

Bazy Danych

Entity Framework

Autorzy

Grupa 8 - [Śr 13:15 , A]

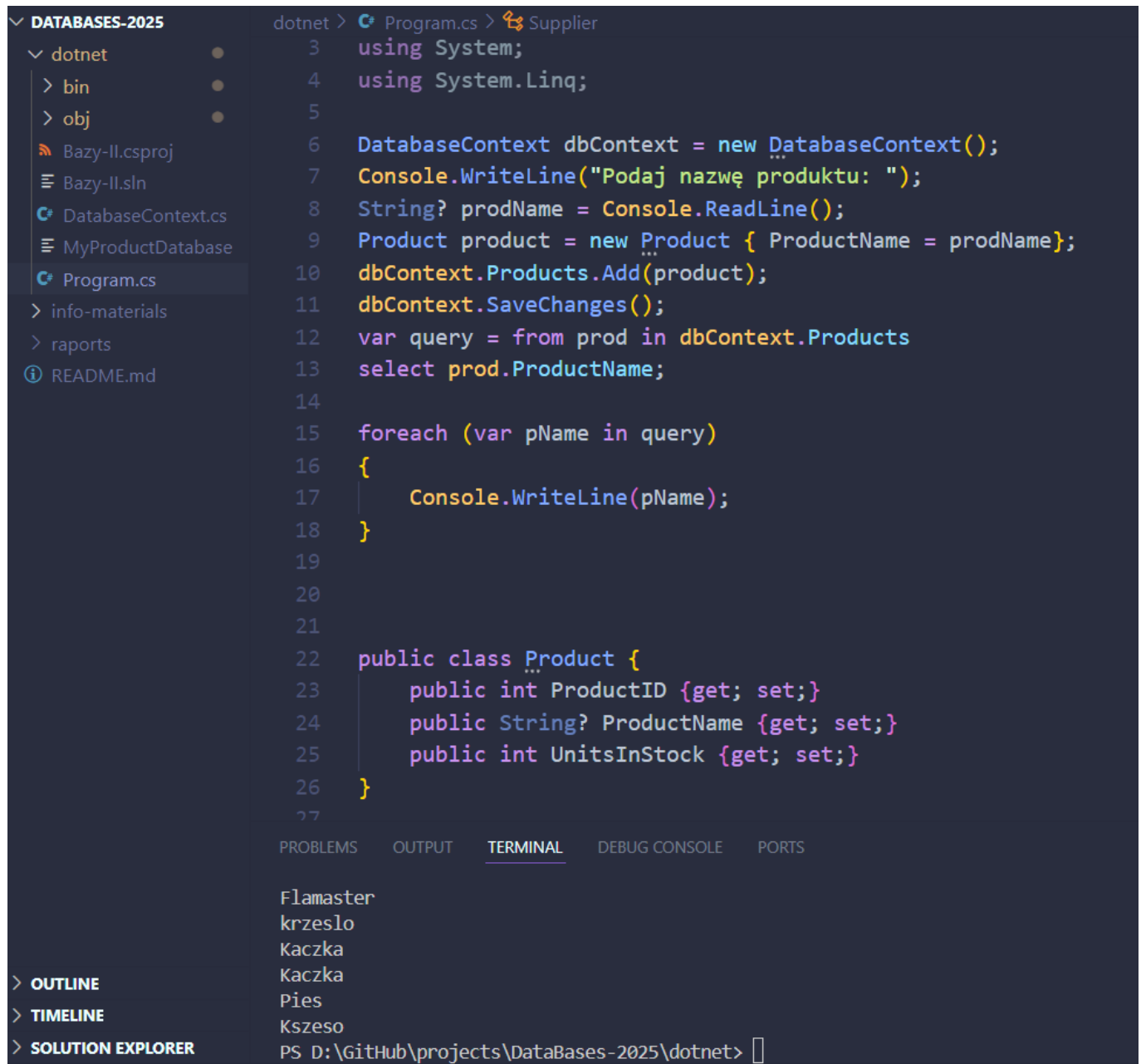
- Bartosz Ludwin
- Mateusz Pawliczek
- Filip Malejki

Na zajęciach udało się zrealizować **Część I** oraz podpunkty **a** i **b** z **Części II**. Reszta zadań została zrealizowana po zajęciach.

Część I:

```
1 using Microsoft.EntityFrameworkCore;
2 public class ProdContext : DbContext
3 {
4     public DbSet<Product> Products { get; set; }
5
6     protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
7     {
8         base.OnConfiguring(optionsBuilder);
9         optionsBuilder.UseSqlite("DataSource=MyProductDatabase");
10    }
11 }
12 }
```

```
1 // See https://aka.ms/new-console-template for more information
2
3 using System;
4 using System.Linq;
5
6 ProdContext prodContext = new ProdContext();
7 Console.WriteLine("Podaj nazwę produktu: ");
8 String? prodName = Console.ReadLine();
9 Product product = new Product { ProductName = prodName};
10 prodContext.Products.Add(product);
11 prodContext.SaveChanges();
12 var query = from prod in prodContext.Products
13 select prod.ProductName;
14
15 foreach (var pName in query)
16 {
17     Console.WriteLine(pName);
18 }
19
20
21
22 public class Product {
23     public int ProductID {get; set;}
24     public String? ProductName {get; set;}
25     public int UnitsInStock {get; set;}
26 }
```



```
dotnet > Program.cs > Supplier
3  using System;
4  using System.Linq;
5
6  DbContext dbContext = new DbContext();
7  Console.WriteLine("Podaj nazwę produktu: ");
8  String? prodName = Console.ReadLine();
9  Product product = new Product { ProductName = prodName};
10 dbContext.Products.Add(product);
11 dbContext.SaveChanges();
12 var query = from prod in dbContext.Products
13 select prod.ProductName;
14
15 foreach (var pName in query)
16 {
17     Console.WriteLine(pName);
18 }
19
20
21
22 public class Product {
23     public int ProductID {get; set;}
24     public String? ProductName {get; set;}
25     public int UnitsInStock {get; set;}
26 }
27
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE PORTS

Flamaster
krzeslo
Kaczka
Kaczka
Pies
Kszeso
PS D:\GitHub\projects\DataBases-2025\dotnet>

Część II:

A) Tworzenie tabel

