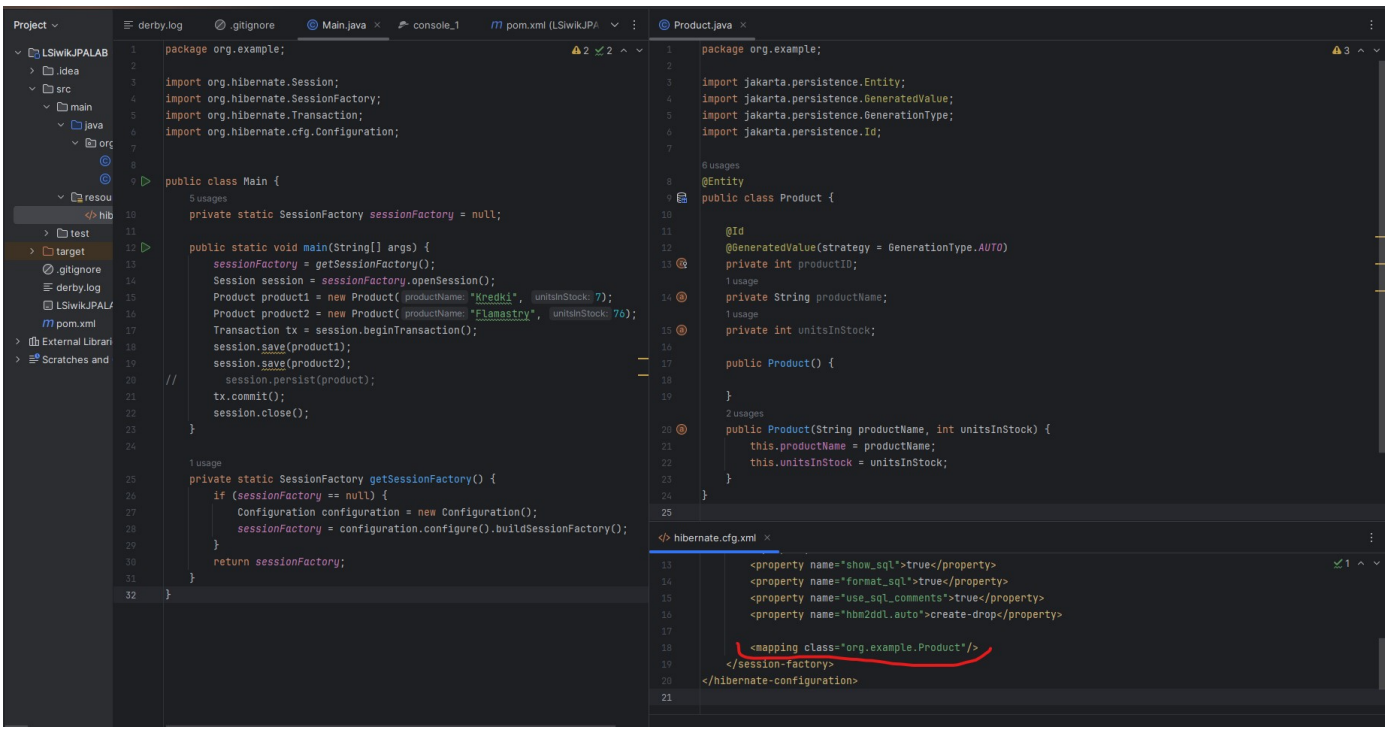
Hibernate, JPA – laboratorium

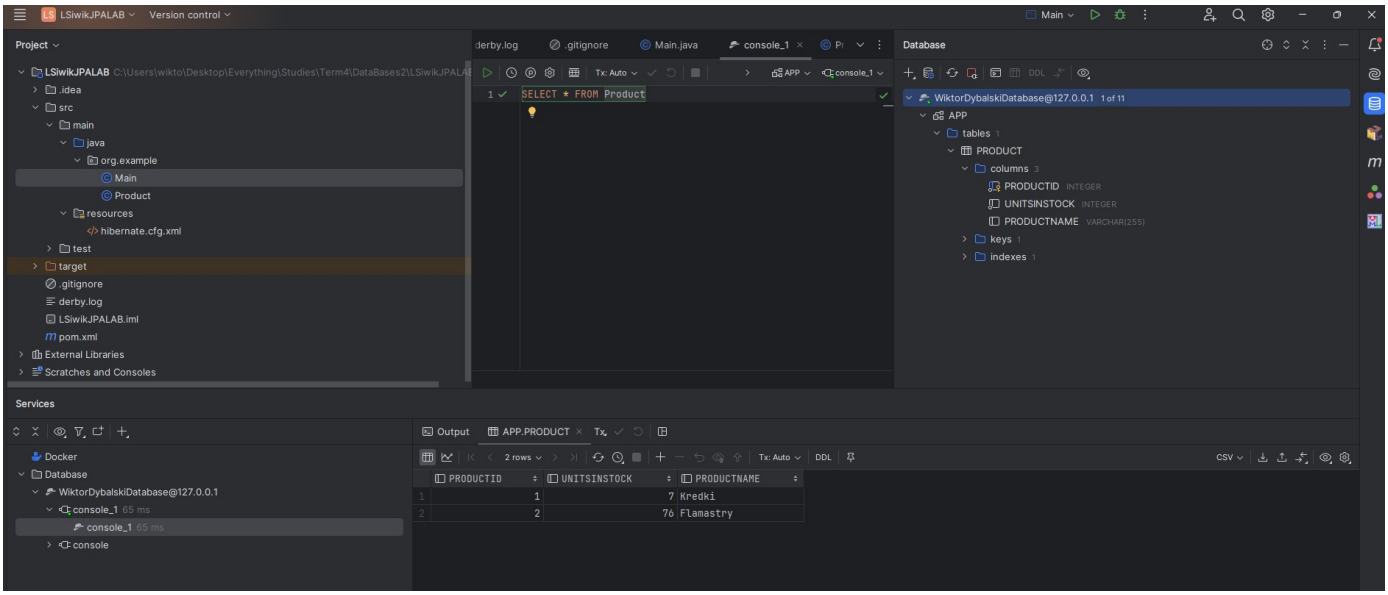
ćwiczenie 4

Imiona i nazwiska autorów: Stas Kochevenko & Wiktor Dybalski

Wprowadzenie

W trakcie części "przewodnikowej" została dodana klasa Product. Dodaliśmy przykładowe obiekty tej klasy do bazy danych z wykorzystaniem technologii hibernate'a, a następnie przekonaliśmy się w tym, że są one widoczne w bazie danych.





Zadanie 1 - wprowadzenie pojęcia Dostawcy

Zostały dodane podstawowe klasy Product, Supplier oraz Program, baza danych Database.

Kod:

- Product:

```
@Entity
class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productID;
    private String productName;
    private int unitsInStock;

    @ManyToOne
    private Supplier supplier;

    public Product() {

    }

    public Product(String productName, int unitsInStock, Supplier supplier) {
        this.productName = productName;
        this.unitsInStock = unitsInStock;
        this.supplier = supplier;
    }
}
```

- Dodane zostały dwie linijki do klasy hibernate.cfg/xml:

```
        <mapping class="Supplier"/>
        <mapping class="Product"/>
```

- Supplier:

```
@Entity
class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    String companyName;
    String street;
    String city;

    public Supplier() {

    }

    public Supplier(String companyName, String street, String city) {
        this.companyName = companyName;
        this.street = street;
        this.city = city;
    }


    public int getSupplierID() {
        return supplierID;
    }
```

```
    public String getCompanyName() {
        return companyName;
    }

    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    @Override
    public String toString() { return companyName; }
}
```

