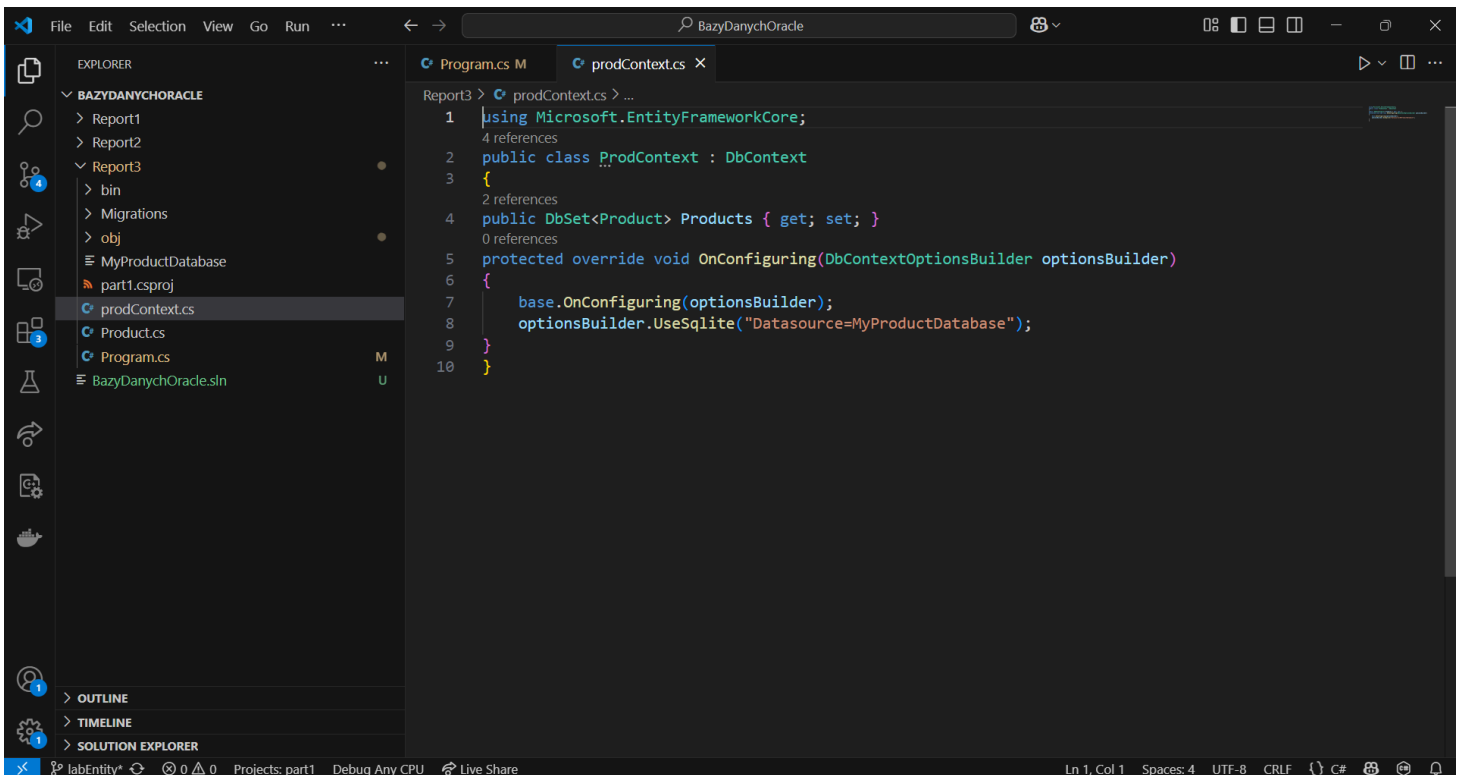
Autorzy: Radosław Szepielak, Kacper Wdowiak

Część 1:

Poniżej przedstawiamy zrzuty ekranu wykonania pracy podczas zajęć:







Część 2:

Zadanie a: Modyfikacja modelu i wprowadzenie Dostawcy (is supplied by)

Realizację zadania a) przedstawia kod poniżej:

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

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

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

```
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");
    }
}
```

```

using System;

class Program {

    static void Main()
    {
        Console.WriteLine("Podaj nazwę produktu: \n");
        String? prodName = Console.ReadLine();
        Console.WriteLine("Podaj liczbę dostępnych sztuk: \n");
        int quantity = Int32.Parse(Console.ReadLine());

        ProdContext productContext = new ProdContext();
        Product product = new Product { ProductName = prodName, UnitsInStock = quantity };
        productContext.Products.Add(product);

        bool createdNewSupplier = false;
        bool isValidChoice = false;
        Supplier? supplier = null;

        do
        {
            Console.WriteLine("Czy chcesz dodać nowego dostawcę? (tak/nie)");
            string choice = Console.ReadLine();

            switch (choice)
            {
                case "tak":
                    isValidChoice = true;
                    supplier = CreateNewSupplier();
                    createdNewSupplier = true;
                    break;
                case "nie":
                    isValidChoice = true;
                    ShowAllSuppliers(productContext);
                    supplier = FindSupplier(productContext);
                    break;
            }
        }

        while (!isValidChoice);

        product.Supplier = supplier;

        if (createdNewSupplier) {

```

```

        productContext.Suppliers.Add(supplier);
    }

    productContext.Products.Add(product);
    productContext.SaveChanges();
}

private static Supplier CreateNewSupplier()
{
    Console.WriteLine("\n\nuzupełnij nazwę dostawcy: ");
    string companyName = Console.ReadLine();
    Console.WriteLine("\nUzupełnij miasto: ");
    string city = Console.ReadLine();
    Console.WriteLine("\nUzupełnij ulicę: ");

    string street = Console.ReadLine();

    Supplier supplier = new Supplier
    {
        CompanyName = companyName,
        City = city,
        Street = street
    };
    Console.WriteLine($" \n Utworzono pomyślnie dostawcę: {supplier.CompanyName}");
    return supplier;
}

private static Supplier FindSupplier(ProdContext productContext) {
    Console.WriteLine("\n Podaj ID istniejącego dostawcy: ");
    int id = Int32.Parse(Console.ReadLine());

    var query = from sup in productContext.Suppliers
                where sup.SupplierID == id
                select sup;
    return query.FirstOrDefault();
}

private static void ShowAllSuppliers(ProdContext productContext)
{
    Console.WriteLine("\n Lista wszystkich istniejących dostawców: ");
    foreach (Supplier supplier in productContext.Suppliers)