```

3 3 references
  public class Product {
4      0 references
      public int ProductID {get; set;}
5      2 references
      public String ProductName { get; set; }
6      0 references
      public int UnitsOnStock {get; set;}
7
8      1 reference
      public int SupplierID { get; set; }
9      5 references
      public Supplier? Supplier { get; set; }
10 }
11
12 4 references
  public class Supplier {
13      0 references
      public int SupplierID {get; set;}
14      4 references
      public String CompanyName {get; set;}
15      0 references
      public String? Street {get; set;}
16      0 references
      public String? City {get; set;}
17
18      1 reference
      public ICollection<Product> Products { get; set; } = new List<Product>();
19 }
20

```

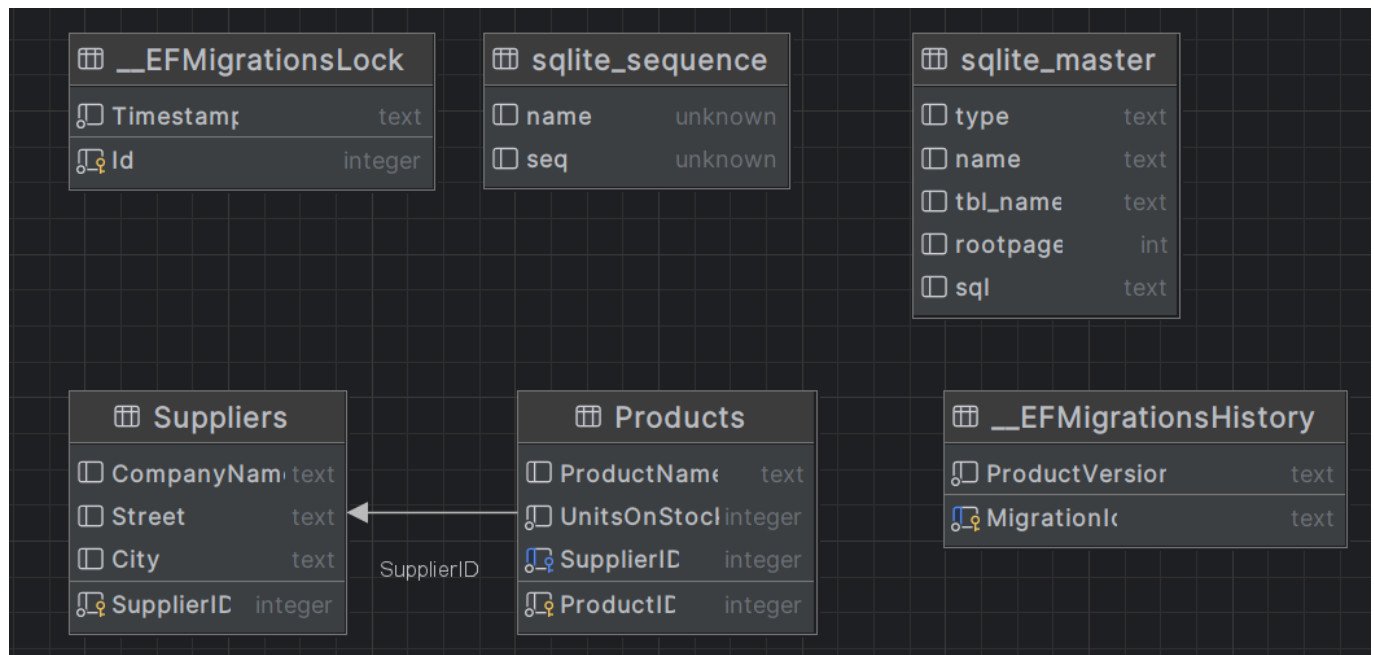
Dodanie relacji:

```

1 using Microsoft.EntityFrameworkCore;
2
3 5 references
  public class DatabaseContext : DbContext
4  {
5      1 reference
      public DbSet<Product> Products { get; set; }
6      2 references
      public DbSet<Supplier> Suppliers { get; set; }
7
8      0 references
      protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
9      {
10          base.OnConfiguring(optionsBuilder);
11          optionsBuilder.UseSqlite("Datasource=MyProductDatabase");
12      }
13
14      0 references
      protected override void OnModelCreating(ModelBuilder modelBuilder)
15      {
16          modelBuilder.Entity<Product>()
17              .HasOne(p => p.Supplier)
18              .WithMany(s => s.Products)
19              .HasForeignKey(p => p.SupplierID);
20      }
21 }

```

Diagram bazy (wygenerowany w DataGrip):



Dodanie nowego dostawcy (podpunkt i)