- Main:

```
class Main {
    private static SessionFactory sessionFactory = null;

    private static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            Configuration configuration = new Configuration();
            sessionFactory = configuration.configure().buildSessionFactory();
        }
        return sessionFactory;
    }

    public static void main(String[] args) {

        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();

        Transaction tx;
        Scanner scanner = new Scanner(System.in);
        int option;

        while (true) {
            System.out.println("--------------------");
            System.out.println("What do you want to do?: ");
            System.out.println("1. Add product");
            System.out.println("2. Add supplier");
            System.out.println("3. Show products");
            System.out.println("4. Show suppliers");
            System.out.println("5. Find last added supplier");
            System.out.println("6. Set supplier for product");
            System.out.println("7. Exit");
            System.out.println("--------------------");
            option = scanner.nextInt();
            scanner.nextLine();

            switch (option) {
                case 1:
                    tx = session.beginTransaction();
                    System.out.print("Enter product name: ");
                    String productName = scanner.nextLine();
                    System.out.print("Enter units in stock: ");
```

```
                    int unitsInStock = scanner.nextInt();
                    scanner.nextLine();

                    List<Supplier> suppliers = session.createQuery("FROM Supplier", Supplier.class).list();
                    Supplier selectedSupplier;
                    if (suppliers.isEmpty()) {
                        System.out.println("No suppliers found. You need to add a supplier first.");
                        System.out.print("Enter supplier name: ");
                        String newSupplierName = scanner.nextLine();
                        System.out.print("Enter supplier street: ");
                        String newSupplierStreet = scanner.nextLine();
                        System.out.print("Enter supplier city: ");
                        String newSupplierCity = scanner.nextLine();

                        selectedSupplier = new Supplier(newSupplierName, newSupplierStreet, newSupplierCity);
                        session.save(selectedSupplier);
                        System.out.println("New supplier added successfully.");
                    } else {
                        System.out.println("Select a supplier (by ID):");
                        for (Supplier supplier : suppliers) {
                            System.out.println("ID: " + supplier.getSupplierID() + " - Name: " +
supplier.getCompanyName());
                        }
                        System.out.print("Enter supplier ID: ");
                        int supplierId = scanner.nextInt();
                        scanner.nextLine();

                        selectedSupplier = session.get(Supplier.class, supplierId);

                        if (selectedSupplier == null) {
                            System.out.println("Invalid supplier ID. Please try again.");
                            break;
                        }
                    }

                    Product product = new Product(productName, unitsInStock, selectedSupplier);
                    session.save(product);
                    tx.commit();
                    System.out.println("Product added successfully.");
                    break;

                case 2:
                    tx = session.beginTransaction();
                    System.out.print("Enter supplier name: ");
                    String supplierName = scanner.nextLine();
                    System.out.print("Enter supplier street: ");
                    String supplierRoad = scanner.nextLine();
                    System.out.print("Enter supplier city: ");
                    String supplierCity = scanner.nextLine();

                    Supplier supplier = new Supplier(supplierName, supplierRoad, supplierCity);
                    session.save(supplier);
                    tx.commit();
                    System.out.println("Supplier added successfully.");
                    break;
                case 3:
                    List<Product> products = session.createQuery("FROM Product", Product.class).list();
                    if (!products.isEmpty()) {
                        for (Product p : products) {
                            System.out.print("Product ID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.print(", Units in Stock: " + p.getUnitsInStock());
                            System.out.println(", Supplier: " + p.getSupplier().getCompanyName());
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    break;

                case 4:
                    suppliers = session.createQuery("FROM Supplier", Supplier.class).list();
                    if (!suppliers.isEmpty()) {
```

```
                        for (Supplier s : suppliers) {
                            System.out.print("Supplier ID: " + s.getSupplierID());
                            System.out.print(", Name: " + s.getCompanyName());
                            System.out.print(", Street: " + s.getStreet());
                            System.out.println(", City: " + s.getCity());
                        }
                    } else {
                        System.out.println("No suppliers found.");
                    }
                    break;
                case 5:
                    Query<Supplier> query = session.createQuery("FROM Supplier ORDER BY id DESC",
Supplier.class);
                    query.setMaxResults(1);
                    Supplier lastSupplier = query.uniqueResult();

                    if (lastSupplier != null) {
                        System.out.print("Last Supplier:");
                        System.out.print(", Name: " + lastSupplier.getCompanyName());
                        System.out.print(", Street: " + lastSupplier.getStreet());
                        System.out.println(", City: " + lastSupplier.getCity());
                    } else {
                        System.out.println("No suppliers found.");
                    }
                    break;
                case 6:
                    Query<Supplier> lastSupQuery = session.createQuery("FROM Supplier ORDER BY id DESC",
Supplier.class);
                    lastSupQuery.setMaxResults(1);
                    lastSupplier = lastSupQuery.uniqueResult();

                    Query<Product> lastProdQuery = session.createQuery("FROM Product ORDER BY id DESC",
Product.class);
                    lastProdQuery.setMaxResults(1);
                    Product lastProduct = lastProdQuery.uniqueResult();
                    lastProduct.setSupplier(lastSupplier);
                    System.out.println("Setting supplier: " + lastSupplier.getCompanyName() + "for product: " +
lastProduct.getProductName());
                    break;
                case 7:
                    session.close();
                    scanner.close();
                    System.out.println("Goodbye!");
                    return;

                default:
                    System.out.println("Invalid option, please try again.");
            }
        }
    }
}
```

Po dodaniu kilku produktów oraz dostawców nasza baza wygląda tak:

Main.java × | Supplier.java | Product.java | console | WiktorDybalskiDatabase@127.0.0.1 | hibernate.cfg.xml | console_

```java
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

public class Main {
    5 usages
    private static SessionFactory sessionFactory = null;

    public static void main(String[] args) {
        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();

        Supplier supplier1 = new Supplier( companyName: "Samsung",  street: "Kawiory",  city: "Kraków");
        Supplier supplier2 = new Supplier( companyName: "Sony",  street: "Wroclawska",  city: "Kraków");

        Product product1 = new Product( productName: "Kredki",  unitsInStock: 7, supplier1);
        Product product2 = new Product( productName: "Flamastry",  unitsInStock: 76, supplier1);
        Product product3 = new Product( productName: "Rowery",  unitsInStock: 11, supplier2);
        Product product4 = new Product( productName: "Samochody",  unitsInStock: 3, supplier2);

        Transaction tx = session.beginTransaction();
        session.save(supplier1);
        session.save(supplier2);
        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(product4);
```

Products:

```
--------------------
What do you want to do?:
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Find last added supplier
6. Exit
--------------------
3
Product ID: 1, Name: Kredki, Units in Stock: 7, Supplier: Samsung
Product ID: 2, Name: Flamastry, Units in Stock: 76, Supplier: Samsung
Product ID: 3, Name: Rowery, Units in Stock: 11, Supplier: Sony
Product ID: 4, Name: Samochody, Units in Stock: 3, Supplier: Sony
```

Suppliers:

```
What do you want to do?:
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Find last added supplier
6. Exit
--------------------
4
Supplier ID: 1, Name: Samsung, Street: Kawiory, City: Kraków
Supplier ID: 2, Name: Sony, Street: Wroclawska, City: Kraków
```

Sprawdzamy następnie dodanie nowego produktu oraz wyszukanie ostatniego dostawcy:

Dodanie produktu:

```
What do you want to do?:
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Find last added supplier
6. Exit
--------------------
1
Enter product name: Myszki
Enter units in stock: 13
Select a supplier (by ID):
ID: 1 - Name: Samsung
ID: 2 - Name: Sony
Enter supplier ID: 1
Product added successfully.
```

Output APP.PRODUCT Tx

5 rows Tx: Auto DDL

	PRODUCTID	SUPPLIER_SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	7	Kredki
2	2	1	76	Flamastry
3	3	2	11	Rowery
4	4	2	3	Samochody
5	5052	1	14	Myszki

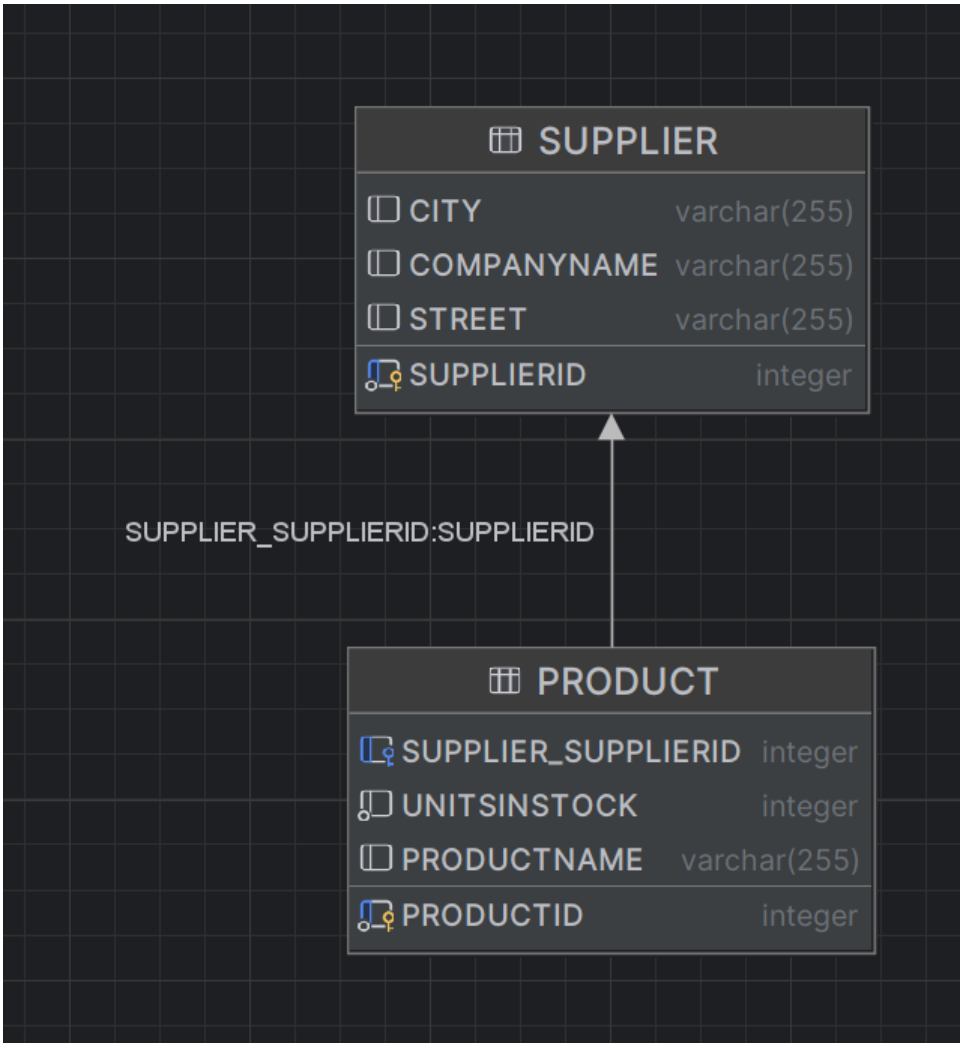
Wyszukanie ostatniego dodanego dostawcy:

```
What do you want to do?:
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Find last added supplier
6. Remove product
7. Remove supplier
8. Exit
--------------------
5
Last Supplier:, Name: Sony, Street: Wroclawska, City: Kraków
--------------------
```

Ustawienie ostatnio dodanego produktu(id=5052) ostatnio dodanego suppliera:

	PRODUCTID	SUPPLIER_SUPPLIERID	UNITSINSTOCK	PRODUCTNAME	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	1	7	Kredki	1	Kraków	Samsung	Kawiory
2	2	1	76	Flamastry	1	Kraków	Samsung	Kawiory
3	5052	1	14	Myszki	1	Kraków	Samsung	Kawiory
4	3	2	11	Rowery	2	Kraków	Sony	Wroclawska
5	4	2	3	Samochody	2	Kraków	Sony	Wroclawska

Diagram naszej bazy wygląda tak:



Analizując schemat widać, że baza danych poprawnie zoptymalizowała sobie naszą relację Product-Supplier a sam SupplierID jest kluczem obcym w tabeli Product