```

```

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

```

Poniżej zamieszczamy zrzuty ekranu realizacji przez nas powyższego kodu wraz z uzyskanym rezultatem:

```

PS C:\Users\radsz\Desktop\BazyDanychOracle\Report3\a> dotnet build
Restore complete (0.3s)
  part1 succeeded (0.2s) -> bin\Debug\net9.0\part1.dll

Build succeeded in 0.9s
PS C:\Users\radsz\Desktop\BazyDanychOracle\Report3\a> dotnet run
Podaj nazwę produktu:
Flamaster
Podaj liczbę dostępnych sztuk:
10
Czy chcesz dodać nowego dostawcę? (tak/nie)
tak

uzupełnij nazwę dostawcy:
Hurtownia Apex

Uzupełnij miasto:
Kraków

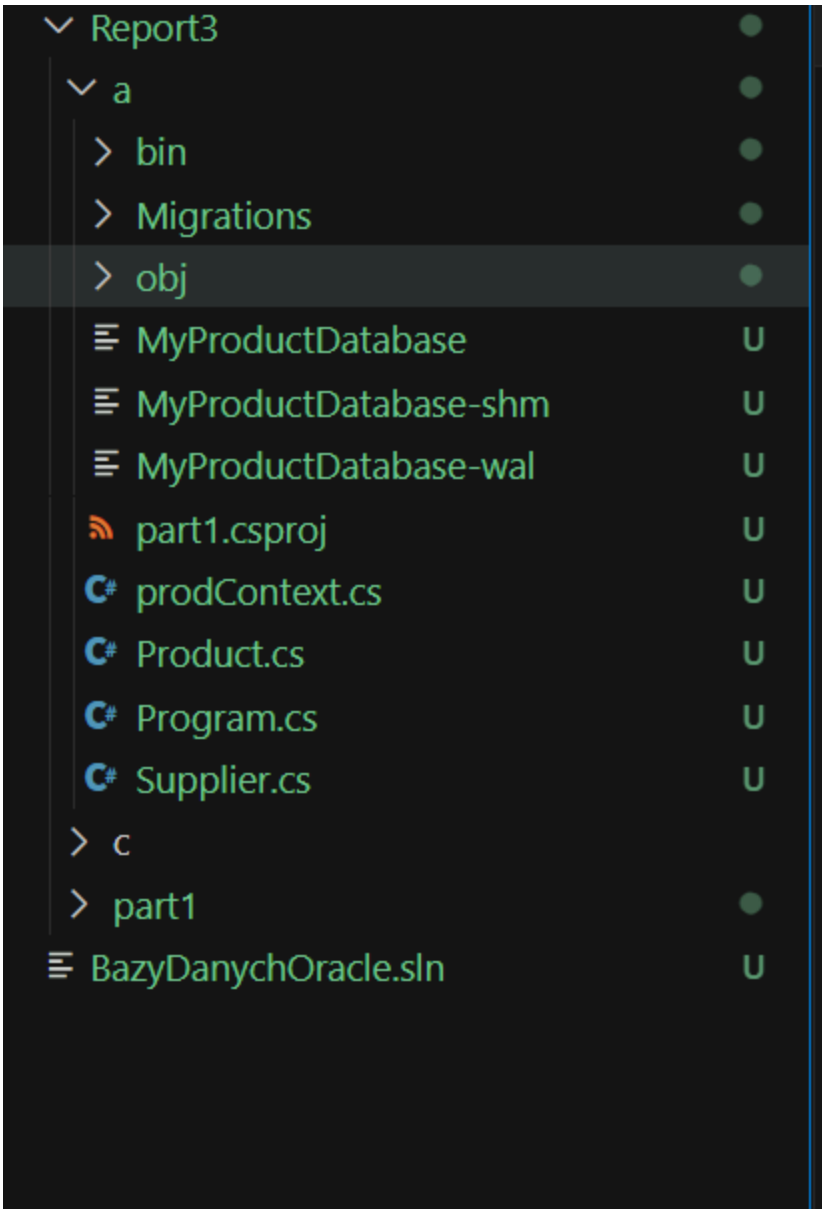
Uzupełnij ulicę:
Mickiewicza 14d

Utworzono pomyślnie dostawcę: Hurtownia Apex
PS C:\Users\radsz\Desktop\BazyDanychOracle\Report3\a>

```

SupplierID	CompanyName	Street	City
1	Hurtownia Apex	Mickiewicza 14d	Kraków

```
PS C:\Users\radsz\Desktop\BazyDanychOracle\Report3\a> sqlite3 .\MyProductDatabase
SQLite version 3.47.2 2024-12-07 20:39:59
Enter ".help" for usage hints.
Products          Suppliers          __EFMigrationsHistory
1|Flamaster|10|1   select * from Products;
1|Hurtownia Apex|Mickiewicza 14d|Kraków select * from Suppliers;
```



Zadanie b:

Realizację zadania b) przedstawia kod poniżej:


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

using System.Collections.Generic;

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

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

```
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_b.db");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Supplier>()
            .HasMany(s => s.Products)
            .WithOne()
            .HasForeignKey("SupplierID")
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Podaj nazwę produktu: ");
        string? prodName = Console.ReadLine();

        Console.WriteLine("Podaj liczbę dostępnych sztuk: ");
        int quantity = int.Parse(Console.ReadLine());

        ProdContext productContext = new ProdContext();
        Product product = new Product { ProductName = prodName, UnitsInStock = quantity };

        bool createdNewSupplier = false;
        bool isValidChoice = false;
        Supplier? supplier = null;

        do
        {
            Console.WriteLine("Czy chcesz dodać nowego dostawcę? (tak/nie)");
            string? choice = Console.ReadLine();

            switch (choice)
            {
                case "tak":
                    isValidChoice = true;
                    supplier = CreateNewSupplier();
                    createdNewSupplier = true;
                    break;
                case "nie":
                    isValidChoice = true;
                    ShowAllSuppliers(productContext);
                    supplier = FindSupplier(productContext);
                    break;
                default:
                    Console.WriteLine("Wybierz: tak / nie");
                    break;
            }
        } while (!isValidChoice);

        if (supplier != null)
```

```

{
    if (supplier.Products == null)
        supplier.Products = new System.Collections.Generic.List<Product>();

    supplier.Products.Add(product);

    if (createdNewSupplier)
    {
        productContext.Suppliers.Add(supplier);
    }

    productContext.SaveChanges();
    Console.WriteLine("Produkt został zapisany i przypisany do dostawcy.");
}
else
{
    Console.WriteLine("Nie udało się znaleźć lub utworzyć dostawcy.");
}
}

private static Supplier CreateNewSupplier()
{
    Console.WriteLine("Uzupełnij nazwę dostawcy: ");
    string companyName = Console.ReadLine();

    Console.WriteLine("Uzupełnij miasto: ");
    string city = Console.ReadLine();

    Console.WriteLine("Uzupełnij ulicę: ");
    string street = Console.ReadLine();

    Supplier supplier = new Supplier
    {
        CompanyName = companyName,
        City = city,
        Street = street
    };

    Console.WriteLine($"{nUtworzono dostawcę: {supplier.CompanyName}");
    return supplier;
}

private static Supplier? FindSupplier(ProdContext productContext)

```

```

{
    Console.WriteLine("Podaj ID istniejącego dostawcy: ");
    int id = int.Parse(Console.ReadLine());

    foreach (Supplier s in productContext.Suppliers)
    {
        if (s.SupplierID == id)
            return s;
    }

    return null;
}

private static void ShowAllSuppliers(ProdContext productContext)
{
    Console.WriteLine("Lista wszystkich dostawców:");
    foreach (Supplier supplier in productContext.Suppliers)
    {
        Console.WriteLine($"[{supplier.SupplierID}] {supplier.CompanyName}, {supplier.City}");
    }
}
}

```

Poniżej zamieszczamy zrzuty ekranu realizacji przez nas powyższego kodu wraz z uzyskanym rezultatem:

```

● PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\b> dotnet run
Podaj nazwę produktu:
Długopis
Podaj liczbę dostępnych sztuk:
10
Czy chcesz dodać nowego dostawcę? (tak/nie)
tak
Uzupełnij nazwę dostawcy:
Długopisy.pl
Uzupełnij miasto:
Krakow
Uzupełnij ulicę:
Długa 10

Utworzono dostawcę: Długopisy.pl
Produkt został zapisany i przypisany do dostawcy.