```

4
5 dbContext.Suppliers.Add(
6     new Supplier()
7     {
8         CompanyName = "DOGE CORP"
9     }
10 );
11

```

Dodanie stworzonego dostawcy do wcześniejszego produktu (podpunkt ii)

```

1 using Microsoft.EntityFrameworkCore;
2
3 using DbContext dbContext = new DbContext();
4
5 var product = dbContext.Products
6     .Include(p => p.Supplier) // warto dołączyć, jeśli korzystamy z nawigacji
7     .SingleOrDefault(p => p.ProductName == "Kaczka");
8
9 var supplier = dbContext.Suppliers.SingleOrDefault(s => s.CompanyName == "ACME Corp");
10
11 if (product != null)
12     product.Supplier = supplier;
13 dbContext.SaveChanges();
14

```

Wynik przed:

```

C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet\Program.cs
Laptop, Dostawca: ACME Corp
Mouse, Dostawca: ACME Corp
Krzesło, Dostawca:
Kaczka, Dostawca:
Stół, Dostawca:

```

Wynik po:

```
C:\Users\Bartek\Documents\GitHub\DataBases-2025\dc
Laptop, Dostawca: ACME Corp
Mouse, Dostawca: ACME Corp
Krzesło, Dostawca:
Kaczka, Dostawca: ACME Corp
Stół, Dostawca:

Companies:
ACME Corp
```

B) Zmieniamy relację

```
1 0 references
2 protected override void OnModelCreating(ModelBuilder modelBuilder)
3 {
4     modelBuilder.Entity<Supplier>()
5         .HasMany(s => s.Products)
6         .WithOne(p => p.Supplier)
7         .HasForeignKey(p => p.SupplierID);
8 }
9 }
```

Dodajemy kilka produktów do nowego dostawcy

```
4
5 List<Product> newProducts = new List<Product>()
6 {
7     new Product() { ProductName = "Mop"},
8     new Product() { ProductName = "Zegarek"},
9     new Product() { ProductName = "Telefon"}
10 };
11
12 var newSupplier = new Supplier()
13 {
14     CompanyName = "Corn Corp",
15     Products = newProducts
16 };
17 dbContext.Suppliers.Add(newSupplier);
18 dbContext.SaveChanges();
19
```

Wynik przed:

```
C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet\Prog
Laptop, Dostawca: ACME Corp
Mouse, Dostawca: ACME Corp
Kaczka, Dostawca: ACME Corp
Mop, Dostawca: Corn Corp
Zegarek, Dostawca: Corn Corp
Telefon, Dostawca: Corn Corp

Companies:
ACME Corp
DOGE CORP
Corn Corp
PS C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet>
```

Wynik po:

```
C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet\Program.cs(28,41):
Laptop, Dostawca: ACME Corp
Mouse, Dostawca: ACME Corp
Krzesło, Dostawca:
Kaczka, Dostawca: ACME Corp
Stół, Dostawca:

Companies:
ACME Corp
DOGE CORP
PS C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet>
```

C) Modyfikacja relacji

```
6 references
20 public class DatabaseContext : DbContext
21 {
22     2 references
23     public DbSet<Product> Products { get; set; }
24     4 references
25     public DbSet<Supplier> Suppliers { get; set; }
26
27     0 references
28     protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
29     {
30         base.OnConfiguring(optionsBuilder);
31         optionsBuilder.UseSqlite("DataSource=MyProductDatabase");
32     }
33
34     0 references
35     protected override void OnModelCreating(ModelBuilder modelBuilder)
36     {
37         modelBuilder.Entity<Supplier>()
38             .HasMany(s => s.Products)
39             .WithOne(p => p.Supplier)
40             .HasForeignKey(p => p.SupplierID);
41     }
42 }
```

Dodajemy kilka produktów do nowego dostawcy

```
4
5  List<Product> newProducts = new List<Product>()
6  {
7      new Product() { ProductName = "Rower"},
8      new Product() { ProductName = "Auto"},
9      new Product() { ProductName = "Klocki Lego"}
10 };
11
12 var newSupplier = new Supplier()
13 {
14     CompanyName = "Myarse Company",
15     Products = newProducts
16 };
17 dbContext.Suppliers.Add(newSupplier);
18 dbContext.SaveChanges();
19
20
```

Wynik po:

```
● C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet\Pro
Rower, Dostawca: Myarse Company
Auto, Dostawca: Myarse Company
Klocki Lego, Dostawca: Myarse Company

Companies:
Myarse Company
PS C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet
```

D) Stworzenie nowych tabel

(dodajemy również pośrednią encję InvoiceProduct, ponieważ inaczej nie moglibyśmy dodać wartości "Quantity")