Zadanie 2 - odwrócenie relacji Supplier -> Product

a) z tabelą łącznikową:

W klasie main dodano do case przypadek zerowy dodający automatycznie kilka produktów oraz dostwaców:

- Main:

```
class Main {
    private static SessionFactory sessionFactory = null;

    private static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            Configuration configuration = new Configuration();
            sessionFactory = configuration.configure().buildSessionFactory();
        }
        return sessionFactory;
    }

    public static void main(String[] args) {

        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();

        Transaction tx;
        Scanner scanner = new Scanner(System.in);
        int option;

        while (true) {
            System.out.println("--------------------");
            System.out.println("What do you want to do?: ");
            System.out.println("0. Add template");
            System.out.println("1. Add product");
            System.out.println("2. Add supplier");
            System.out.println("3. Show products");
            System.out.println("4. Show suppliers");
            System.out.println("5. Find last added supplier");
            System.out.println("6. Add products to last Supplier");
            System.out.println("7. Exit");
            System.out.println("--------------------");
            option = scanner.nextInt();
            scanner.nextLine();

            switch (option) {
                case 0:
                    tx = session.beginTransaction();

                    Supplier supplier1 = new Supplier("Acme Corp", "123 Main St", "Springfield");
                    Supplier supplier2 = new Supplier("Tech Solutions", "456 Elm St", "Shelbyville");
                    Supplier supplier3 = new Supplier("Global Traders", "789 Oak St", "Capital City");

                    Product product1 = new Product("Laptop", 20);
                    Product product2 = new Product("Smartphone", 50);
                    Product product3 = new Product("Tablet", 30);
                    Product product4 = new Product("Monitor", 15);
                    Product product5 = new Product("Printer", 10);

                    supplier1.addProduct(product1);
                    supplier1.addProduct(product2);

                    supplier2.addProduct(product3);
                    supplier2.addProduct(product4);

                    supplier3.addProduct(product5);

                    session.save(supplier1);
                    session.save(supplier2);
                    session.save(supplier3);
```

```
                    session.save(product1);
                    session.save(product2);
                    session.save(product3);
                    session.save(product4);
                    session.save(product5);

                    tx.commit();
                    System.out.println("Product added successfully.");
                    break;
                case 1:
                    tx = session.beginTransaction();
                    System.out.print("Enter product name: ");
                    String productName = scanner.nextLine();
                    System.out.print("Enter units in stock: ");
                    int unitsInStock = scanner.nextInt();
                    Product product = new Product(productName, unitsInStock);
                    session.save(product);
                    tx.commit();
                    System.out.println("Product added successfully.");
                    break;

                case 2:
                    tx = session.beginTransaction();
                    System.out.print("Enter supplier name: ");
                    String supplierName = scanner.nextLine();
                    System.out.print("Enter supplier street: ");
                    String supplierRoad = scanner.nextLine();
                    System.out.print("Enter supplier city: ");
                    String supplierCity = scanner.nextLine();

                    Supplier supplier = new Supplier(supplierName, supplierRoad, supplierCity);
                    session.save(supplier);
                    tx.commit();
                    System.out.println("Supplier added successfully.");
                    break;
                case 3:
                    tx = session.beginTransaction();
                    List<Product> products = session.createQuery("FROM Product", Product.class).list();
                    if (!products.isEmpty()) {
                        for (Product p : products) {
                            System.out.print("Product ID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.println(", Units in Stock: " + p.getUnitsInStock());
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;

                case 4:
                    tx = session.beginTransaction();
                    List<Supplier> suppliers = session.createQuery("FROM Supplier", Supplier.class).list();
                    if (!suppliers.isEmpty()) {
                        for (Supplier s : suppliers) {
                            System.out.print("Supplier ID: " + s.getSupplierID());
                            System.out.print(", Name: " + s.getCompanyName());
                            System.out.print(", Street: " + s.getStreet());
                            System.out.println(", City: " + s.getCity());
                        }
                    } else {
                        System.out.println("No suppliers found.");
                    }
                    tx.commit();
                    break;
                case 5:
                    Query<Supplier> query = session.createQuery("FROM Supplier ORDER BY id DESC",
Supplier.class);
                    query.setMaxResults(1);
                    Supplier lastSupplier = query.uniqueResult();
```

```
                    if (lastSupplier != null) {
                        System.out.print("Last Supplier:");
                        System.out.print(", Name: " + lastSupplier.getCompanyName());
                        System.out.print(", Street: " + lastSupplier.getStreet());
                        System.out.println(", City: " + lastSupplier.getCity());
                    } else {
                        System.out.println("No suppliers found.");
                    }
                    break;
                case 6:
                    tx = session.beginTransaction();
                    Query<Supplier> lastSupQuery = session.createQuery("FROM Supplier ORDER BY id DESC",
Supplier.class);
                    lastSupQuery.setMaxResults(1);
                    lastSupplier = lastSupQuery.uniqueResult();

                    Query<Product> lastProdQuery = session.createQuery("FROM Product ORDER BY id DESC",
Product.class);
                    List<Product> lastProductsList = lastProdQuery.setMaxResults(3).list();

                    for (Product productt : lastProductsList) {
                        lastSupplier.addProduct(productt);
                        System.out.println("Adding product: " + productt.getProductName() + " to supplier: " +
lastSupplier.getCompanyName());
                    }
                    tx.commit();
                    System.out.println(lastSupplier.getProducts());
                    break;
                case 7:
                    session.close();
                    scanner.close();
                    System.out.println("Goodbye!");
                    return;

                default:
                    System.out.println("Invalid option, please try again.");
            }
        }
    }
}
```