```

```
● PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\b> dotnet run
Podaj nazwę produktu:
Zeszyt
Podaj liczbę dostępnych sztuk:
15
Czy chcesz dodać nowego dostawcę? (tak/nie)
tak
Uzupełnij nazwę dostawcy:
Zeszyty.pl
Uzupełnij miasto:
Krakow
Uzupełnij ulicę:
Krotka 1

Utworzono dostawcę: Zeszyty.pl
Produkt został zapisany i przypisany do dostawcy.
```

```
● PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\b> dotnet run
Podaj nazwę produktu:
Olowek
Podaj liczbę dostępnych sztuk:
5
Czy chcesz dodać nowego dostawcę? (tak/nie)
tak
Uzupełnij nazwę dostawcy:
Olowki.pl
Uzupełnij miasto:
Krakow
Uzupełnij ulicę:
Srednia 5

Utworzono dostawcę: Olowki.pl
Produkt został zapisany i przypisany do dostawcy.
```

```
● PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\b> dotnet run
Podaj nazwę produktu:
Olowek
Podaj liczbę dostępnych sztuk:
5
Czy chcesz dodać nowego dostawcę? (tak/nie)
tak
Uzupełnij nazwę dostawcy:
Olowki.pl
Uzupełnij miasto:
Krakow
Uzupełnij ulicę:
Srednia 5

Utworzono dostawcę: Olowki.pl
Produkt został zapisany i przypisany do dostawcy.
```

```
sqlite> SELECT * FROM Suppliers;
1|Długopisy.pl|Długa 10|Krakow
2|Zeszyty.pl|Krotka 1|Krakow
3|Olowki.pl|Srednia 5|Krakow
sqlite> SELECT * FROM Products;
1|Długopis|10|1
2|Zeszyt|15|2
3|Olowek|5|3
```

Zadanie c: Zamodelowanie relacji dwustronnej (supplies & is supplied by)

Realizację zadania c) przedstawia kod poniżej:

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

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

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

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



```

using System;

class Program {

    static void Main()
    {
        Console.WriteLine("Podaj nazwę produktu: \n");
        String? prodName = Console.ReadLine();
        Console.WriteLine("Podaj liczbę dostępnych sztuk: \n");
        int quantity = Int32.Parse(Console.ReadLine());

        ProdContext productContext = new ProdContext();
        Product product = new Product { ProductName = prodName, UnitsInStock = quantity };
        productContext.Products.Add(product);

        bool createdNewSupplier = false;
        bool isValidChoice = false;
        Supplier? supplier = null;

        do
        {
            Console.WriteLine("Czy chcesz dodać nowego dostawcę? (tak/nie)");
            string choice = Console.ReadLine();

            switch (choice)
            {
                case "tak":
                    isValidChoice = true;
                    supplier = CreateNewSupplier();
                    createdNewSupplier = true;
                    break;
                case "nie":
                    isValidChoice = true;
                    ShowAllSuppliers(productContext);
                    supplier = FindSupplier(productContext);
                    break;
            }
        }

        while (!isValidChoice);

        product.Supplier = supplier;
        supplier.Products.Add(product); // nowa linia w c)
    }
}

```

```

        if (createdNewSupplier) {
            productContext.Suppliers.Add(supplier);
        }

        productContext.Products.Add(product);
        productContext.SaveChanges();
    }

    private static Supplier CreateNewSupplier()
    {
        Console.WriteLine("\n\nuzupełnij nazwę dostawcy: ");
        string companyName = Console.ReadLine();
        Console.WriteLine("\nUzupełnij miasto: ");
        string city = Console.ReadLine();
        Console.WriteLine("\nUzupełnij ulicę: ");

        string street = Console.ReadLine();

        Supplier supplier = new Supplier
        {
            CompanyName = companyName,
            City = city,
            Street = street
        };
        Console.WriteLine($" \n Utworzono pomyślnie dostawcę: {supplier.CompanyName}");
        return supplier;
    }

    private static Supplier FindSupplier(ProdContext productContext) {
        Console.WriteLine("\n Podaj ID istniejącego dostawcy: ");
        int id = Int32.Parse(Console.ReadLine());

        var query = from sup in productContext.Suppliers
                     where sup.SupplierID == id
                     select sup;
        return query.FirstOrDefault();
    }

    private static void ShowAllSuppliers(ProdContext productContext)
    {
        Console.WriteLine("\n Lista wszystkich istniejących dostawców: ");
    }

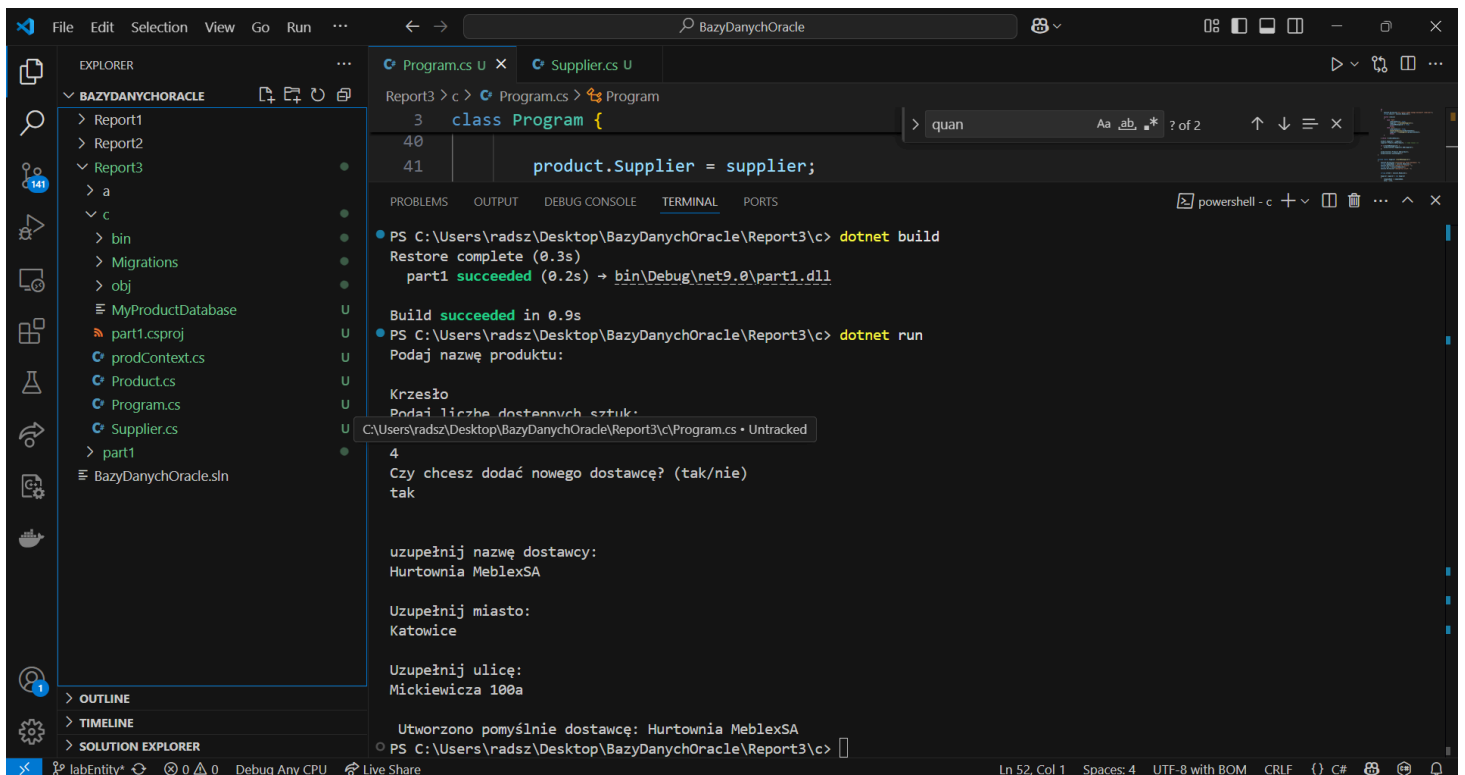
```

```

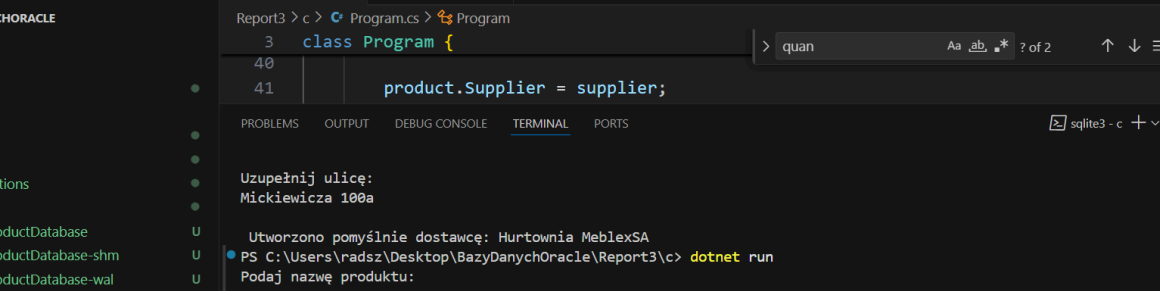
foreach (Supplier supplier in productContext.Suppliers)
{
    Console.WriteLine($"[{supplier.SupplierID}] {supplier.CompanyName}");
}
}
}

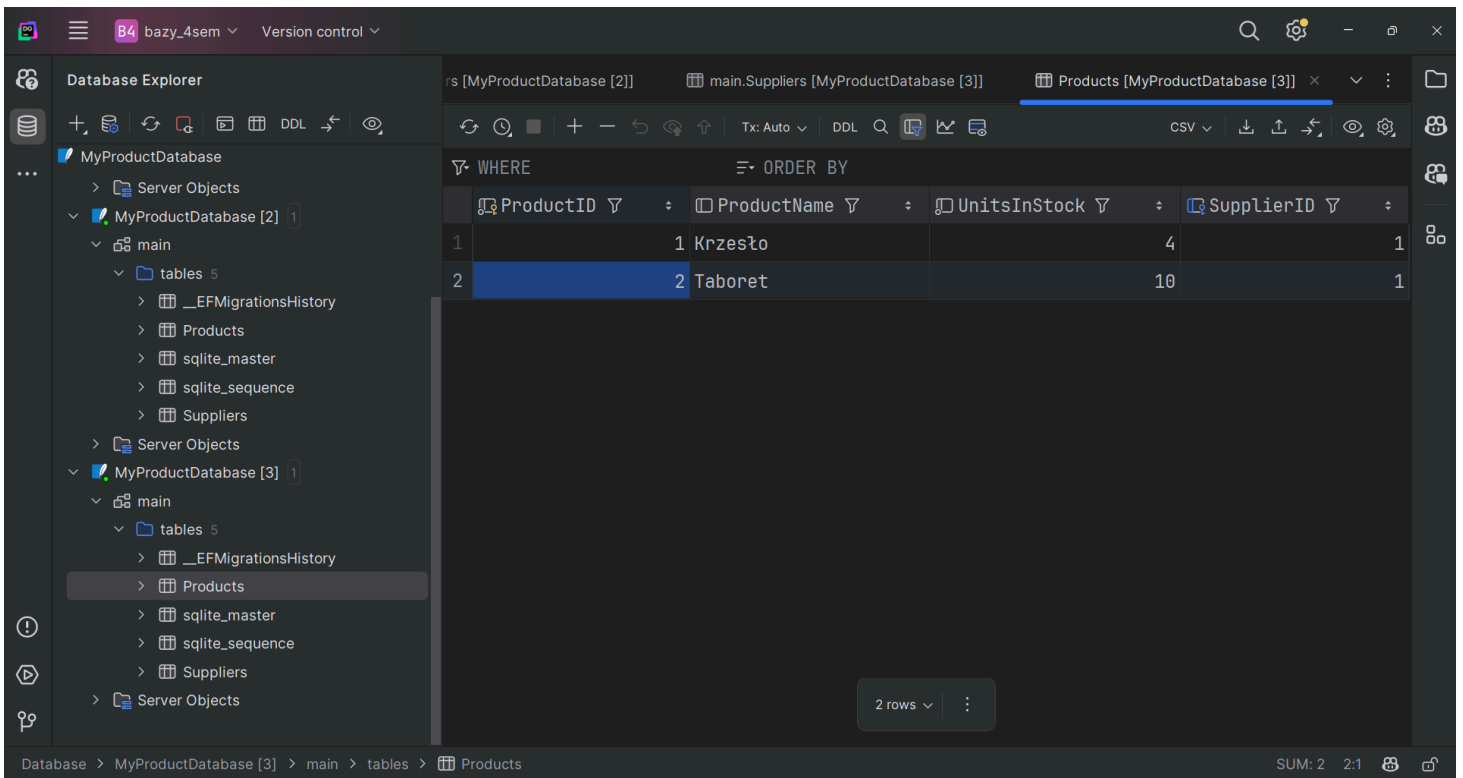
```