```

5 references
22 | public class Invoice
23 | {
24 |     4 references
        public int InvoiceID { get; set; }
25 |     4 references
        public required string InvoiceNumber { get; set; }
26 |     3 references
        public DateTime InvoiceDate { get; set; }
27 |     2 references
        public ICollection<InvoiceProduct> InvoiceProducts { get; set; } = new List<InvoiceProduct>();
28 | }
29 |
14 references
30 | public class InvoiceProduct
31 | {
32 |     7 references
        public int InvoiceID { get; set; }
33 |     5 references
        public Invoice Invoice { get; set; } = default!;
34 |     8 references
        public int ProductID { get; set; }
35 |     2 references
        public Product Product { get; set; } = default!;
36 |     5 references
        public int Quantity { get; set; }
37 | }
38 |

```

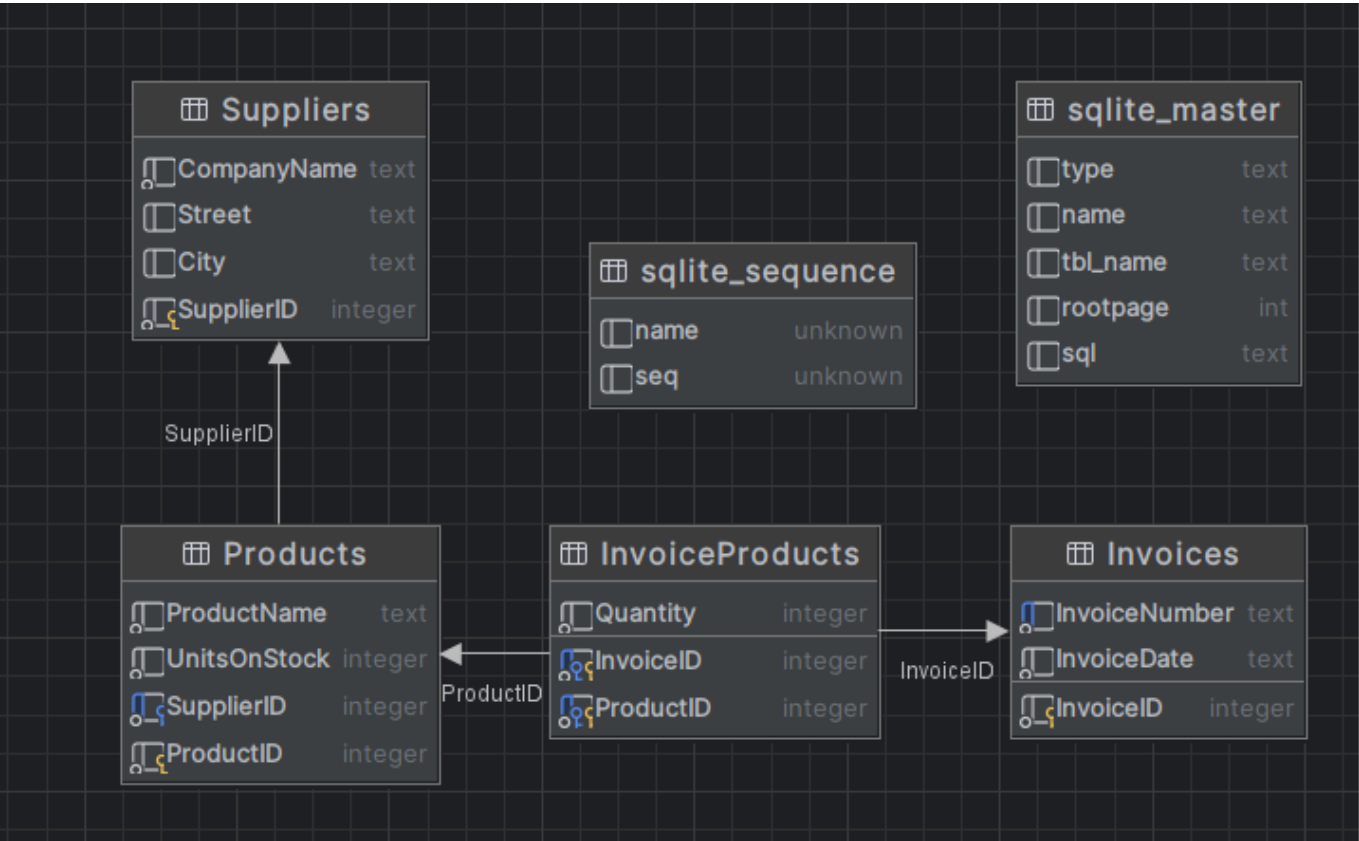
Dodanie relacji:

```

52 |
53 |     0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
54 |     {
55 |         // relacja Supplier - Product
56 |         modelBuilder.Entity<Supplier>()
57 |             .HasMany(s => s.Products)
58 |             .WithOne(p => p.Supplier)
59 |             .HasForeignKey(p => p.SupplierID);
60 |
61 |         // relacja wiele-do-wielu przez encję InvoiceProduct
62 |         modelBuilder.Entity<InvoiceProduct>()
63 |             .HasKey(ip => new { ip.InvoiceID, ip.ProductID });
64 |
65 |         modelBuilder.Entity<InvoiceProduct>()
66 |             .HasOne(ip => ip.Invoice)
67 |             .WithMany(inv => inv.InvoiceProducts)
68 |             .HasForeignKey(ip => ip.InvoiceID);
69 |
70 |         modelBuilder.Entity<InvoiceProduct>()
71 |             .HasOne(ip => ip.Product)
72 |             .WithMany(prod => prod.InvoiceProducts)
73 |             .HasForeignKey(ip => ip.ProductID);
74 |     }
75 | }
76 |
77 |

```


Schemat graficzny:



Dodanie kilku produktów i sprzedanie ich w ramach faktur:

```
19 var produkt1 = new Product { ProductName = "Rower", UnitsOnStock = 10, SupplierID = 1 };
20 var produkt2 = new Product { ProductName = "Auto", UnitsOnStock = 5, SupplierID = 1 };
21 var produkt3 = new Product { ProductName = "Klocki Lego", UnitsOnStock = 20, SupplierID = 1 };
22
23 dbContext.Products.AddRange(produkt1, produkt2, produkt3);
24 dbContext.SaveChanges();
25
26 var faktura1 = new Invoice
27 {
28     InvoiceNumber = "F2025-001",
29     InvoiceDate = new DateTime(2025, 5, 10)
30 };
31 var faktura2 = new Invoice
32 {
33     InvoiceNumber = "F2025-002",
34     InvoiceDate = new DateTime(2025, 5, 15)
35 };
36
37 dbContext.Invoices.AddRange(faktura1, faktura2);
38 dbContext.SaveChanges();
39
40 var sprzedaz1 = new InvoiceProduct
41 {
42     InvoiceID = faktura1.InvoiceID,
43     ProductID = produkt1.ProductID,
44     Quantity = 2
45 };
46 var sprzedaz2 = new InvoiceProduct
47 {
48     InvoiceID = faktura1.InvoiceID,
49     ProductID = produkt2.ProductID,
50     Quantity = 1
51 };
52 var sprzedaz3 = new InvoiceProduct
53 {
54     InvoiceID = faktura2.InvoiceID,
55     ProductID = produkt3.ProductID,
56     Quantity = 3
57 };
58 var sprzedaz4 = new InvoiceProduct
59 {
60     InvoiceID = faktura2.InvoiceID,
61     ProductID = produkt1.ProductID,
62     Quantity = 1
63 };
64
```

Kod wyświetlający produkty sprzedane w ramach wybranej faktury/transakcji:

```
68 string searched = "F2025-001";
69
70 var invoice = dbContext.Invoices
71     .SingleOrDefault(inv => inv.InvoiceNumber == searched);
72
73 if (invoice == null)
74 {
75     Console.WriteLine($"Faktura o numerze {searched} nie istnieje.");
76     return;
77 }
78
79 var positions = dbContext.InvoiceProducts
80     .Where(ip => ip.InvoiceID == invoice.InvoiceID)
81     .Include(ip => ip.Product)
82     .ToList();
83
84 Console.WriteLine($"Produkty sprzedane na fakturze {searched}:");
85 foreach (var ip in positions)
86 {
87     Console.WriteLine($"- {ip.Product.ProductName}, Ilość: {ip.Quantity}");
88 }
89
```

Kod wyświetlający faktury, w ramach których sprzedany został wybrany produkt:

```
91 var searched = "Rower";
92 var wybranyProdukt = dbContext.Products
93     .SingleOrDefault(p => p.ProductName == searched);
94
95 if (wybranyProdukt == null)
96 {
97     Console.WriteLine($"Produkt o nazwie {searched} nie istnieje.");
98     return;
99 }
100
101 var fakturyZProduktami = dbContext.InvoiceProducts
102     .Where(ip => ip.ProductID == wybranyProdukt.ProductID)
103     .Include(ip => ip.Invoice)
104     .ToList();
105
106 Console.WriteLine($"Faktury, w których sprzedano produkt '{searched}':");
107 foreach (var ip in fakturyZProduktami)
108 {
109     Console.WriteLine($"- Numer faktury: {ip.Invoice.InvoiceNumber}, Data: {ip.Invoice.InvoiceDate:d}, Ilość: {ip.Quantity}");
110 }
111
```

Wynik:

```
C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet\Migrations
zerwowane dla języka.
Produkty sprzedane na fakturze F2025-001:
- Rower, Ilość: 2
- Auto, Ilość: 1
Faktury, w których sprzedano produkt 'Rower':
- Numer faktury: F2025-001, Data: 10.05.2025, Ilość: 2
- Numer faktury: F2025-002, Data: 15.05.2025, Ilość: 1
PS C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet>
```

E) Strategia Table-per-Hierarchy

Dodanie nowych tabel:

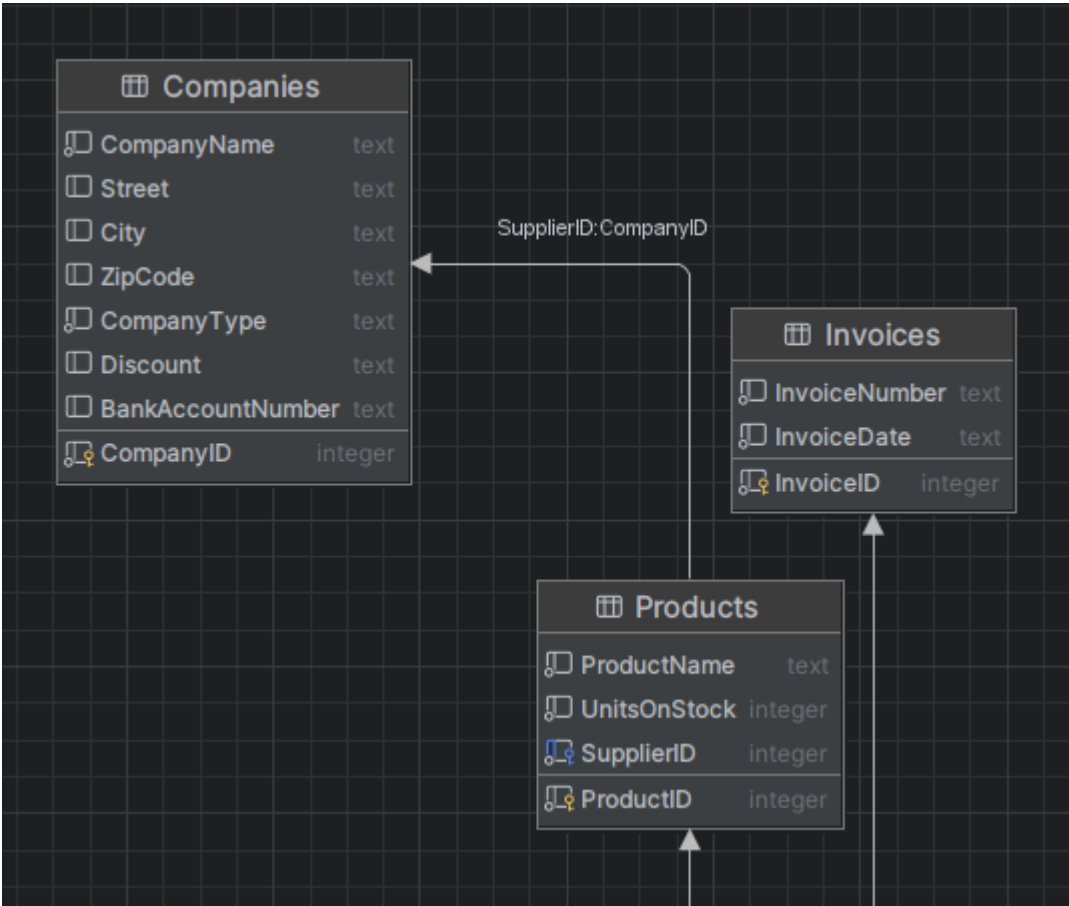
```
13 4 references
13 public abstract class Company
14 {
15     10 references
15     public int CompanyID { get; set; }
16     9 references
16     public required string CompanyName { get; set; }
17     4 references
17     public string? Street { get; set; }
18     8 references
18     public string? City { get; set; }
19     4 references
19     public string? ZipCode { get; set; }
20 }
21
22 7 references
22 public class Supplier : Company
23 {
24     4 references
24     public string? BankAccountNumber { get; set; }
25     1 reference
25     public ICollection<Product> Products { get; set; } = new List<Product>();
26 }
27
28 5 references
28 public class Customer : Company
29 {
30     4 references
30     public decimal Discount { get; set; }
31 }
32
```

Dodanie relacji między nimi

0 references

```
67 protected override void OnModelCreating(ModelBuilder modelBuilder)
68 {
69     modelBuilder.Entity<Company>()
70         .HasDiscriminator<string>("CompanyType")
71         .HasValue<Supplier>("Supplier")
72         .HasValue<Customer>("Customer");
73
74     modelBuilder.Entity<Supplier>()
75         .HasMany(s => s.Products)
76         .WithOne(p => p.Supplier)
77         .HasForeignKey(p => p.SupplierID);
78
79     modelBuilder.Entity<InvoiceProduct>()
80         .HasKey(ip => new { ip.InvoiceID, ip.ProductID });
81
82     modelBuilder.Entity<InvoiceProduct>()
83         .HasOne(ip => ip.Invoice)
84         .WithMany(inv => inv.InvoiceProducts)
85         .HasForeignKey(ip => ip.InvoiceID);
86
87     modelBuilder.Entity<InvoiceProduct>()
88         .HasOne(ip => ip.Product)
89         .WithMany(prod => prod.InvoiceProducts)
90         .HasForeignKey(ip => ip.ProductID);
91 }
92 }
```

Schemat:



Kod dodający klientów:

```
8  var supplier1 = new Supplier
9  {
10     CompanyName = "Delta Logistics",
11     Street = "ul. Transportowa 12",
12     City = "Poznań",
13     ZipCode = "60-001",
14     BankAccountNumber = "PL69 1090 1014 0000 2137 1981 2874"
15 };
16
17 var supplier2 = new Supplier
18 {
19     CompanyName = "TechFusion",
20     Street = "ul. Nowowiejska 5",
21     City = "Kraków",
22     ZipCode = "31-000",
23     BankAccountNumber = "PL61 2345 6789 0420 0123 4567 8901"
24 };
25
26 dbContext.Suppliers.AddRange(supplier1, supplier2);
27 dbContext.SaveChanges();
28
29 var customer1 = new Customer
30 {
31     CompanyName = "Sklep Zabawki",
32     Street = "ul. Dziecięca 3",
33     City = "Wrocław",
34     ZipCode = "50-001",
35     Discount = 0.12m
36 };
37
38 var customer2 = new Customer
39 {
40     CompanyName = "SuperMarket S.A.",
41     Street = "ul. Ogrodowa 7",
42     City = "Gdańsk",
43     ZipCode = "80-001",
44     Discount = 0.05m
45 };
46
47 dbContext.Customers.AddRange(customer1, customer2);
48 dbContext.SaveChanges();
49
```

Kod wyświetlających klientów:

```
117 // tylko suppliers
118 Console.WriteLine("Wyłącznie dostawcy:");
119 var dostawcy = dbContext.Suppliers
120     .AsNoTracking()
121     .OrderBy(s => s.CompanyID)
122     .ToList();
123
124 foreach (var d in dostawcy)
125 {
126     Console.WriteLine($"- SupplierID = {d.CompanyID}, Nazwa = {d.CompanyName}
127         | | | | | | | | | | }, Bank = {d.BankAccountNumber}, Miasto = {d.City}");
128 }
129
130 Console.WriteLine();
131
132 // tylko klienci
133 Console.WriteLine("Wyłącznie klienci:");
134 var klienci = dbContext.Customers
135     .AsNoTracking()
136     .OrderBy(c => c.CompanyID)
137     .ToList();
138
139 foreach (var k in klienci)
140 {
141     Console.WriteLine($"- CustomerID = {k.CompanyID}, Nazwa = {k.CompanyName}
142         | | | | | | | | | | }, Discount = {k.Discount:P0}, Miasto = {k.City}");
143 }
144
```

Wynik:

```
Wyłącznie dostawcy:
- SupplierID = 1, Nazwa = Delta Logistics, Bank = PL69 1090 1014 0000 2137 1981 2874, Miasto = Poznań
- SupplierID = 2, Nazwa = TechFusion, Bank = PL61 2345 6789 0420 0123 4567 8901, Miasto = Kraków

Wyłącznie klienci:
- CustomerID = 3, Nazwa = Sklep Zabawki, Discount = 12%, Miasto = Wrocław
- CustomerID = 4, Nazwa = SuperMarket S.A., Discount = 5%, Miasto = Gdańsk
PS C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet>
```

F) Strategia Table-Per-Type

Użyte tabele (bez zmian)


```
13 public abstract class Company
14 {
15     4 references
16     public int CompanyID { get; set; }
17     6 references
18     public required string CompanyName { get; set; }
19     4 references
20     public string? Street { get; set; }
21     4 references
22     public string? City { get; set; }
23     4 references
24     public string? ZipCode { get; set; }
25 }
26
27 6 references
28 public class Supplier : Company
29 {
30     3 references
31     public string? BankAccountNumber { get; set; }
32     1 reference
33     public ICollection<Product> Products { get; set; } = new List<Product>();
34 }
35
36 4 references
37 public class Customer : Company
38 {
39     3 references
40     public decimal Discount { get; set; }
41 }
42
```

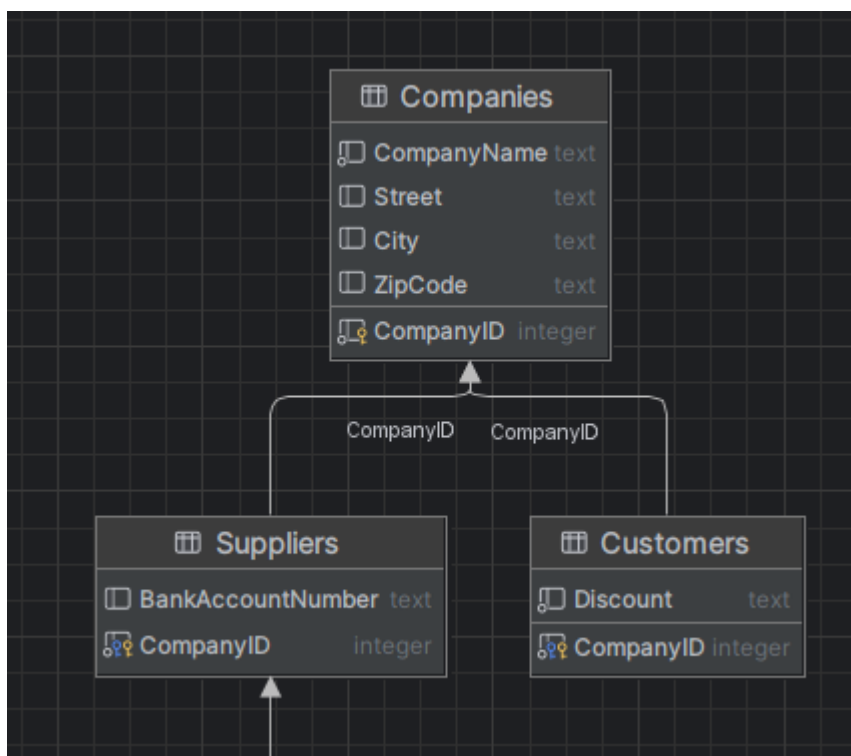
Nowa modyfikacja relacji:

```

67 protected override void OnModelCreating(ModelBuilder modelBuilder)
68 {
69     // Mapowanie tabeli głównej dla Company
70     modelBuilder.Entity<Company>()
71         .ToTable("Companies"); // wspólne CompanyID, CompanyName, Street, City, ZipCode
72
73     // Mapowanie tabeli dla Supplier
74     modelBuilder.Entity<Supplier>()
75         .ToTable("Suppliers"); // tutaj EF Core utworzy CompanyID i BankAccountNumber
76
77     // Mapowanie tabeli dla Customer
78     modelBuilder.Entity<Customer>()
79         .ToTable("Customers"); // a tutaj EF Core doda CompanyID i Discount
80
81     // Relacja Supplier - Product
82     modelBuilder.Entity<Supplier>()
83         .HasMany(s => s.Products)
84         .WithOne(p => p.Supplier)
85         .HasForeignKey(p => p.SupplierID);
86
87     // Relacja InvoiceProduct jako łącznik wiele-do-wielu
88     modelBuilder.Entity<InvoiceProduct>()
89         .HasKey(ip => new { ip.InvoiceID, ip.ProductID });
90
91     modelBuilder.Entity<InvoiceProduct>()
92         .HasOne(ip => ip.Invoice)
93         .WithMany(inv => inv.InvoiceProducts)
94         .HasForeignKey(ip => ip.InvoiceID);
95
96     modelBuilder.Entity<InvoiceProduct>()
97         .HasOne(ip => ip.Product)
98         .WithMany(prod => prod.InvoiceProducts)
99         .HasForeignKey(ip => ip.ProductID);
100 }
101 }

```

Schemat:



Kod dodający klientów:

```
8  var supplier1 = new Supplier
9  {
10     CompanyName = "Delta Logistics",
11     Street = "ul. Transportowa 12",
12     City = "Poznań",
13     ZipCode = "60-001",
14     BankAccountNumber = "PL61 1090 1014 0000 0712 1981 2874"
15 };
16 var supplier2 = new Supplier
17 {
18     CompanyName = "TechFusion",
19     Street = "ul. Nowowiejska 5",
20     City = "Kraków",
21     ZipCode = "31-000",
22     BankAccountNumber = "PL61 2345 6789 0000 0123 4567 8901"
23 };
24 dbContext.Suppliers.AddRange(supplier1, supplier2);
25 dbContext.SaveChanges();
26
27 var customer1 = new Customer
28 {
29     CompanyName = "Sklep Zabawki",
30     Street = "ul. Dziecięca 3",
31     City = "Wrocław",
32     ZipCode = "50-001",
33     Discount = 0.12m
34 };
35 var customer2 = new Customer
36 {
37     CompanyName = "SuperMarket S.A.",
38     Street = "ul. Ogrodowa 7",
39     City = "Gdańsk",
40     ZipCode = "80-001",
41     Discount = 0.05m
42 };
43 dbContext.Customers.AddRange(customer1, customer2);
44 dbContext.SaveChanges();
45
```

Kod wyświetlających klientów:

```
46 // supplierzy
47 Console.WriteLine("Supplier:");
48 var dostawcy = dbContext.Suppliers
49     .AsNoTracking()
50     .OrderBy(s => s.CompanyID)
51     .ToList();
52 foreach (var d in dostawcy)
53 {
54     Console.WriteLine($"- SupplierID={d.CompanyID}, Nazwa={d.CompanyName}, Bank={d.BankAccountNumber}");
55 }
56
57 Console.WriteLine();
58
59 // customersi
60 Console.WriteLine("Customer:");
61 var klienci = dbContext.Customers
62     .AsNoTracking()
63     .OrderBy(c => c.CompanyID)
64     .ToList();
65 foreach (var k in klienci)
66 {
67     Console.WriteLine($"- CustomerID={k.CompanyID}, Nazwa={k.CompanyName}, Discount={k.Discount:P0}");
68 }
```

Wynik:

```
Supplier:
- SupplierID=1, Nazwa=Delta Logistics, Bank=PL61 1090 1014 0000 0712 1981 2874
- SupplierID=2, Nazwa=TechFusion, Bank=PL61 2345 6789 0000 0123 4567 8901

Customer:
- CustomerID=3, Nazwa=Sklep Zabawki, Discount=12%
- CustomerID=4, Nazwa=SuperMarket S.A., Discount=5%
PS C:\Users\Bartek\Documents\GitHub\DataBases-2025\dotnet> 
```

G) Porównanie obu strategii

Strategie **TPH** (**Table-per-Hierarchy**) i **TPT** (**Table-per-Type**) różnią się sposobem odwzorowania dziedziczenia w relacyjnej bazie danych, a co za tym idzie: wydajnością, przejrzystością danych i ich spójnością.

TPH umieszcza całą hierarchię dziedziczenia w jednej tabeli. Skutkuje to tym, że wszystkie właściwości z klas bazowych i pochodnych znajdują się w jednej strukturze danych, a nieużywane kolumny dla danego rekordu pozostają puste. TPH upraszcza model bazy, jest bardziej wydajna przy odczytach, ponieważ eliminuje konieczność używania złożonych zapytań z JOINami. Jednak ta prostota i wydajność okupiona jest mniejszą przejrzystością danych, trudnością w zapewnieniu spójności oraz obecnością wielu NULL-i w kolumnach nieistotnych dla konkretnego typu.

Z kolei **TPT** rozdziela strukturę danych między osobne tabele – każda klasa dziedzicząca ma własną tabelę zawierającą wyłącznie swoje właściwości, a dane wspólne są przechowywane w tabeli bazowej. Zapytania do takich danych wymagają dołączeń między tabelami, co znacząco wpływa na wydajność przy dużych zbiorach danych lub częstym odczycie. TPT zapewnia jednak większą przejrzystość modelu i pozwala zachować ścisłą spójność danych (np. pola typowe tylko dla PublicCompany nie mogą pojawić się przypadkowo w rekordzie

PrivateCompany). Struktura jest bliższa obiektowemu dziedziczeniu i czytelniejsza, szczególnie przy bardziej rozbudowanych hierarchiach klas.