W dostawcy zmienia się tylko to, że dodajemy listę produktów jakie dostarcza dany dostawca korzystając z tablicy łącznikowej

- Supplier

```
@Table(name = "Suppliers")
@Entity
@SequenceGenerator(name = "Supplier_SEQ")
class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Supplier_SEQ")
    private int supplierID;

    private String companyName;
    private String street;
    private String city;

    @OneToMany
    @JoinTable(
            name = "SupplierProducts",
            joinColumns = @JoinColumn(name = "supplierID"),
            inverseJoinColumns = @JoinColumn(name = "productID")
    )
    private List<Product> products = new ArrayList<>();

    public Supplier() {
    }
```

```java
    Supplier(String companyName, String street, String city) {
        this.companyName = companyName;
        this.street = street;
        this.city = city;
    }

    int getSupplierID() {
        return supplierID;
    }

    String getCompanyName() {
        return companyName;
    }

    String getStreet() {
        return street;
    }

    String getCity() {
        return city;
    }

    List<Product> getProducts() {
        return products;
    }

    void addProduct(Product product) {
        this.products.add(product);
    }

    @Override
    public String toString() { return companyName; }
}
```

W klasie Product zostało usunięte pole Supplier supplier;

Rezultat wykonania case 0:

- Suppliers

0 rows

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Springfield	Acme Corp	123 Main St
2	2	Shelbyville	Tech Solutions	456 Elm St
3	3	Capital City	Global Traders	789 Oak St

- Products

5 rows Tx: Auto

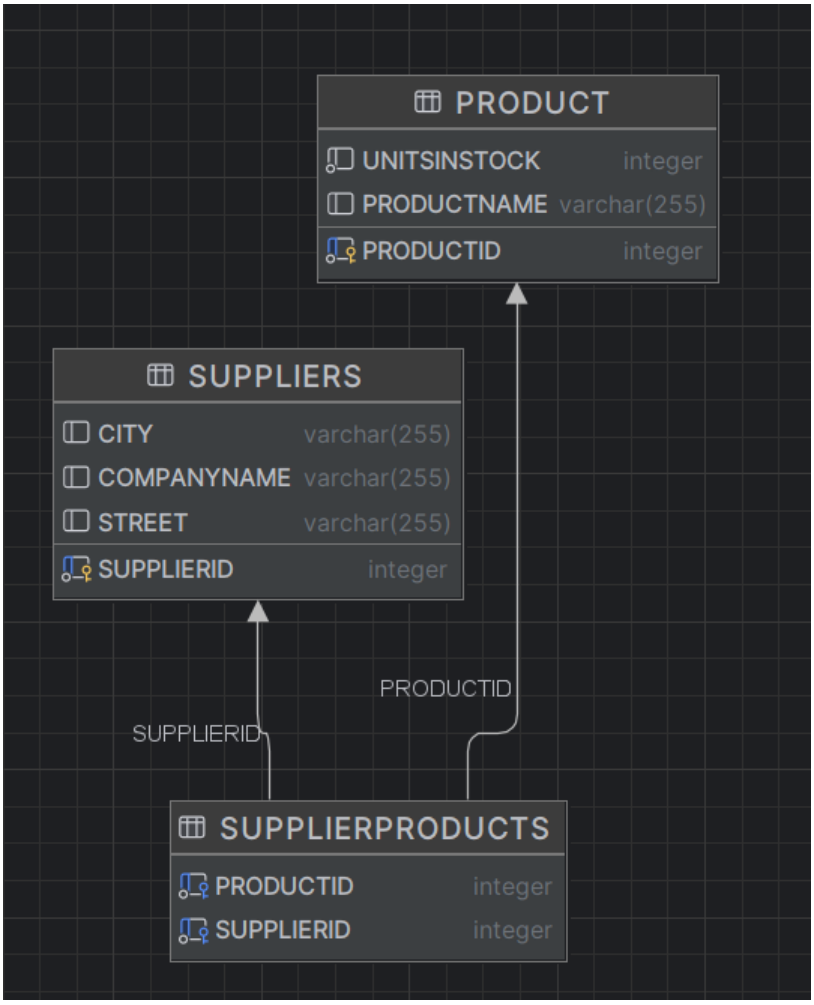
	PRODUCTID	UNITSINSTOCK	PRODUCTNAME
1	1	20	Laptop
2	2	50	Smartphone
3	3	30	Tablet
4	4	15	Monitor
5	5	10	Printer

- SupplierProducts

5 rows

	PRODUCTID	SUPPLIERID
1	1	1
2	2	1
3	3	2
4	4	2
5	5	3

Aktualny schemat bazy:



b) bez tabeli łącznikowej:

Nie zmienia się nic oprócz klasy Supplier. Tabela łącznikowa została zamieniona na listę produktów.