Poniżej zamieszczamy zrzuty ekranu realizacji przez nas powyższego kodu wraz z uzyskanym rezultatem



The screenshot displays the SQL Developer interface. On the left, the Database Explorer shows the 'MyProductDatabase [3]' schema with a 'main' tablespace containing 'Suppliers' and 'Products' tables. The SQL Editor on the right shows a query result for the 'Suppliers' table with columns: SupplierID, CompanyName, Street, and City. The result shows one row: 1, Hurtownia MeblexSA, Mickiewicza 100a, Katowice.





Zadanie d:

Realizację zadania d) przedstawia kod poniżej:

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

    public List<InvoiceProduct>? InvoiceProducts { get; set; }
}
```

```
using System;
using System.Collections.Generic;

public class Invoice
{
    public int InvoiceID { get; set; }
    public DateTime Date { get; set; }

    public List<InvoiceProduct>? InvoiceProducts { get; set; }
}

public class InvoiceProduct
{
    public int ProductID { get; set; }
    public Product Product { get; set; }

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

    public int Quantity { get; set; }
}
```

```

using Microsoft.EntityFrameworkCore;

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

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

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

        modelBuilder.Entity<InvoiceProduct>()
            .HasOne(ip => ip.Product)
            .WithMany(p => p.InvoiceProducts)
            .HasForeignKey(ip => ip.ProductID);

        modelBuilder.Entity<InvoiceProduct>()
            .HasOne(ip => ip.Invoice)
            .WithMany(i => i.InvoiceProducts)
            .HasForeignKey(ip => ip.InvoiceID);
    }
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;

class Program
{
    static void Main()
    {
        using var context = new ProdContext();

        Console.WriteLine("Wybierz akcję:");
        Console.WriteLine("1 - Dodaj produkt i przypisz do faktury");
        Console.WriteLine("2 - Pokaż produkty z faktury");
        Console.WriteLine("3 - Pokaż faktury produktu");

        string? choice = Console.ReadLine();

        switch (choice)
        {
            case "1":
                AddProductToInvoice(context);
                break;
            case "2":
                ShowProductsOfInvoice(context);
                break;
            case "3":
                ShowInvoicesOfProduct(context);
                break;
            default:
                Console.WriteLine("Nieznana opcja.");
                break;
        }
    }

    static void AddProductToInvoice(ProdContext context)
    {
        Console.WriteLine("Podaj nazwę produktu:");
        string? name = Console.ReadLine();

        Console.WriteLine("Podaj ilość dostępnych sztuk:");
        int stock = int.Parse(Console.ReadLine());
    }
}

```



```

Product product = new Product { ProductName = name, UnitsInStock = stock };
context.Products.Add(product);
context.SaveChanges();

Console.WriteLine("Czy chcesz przypisać do istniejącej faktury? (tak/nie)");
string? reuse = Console.ReadLine();

int invoiceId;
if (reuse == "tak")
{
    Console.WriteLine("Podaj ID faktury:");
    invoiceId = int.Parse(Console.ReadLine());
}
else
{
    Invoice invoice = new Invoice { Date = DateTime.Now };
    context.Invoices.Add(invoice);
    context.SaveChanges();
    invoiceId = invoice.InvoiceID;
}

Console.WriteLine("Podaj ilość sprzedanych sztuk w tej fakturze:");
int quantity = int.Parse(Console.ReadLine());