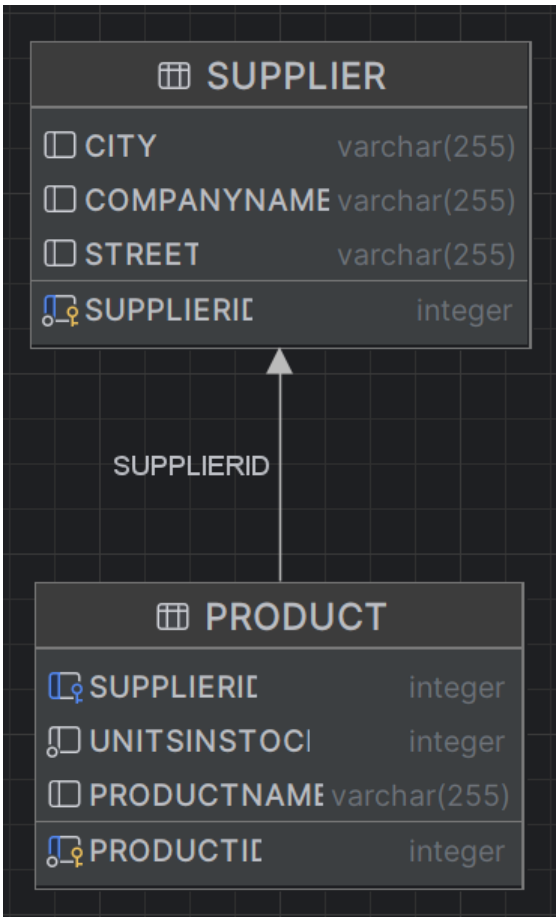
- Suppliers

```
@SequenceGenerator(name = "Supplier_SEQ")
class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Supplier_SEQ")
    private int supplierID;

    private String companyName;
    private String street;
    private String city;

    @OneToMany()
    private List<Product> products = new ArrayList<>();
```

```
    public Supplier() {
    }
    Supplier(String companyName, String street, String city) {
        this.companyName = companyName;
        this.street = street;
        this.city = city;
    }
    int getSupplierID() {
        return supplierID;
    }
    String getCompanyName() {
        return companyName;
    }
    String getStreet() {
        return street;
    }
    String getCity() {
        return city;
    }
    List<Product> getProducts() {
        return products;
    }
    void addProduct(Product product) {
        this.products.add(product);
    }

    @Override
    public String toString() { return companyName; }
}
```

Baza danych optymalizuje sobie połączenie pomiędzy tabelami i przedstawia ją w taki sam sposób jak w przypadku relacji z zadania 1.

Aktualny schemat bazy:



Zadanie 3 - dwustronna relacja Supplier <----> Product

Łączymy poprzednie rozwiązania, czyli w klasie Product będzie się znajdować ID Suppliera, który dostarcza ten produkt, a każdy Supplier będzie miał listę tych produktów, które dostarcza.

- Zmiana w klasie Product:

```
@JoinColumn(name = "supplierID")
private Supplier supplier;
```

- Zmiana w klasie Supplier:

```
@OneToMany(mappedBy = "supplier")
private List<Product> products = new ArrayList<>();
```

W case 0 w klasie Main dodaliśmy podwójną relację.

- Zmiana w klasie Main:

```
case 0:
tx =session.

beginTransaction();

Supplier supplier1 = new Supplier("Acme Corp", "123 Main St", "Springfield");
Supplier supplier2 = new Supplier("Tech Solutions", "456 Elm St", "Shelbyville");
Supplier supplier3 = new Supplier("Global Traders", "789 Oak St", "Capital City");

Product product1 = new Product("Laptop", 20);
Product product2 = new Product("Smartphone", 50);
Product product3 = new Product("Tablet", 30);
Product product4 = new Product("Monitor", 15);
Product product5 = new Product("Printer", 10);

product1.setSupplier(supplier1);
product2.setSupplier(supplier1);
roduct3.setSupplier(supplier2);
product4.setSupplier(supplier2);
product5.setSupplier(supplier3);

supplier1.addProduct(product1);
supplier1.addProduct(product2);
supplier2.addProduct(product3);
supplier2.addProduct(product4);
supplier3.addProduct(product5);

session.save(supplier1);
session.save(supplier2);
session.save(supplier3);
session.save(product1);
session.save(product2);
session.save(product3);
session.save(product4);
session.save(product5);

tx.commit();
System.out.println("Product added successfully.");

break;
```

- SQL logi po wykonaniu kroku 0:

What do you want to do?:

0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Find last added supplier
6. Add products to last Supplier
7. Exit