InvoiceProduct ip = new InvoiceProduct
{
    ProductID = product.ProductID,
    InvoiceID = invoiceId,
    Quantity = quantity
};

context.InvoiceProducts.Add(ip);
context.SaveChanges();

Console.WriteLine($"Dodano produkt do faktury {invoiceId} (ilość: {quantity}).");
}

static void ShowProductsOfInvoice(ProdContext context)
{
    Console.WriteLine("Podaj ID faktury:");
    int invoiceId = int.Parse(Console.ReadLine());

    var products = context.InvoiceProducts

```

```

        .Include(ip => ip.Product)
        .Where(ip => ip.InvoiceID == invoiceId)
        .ToList();

Console.WriteLine($"Produkty z faktury {invoiceId}:");
foreach (var ip in products)
{
    Console.WriteLine($"- {ip.Product.ProductName}, sprzedano: {ip.Quantity} szt.");
}

static void ShowInvoicesOfProduct(ProdContext context)
{
    Console.WriteLine("Podaj ID produktu:");
    int productId = int.Parse(Console.ReadLine());

    var invoices = context.InvoiceProducts
        .Include(ip => ip.Invoice)
        .Where(ip => ip.ProductID == productId)
        .ToList();

    Console.WriteLine($"Faktury z udziałem produktu {productId}:");
    foreach (var ip in invoices)
    {
        Console.WriteLine($"- Faktura ID: {ip.InvoiceID}, data: {ip.Invoice.Date}, ilość: {ip.Quantity}");
    }
}

```

Poniżej zamieszczamy zrzuty ekranu realizacji przez nas powyższego kodu wraz z uzyskanym rezultatem:

```
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\d> dotnet run
Wybierz akcję:
1 - Dodaj produkt i przypisz do faktury
2 - Pokaż produkty z faktury
3 - Pokaż faktury produktu
1
Podaj nazwę produktu:
Długopis
Podaj ilość dostępnych sztuk:
5
Czy chcesz przypisać do istniejącej faktury? (tak/nie)
nie
Podaj ilość sprzedanych sztuk w tej fakturze:
5
Dodano produkt do faktury 1 (ilość: 5).
```

```
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\d> dotnet run
Wybierz akcję:
1 - Dodaj produkt i przypisz do faktury
2 - Pokaż produkty z faktury
3 - Pokaż faktury produktu
1
Podaj nazwę produktu:
Zeszyt
Podaj ilość dostępnych sztuk:
10
Czy chcesz przypisać do istniejącej faktury? (tak/nie)
nie
Podaj ilość sprzedanych sztuk w tej fakturze:
6
Dodano produkt do faktury 2 (ilość: 6).
```

```
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\d> dotnet run
Wybierz akcję:
1 - Dodaj produkt i przypisz do faktury
2 - Pokaż produkty z faktury
3 - Pokaż faktury produktu
1
Podaj nazwę produktu:
Ołówek
Podaj ilość dostępnych sztuk:
3
Czy chcesz przypisać do istniejącej faktury? (tak/nie)
nie
Podaj ilość sprzedanych sztuk w tej fakturze:
2
Dodano produkt do faktury 3 (ilość: 2).
```

```
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\d> dotnet run
Wybierz akcję:
1 - Dodaj produkt i przypisz do faktury
2 - Pokaż produkty z faktury
3 - Pokaż faktury produktu
2
Podaj ID faktury:
2
Produkty z faktury 2:
- Zeszyt, sprzedano: 6 szt.
```

```
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\d> dotnet run
Wybierz akcję:
1 - Dodaj produkt i przypisz do faktury
2 - Pokaż produkty z faktury
3 - Pokaż faktury produktu
3
Podaj ID produktu:
1
Faktury z udziałem produktu 1:
- Faktura ID: 1, data: 02.06.2025 19:01:22, ilość: 5
```

```

PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\d> sqlite3 MyProductDatabase_d.db
SQLite version 3.44.2 2023-11-24 11:41:44 (UTF-16 console I/O)
Enter ".help" for usage hints.
sqlite> .schema
CREATE TABLE IF NOT EXISTS "__EFMigrationsHistory" (
  "MigrationId" TEXT NOT NULL CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY,
  "ProductVersion" TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS "Invoices" (
  "InvoiceID" INTEGER NOT NULL CONSTRAINT "PK_Invoices" PRIMARY KEY AUTOINCREMENT,
  "Date" TEXT NOT NULL
);
CREATE TABLE sqlite_sequence(name,seq);
CREATE TABLE IF NOT EXISTS "Products" (
  "ProductID" INTEGER NOT NULL CONSTRAINT "PK_Products" PRIMARY KEY AUTOINCREMENT,
  "ProductName" TEXT NULL,
  "UnitsInStock" INTEGER NOT NULL
);
CREATE TABLE IF NOT EXISTS "InvoiceProducts" (
  "ProductID" INTEGER NOT NULL,
  "InvoiceID" INTEGER NOT NULL,
  "Quantity" INTEGER NOT 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
);
CREATE INDEX "IX_InvoiceProducts_InvoiceID" ON "InvoiceProducts" ("InvoiceID");

```

```

sqlite> SELECT * FROM Products;
1|Długopis|5
2|Zeszyt|10
3|Ołówek|3
sqlite> SELECT * FROM Invoices;
1|2025-06-02 19:01:22.6038621
2|2025-06-02 19:02:09.9577467
3|2025-06-02 19:02:50.7877646
sqlite> SELECT * FROM InvoiceProducts;
1|1|5
2|2|6
3|3|2

```

Zadanie e: Wprowadzenie do modelu hierarchii dziedziczenia używając strategii Table-Per-Hierarchy

Realizację zadania e) przedstawia kod poniżej:

```

public 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;
}

using Microsoft.EntityFrameworkCore;

public class CompanyContext : DbContext
{
    public DbSet<Company>? Companies { get; set; }
    public DbSet<Supplier>? Suppliers { get; set; }
    public DbSet<Customer>? Customers { get; set; }

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

public static class CompanyType
{
    public const string CUSTOMER = "customer";
    public const string SUPPLIER = "supplier";

    public static List<string> COMPANY_TYPES = new()
    {
        CUSTOMER,
        SUPPLIER
    };
}

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

```

```

using System;

class Program
{

    static void Main()
    {

        using CompanyContext companyContext = new();
        Console.WriteLine("Wybierz co chcesz zrobić (add/display): \n");
        String? action = Console.ReadLine();

        switch (action)
        {
            case "add":
                AddCompany(companyContext);
                break;
            case "display":
                DisplayCompanies(companyContext);
                break;
        }
    }

    private static void AddCompany(CompanyContext companyContext)
    {
        while (true)
        {
            Console.WriteLine("\nWybierz rodzaj firmy, którą chciałbyś dodać: \n");
            foreach (string companyType in CompanyType.COMPANY_TYPES) Console.WriteLine($"{companyType} ");
            String? type = Console.ReadLine();
            Console.WriteLine("Podaj nazwę firmy: \n");
            String? companyName = Console.ReadLine();
            Console.WriteLine("Podaj miasto: \n");
            String? city = Console.ReadLine();
            Console.WriteLine("Podaj ulicę i nr: \n");
            String? street = Console.ReadLine();
            Console.WriteLine("Podaj kod pocztowy: \n");
            String? postalCode = Console.ReadLine();

            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 Customer CreateCustomer(string companyName, string street, string city, string postalCode)
{
    Console.WriteLine("Podaj wartość zniżki (0.0-1.0)\n ");

    float discount = float.Parse(Console.ReadLine());

    return new Customer
    {
        CompanyName = companyName,
        Street = street,
        City = city,
        ZipCode = postalCode,
        Discount = discount
    };
}

```

```

private static Supplier CreateSupplier(string companyName, string city, string street, string postalCode)
{
    Console.WriteLine("Podaj numer konta bankowego: ");
    string bankAccountNumber = Console.ReadLine();

    return new Supplier
    {
        CompanyName = companyName,
        Street = street,
        City = city,
        ZipCode = postalCode,
        BankAccountNumber = bankAccountNumber
    };
}

```



```

}

private static void DisplayCompanies(CompanyContext companyContext)
{
    Console.WriteLine("\nPodaj typ firm, które chcesz wyświetlić:\n -all");
    foreach (string type in CompanyType.COMPANY_TYPES) Console.WriteLine($"-{type}");
    string companyType = Console.ReadLine();

    switch (companyType) {
        case "all":
            Console.WriteLine("Lista wszystkich klientów (firm):");
            foreach (Company company in companyContext.Companies){
                Console.WriteLine(company);
            }

            break;

        case CompanyType.SUPPLIER:
            Console.WriteLine("Lista wszystkich dostawców (firm):");
            foreach (Supplier supplier in companyContext.Suppliers){
                Console.WriteLine(supplier);
            }

            break;

        case CompanyType.CUSTOMER:
            Console.WriteLine("Lista wszystkich klientów (firm):");
            foreach (Customer customer in companyContext.Customers){
                Console.WriteLine(customer);
            }

            break;
    }
}
}

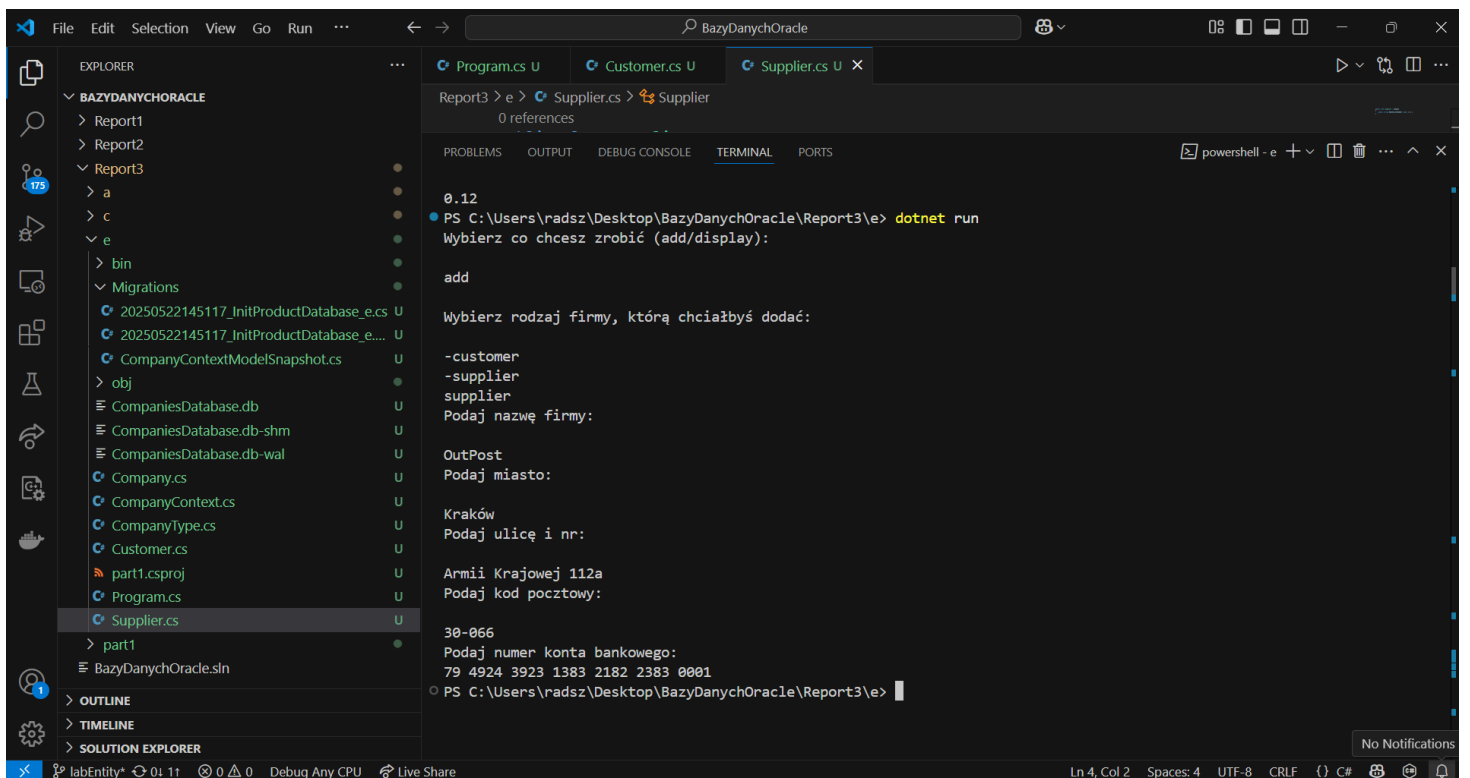
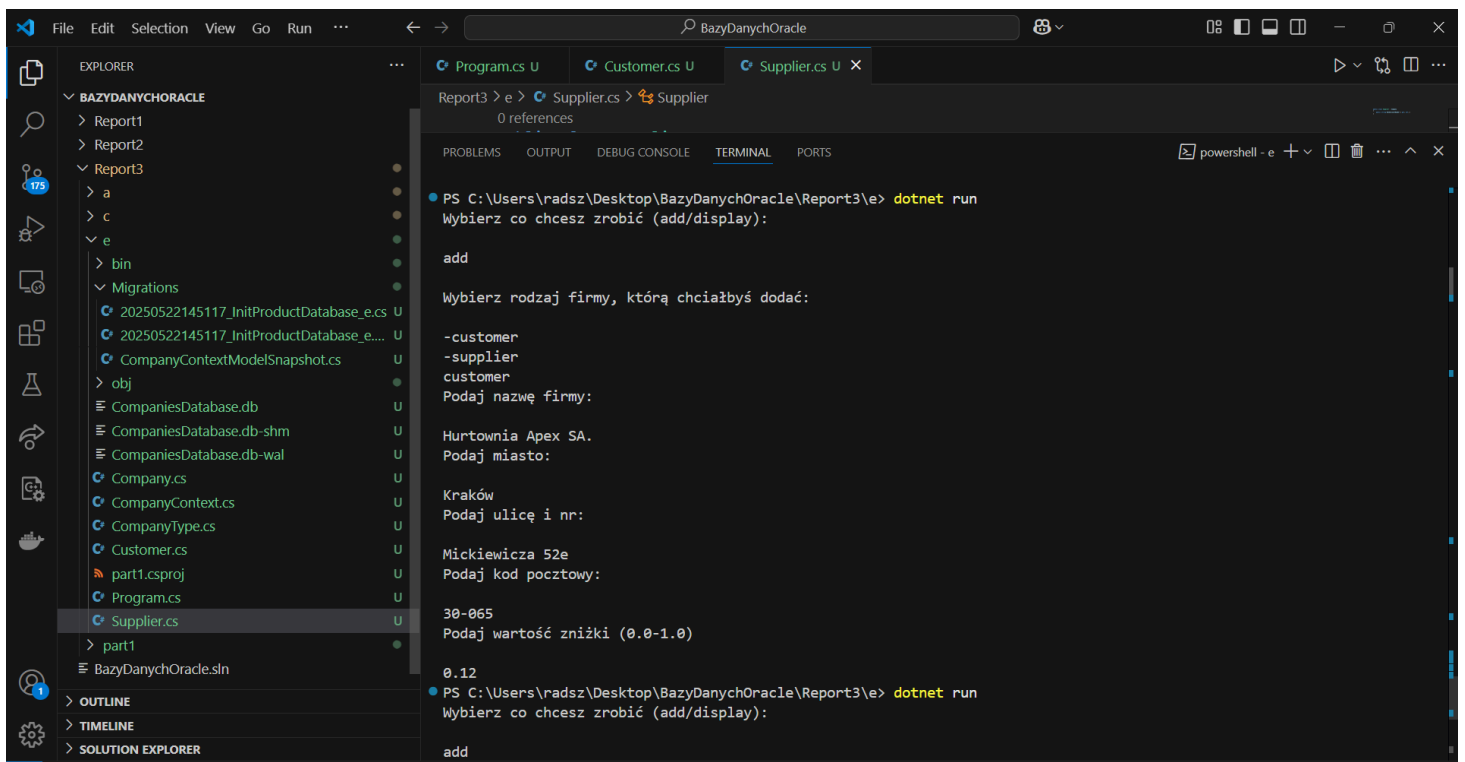
```

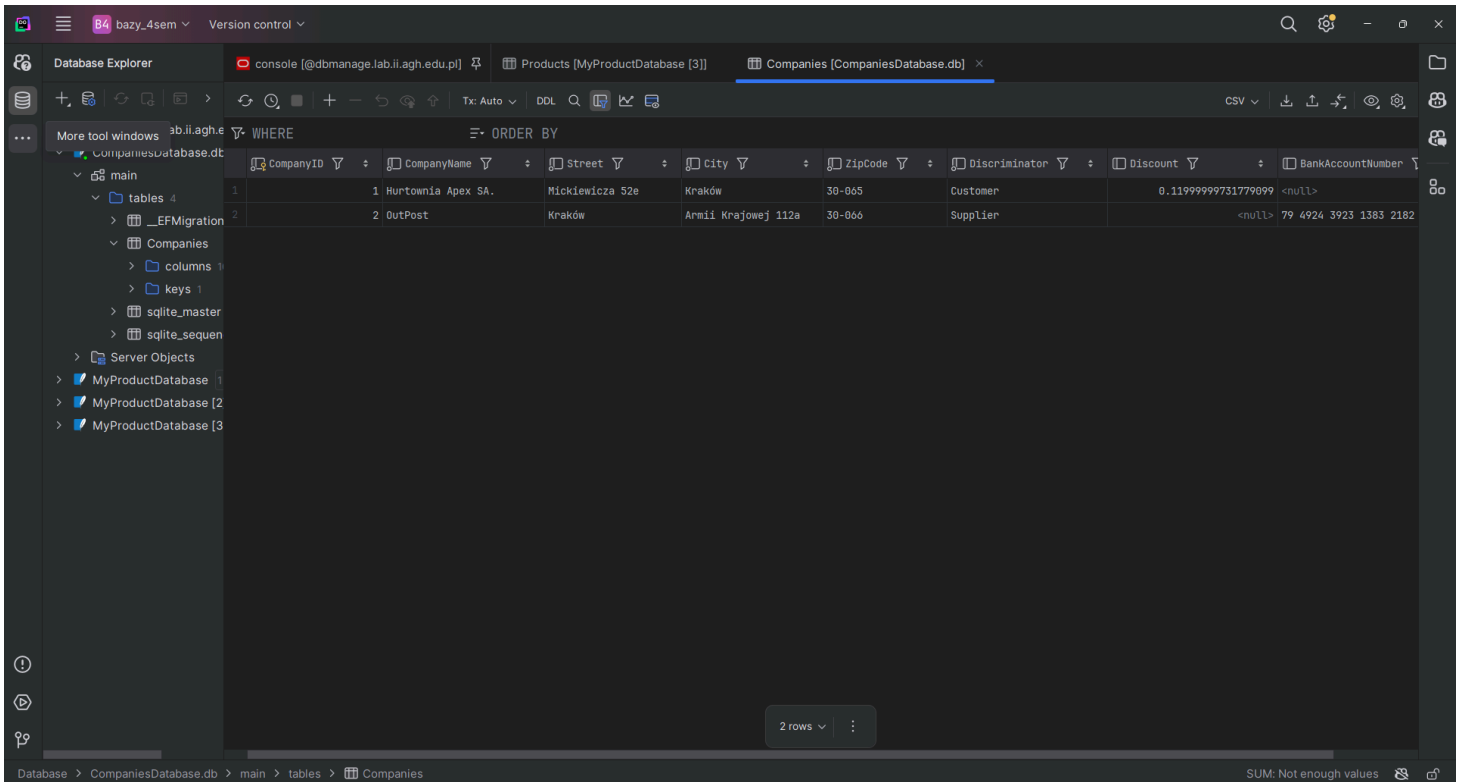
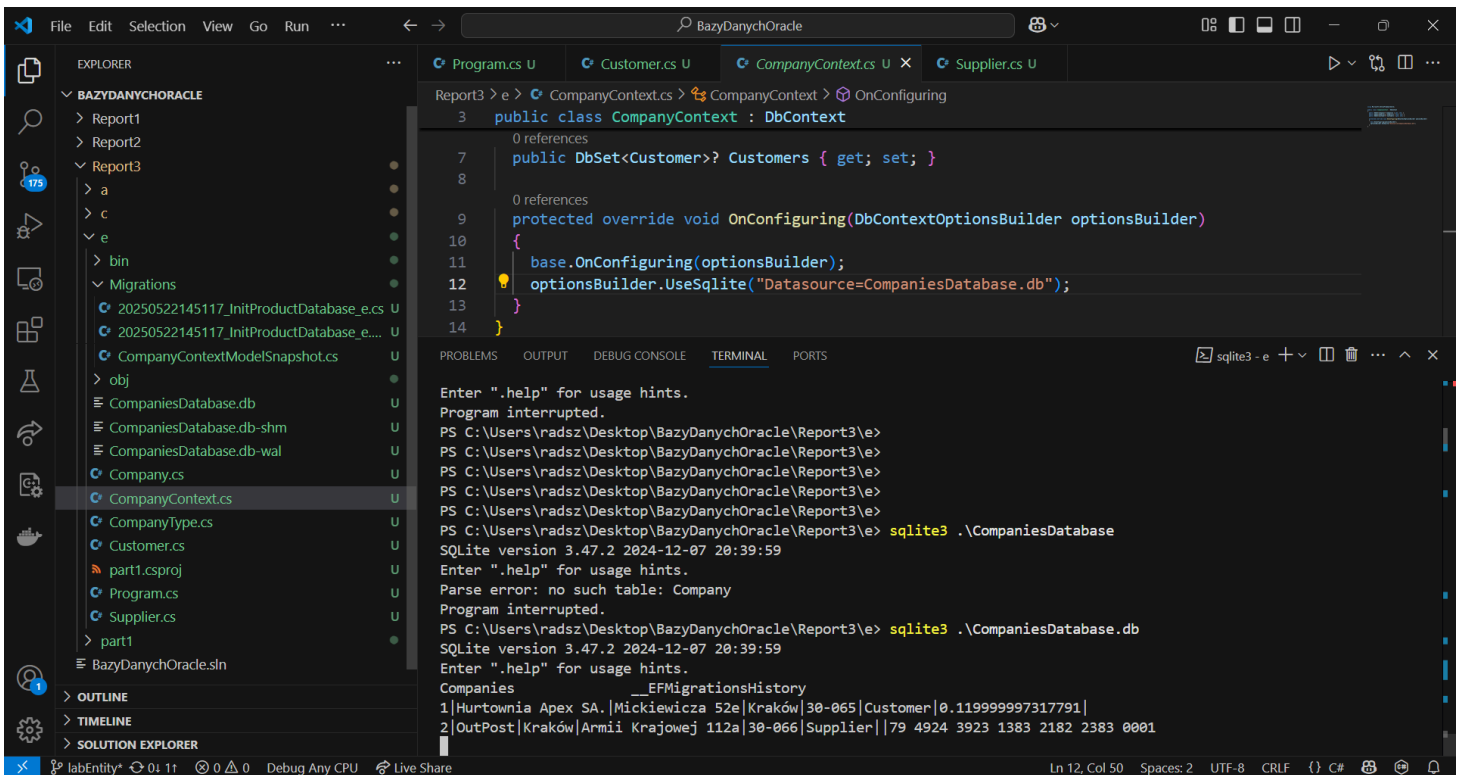
Poniżej zamieszczamy zrzuty ekranu realizacji przez nas powyższego kodu wraz z uzyskanym rezultatem:

```

public class Supplier : Company
{
    public string BankAccountNumber { get; set; }
}

```





Zadanie f:

Realizację zadania f) przedstawia kod poniżej:

```

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; }
}

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

public class Supplier : Company
{
    public string BankAccountNumber { get; set; }
}

using Microsoft.EntityFrameworkCore;

public class CompanyContext : DbContext
{
    public DbSet<Company> Companies { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Customer> Customers { get; set; }

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

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

```

```
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        using var context = new CompanyContext();

        Console.WriteLine("Dodaj firmę jako: supplier czy customer?");
        string? type = Console.ReadLine();

        Console.WriteLine("Podaj nazwę firmy:");
        string name = Console.ReadLine();

        Console.WriteLine("Ulica:");
        string street = Console.ReadLine();

        Console.WriteLine("Miasto:");
        string city = Console.ReadLine();

        Console.WriteLine("Kod pocztowy:");
        string zip = Console.ReadLine();

        if (type == "supplier")
        {
            Console.WriteLine("Podaj numer konta bankowego:");
            string account = Console.ReadLine();

            Supplier supplier = new Supplier
            {
                CompanyName = name,
                Street = street,
                City = city,
                ZipCode = zip,
                BankAccountNumber = account
            };

            context.Suppliers.Add(supplier);
            context.SaveChanges();

            Console.WriteLine("\nDostawcy:");
            foreach (var s in context.Suppliers)
```

```

        {
            Console.WriteLine($"[{s.CompanyID}] {s.CompanyName}, {s.City}, {s.Street}, {s.Z:
        }
    }
else if (type == "customer")
{
    Console.WriteLine("Podaj wartość rabatu (np. 0.1 dla 10%:");
    float discount = float.Parse(Console.ReadLine());

    Customer customer = new Customer
    {
        CompanyName = name,
        Street = street,
        City = city,
        ZipCode = zip,
        Discount = discount
    };

    context.Customers.Add(customer);
    context.SaveChanges();

    Console.WriteLine("\nKlienci:");
    foreach (var c in context.Customers)
    {
        Console.WriteLine($"[{c.CompanyID}] {c.CompanyName}, {c.City}, {c.Street}, {c.Z:
    }
}
else
{
    Console.WriteLine("Nieznany typ firmy.");
}
}
}

```

Poniżej zamieszczamy zrzuty ekranu realizacji przez nas powyższego kodu wraz z uzyskanym rezultatem:

```
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\f> dotnet run
Dodaj firmę jako: supplier czy customer?
supplier
Podaj nazwę firmy:
dostawca1
Ulica:
dluga 10
Miasto:
Krakow
Kod pocztowy:
12-345
Podaj numer konta bankowego:
1234567890

Dostawcy:
[1] dostawca1, Krakow, dluga 10, 12-345, Konto: 1234567890
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\f> dotnet run
Dodaj firmę jako: supplier czy customer?
supplier
Podaj nazwę firmy:
dostawca2
Ulica:
krotka 1
Miasto:
Krakow
Kod pocztowy:
12-345
Podaj numer konta bankowego:
0987654321

Dostawcy:
[1] dostawca1, Krakow, dluga 10, 12-345, Konto: 1234567890
[2] dostawca2, Krakow, krotka 1, 12-345, Konto: 0987654321
```

```
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\f> dotnet run
Dodaj firmę jako: supplier czy customer?
customer
Podaj nazwę firmy:
klient1
Ulica:
krotka 5
Miasto:
Krakow
Kod pocztowy:
12-345
Podaj wartość rabatu (np. 0.1 dla 10%):
0,1

Klienci:
[3] klient1, Krakow, krotka 5, 12-345, Rabat: 10%
PS C:\Users\kacpe\Documents\Bazy\Raporty\BazyOracle-NoSQL\Report3\f> dotnet run
Dodaj firmę jako: supplier czy customer?
customer
Podaj nazwę firmy:
klient2
Ulica:
dluga 8
Miasto:
Krakow
Kod pocztowy:
12-345
Podaj wartość rabatu (np. 0.1 dla 10%):
0,05

Klienci:
[3] klient1, Krakow, krotka 5, 12-345, Rabat: 10%
[4] klient2, Krakow, dluga 8, 12-345, Rabat: 5%
```



```

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

```

```

sqlite> SELECT * FROM Companies;
1|dostawca1|dluga 10|Krakow|12-345
2|dostawca2|krotka 1|Krakow|12-345
3|klient1|krotka 5|Krakow|12-345
4|klient2|dluga 8|Krakow|12-345
sqlite> SELECT * FROM Companies WHERE Discriminator = 'Customer';
Parse error: no such column: Discriminator
      SELECT * FROM Companies WHERE Discriminator = 'Customer';
                        error here ---^
sqlite> SELECT * FROM Suppliers;
1|1234567890
2|0987654321
sqlite> SELECT * FROM Customers;
3|0.100000001490116
4|0.0500000007450581

```

g) Porównanie strategii modelowania dziedziczenia w EF Core: Table-Per-Hierarchy (TPH) vs Table-Per-Type (TPT)

W zadaniach **e** i **f** zaprezentowaliśmy dwie różne strategie modelowania dziedziczenia w Entity Framework Core:

1. Table-Per-Hierarchy (TPH) – zadanie e)

- **Opis:**

- Wszystkie klasy dziedziczące z klasy `Company` są przechowywane w jednej wspólnej tabeli `Companies`.
- Rozróżnienie typów (`Customer` , `Supplier`) odbywa się za pomocą kolumny dyskryminatora.
- W tabeli występują kolumny specyficzne dla wszystkich typów dziedziczących (np. `Discount` , `BankAccountNumber`), nawet jeśli część z nich pozostaje pusta (null).

- **Zalety:**

- Lepsza wydajność odczytu – jeden `SELECT` wystarczy do pobrania wszystkich firm bez potrzeby `JOIN`.
- Prostota struktury bazy danych – tylko jedna tabela.
- Łatwiejsze zarządzanie, gdy mamy małą liczbę typów dziedziczenia

- **Wady:**

- Mniejsza zgodność z zasadami normalizacji bazy danych.
- Trudniejsza kontrola integralności danych – kolumny mogą być null dla niektórych typów.
- Słaba czytelność schematu przy większej liczbie klas pochodnych.

2. Table-Per-Type (TPT) – zadanie f)

- **Opis:**

- Dla każdej klasy dziedziczącej (`Customer` , `Supplier`) tworzona jest osobna tabela (`Customers` , `Suppliers`), która zawiera tylko swoje unikalne kolumny.
- Tabela `Companies` zawiera wspólne dane bazowej klasy abstrakcyjnej.
- Pobranie pełnych danych konkretnej firmy wymaga złączenia (`JOIN`) z tabelą bazową.

- **Zalety:**

- Lepsza normalizacja danych – dane są uporządkowane i logicznie rozdzielone.
- Łatwiejsza kontrola typów i spójności danych w tabelach.
- Przezroczystość struktury – każda tabela zawiera tylko istotne dane dla danej klasy.

- **Wady:**

- Większa złożoność zapytań (JOIN), co może obniżać wydajność.
- Większy koszt utrzymania przy częstych migracjach i zmianach schematu.
- Dłuższy czas zapisu i odczytu przy dużej liczbie rekordów i typów.