0

Hibernate:

```
values
    next value for Supplier_SEQ
    Hibernate:

values
    next value for Supplier_SEQ
    Hibernate:

values
    next value for Products_SEQ
    Hibernate:

values
    next value for Products_SEQ
    Hibernate:
/* insert for
    zad1.Supplier */insert
into
    Supplier (city, companyName, street, supplierID)
values
    (?, ?, ?, ?)
    Hibernate:
/* insert for
    zad1.Supplier */insert
into
    Supplier (city, companyName, street, supplierID)
values
    (?, ?, ?, ?)
    Hibernate:
/* insert for
    zad1.Supplier */insert
into
    Supplier (city, companyName, street, supplierID)
values
    (?, ?, ?, ?)
    Hibernate:
/* insert for
    zad1.Product */insert
into
    Product (productName, supplierID, unitsInStock, productID)
values
    (?, ?, ?, ?)
    Hibernate:
/* insert for
    zad1.Product */insert
into
    Product (productName, supplierID, unitsInStock, productID)
values
    (?, ?, ?, ?)
    Hibernate:
```

```
/* insert for
    zad1.Product */insert
into
    Product (productName, supplierID, unitsInStock, productID)
values
    (?, ?, ?, ?)
    Hibernate:
/* insert for
    zad1.Product */insert
into
    Product (productName, supplierID, unitsInStock, productID)
values
    (?, ?, ?, ?)
    Hibernate:
/* insert for
    zad1.Product */insert
into
    Product (productName, supplierID, unitsInStock, productID)
values
    (?, ?, ?, ?)
    Product added successfully.
```

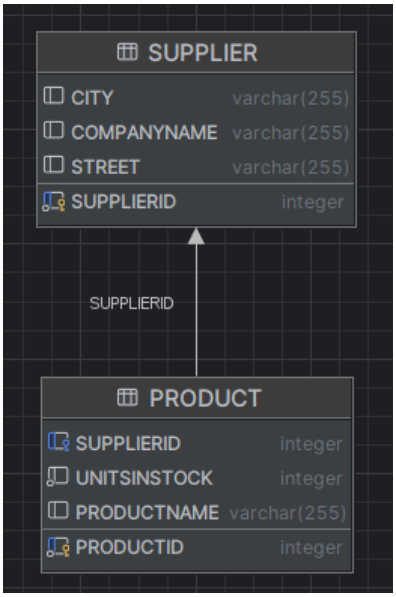
Products:

	PRODUCTID	SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	20	Laptop
2	2	1	50	Smartphone
3	3	2	30	Tablet
4	4	2	15	Monitor
5	5	3	10	Printer

Suppliers:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Springfield	Acme Corp	123 Main St
2	2	Shelbyville	Tech Solutions	456 Elm St
3	3	Capital City	Global Traders	789 Oak St

Zoptymalizowany schemat bazy danych:



Jak widać baza danych znowu zoptymalizowała sobie relacje pomiędzy tabelami, podobnie jak to było w Entity Framework.

Zadanie 4 - dodanie kategorii produktu

Została stworzona relacja identyczna jak dla Suppliera czyli one-to-many

- Do klasy Product dodano:

```
    @ManyToOne
    @JoinColumn(name = "categoryID")
    private Category category;

...
public void setCategory(Category category) {
    this.category = category;
}
```

- Category:

```
@Entity
@SequenceGenerator(name = "Categories_SEQ")
class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "Categories_SEQ")
    int categoryID;
    String name;

    @OneToMany(mappedBy = "category")
    List<Product> products = new ArrayList<>();

    public Category() {
    }

    public Category(String name) {
        this.name = name;
    }

    public int getCategoryID() {
        return categoryID;
    }

    public String getName() {
        return name;
    }

    public List<Product> getProducts() {
        return products;
    }

    public void addProducts(Product product) {
        products.add(product);
    }

    @Override
    public String toString() { return name; }
}
```

- Main po modyfikacji:

```
class Main {
    private static SessionFactory sessionFactory = null;

    private static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
```

```
            Configuration configuration = new Configuration();
            sessionFactory = configuration.configure().buildSessionFactory();
        }
        return sessionFactory;
    }

    public static void main(String[] args) {

        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();

        Transaction tx;
        Scanner scanner = new Scanner(System.in);
        int option;

        while (true) {
            System.out.println("--------------------");
            System.out.println("What do you want to do?: ");
            System.out.println("0. Add template");
            System.out.println("1. Add product");
            System.out.println("2. Add supplier");
            System.out.println("3. Show products");
            System.out.println("4. Show suppliers");
            System.out.println("5. Show suppliers");
            System.out.println("6. Show products with Category");
            System.out.println("7. Show Product Category");
            System.out.println("8. Exit");
            System.out.println("--------------------");
            option = scanner.nextInt();
            scanner.nextLine();

            switch (option) {
                case 0:
                    tx = session.beginTransaction();

                    Supplier supplier1 = new Supplier("Acme Corp", "123 Main St", "Springfield");
                    Supplier supplier2 = new Supplier("Tech Solutions", "456 Elm St", "Shelbyville");
                    Supplier supplier3 = new Supplier("Global Traders", "789 Oak St", "Capital City");

                    Product product1 = new Product("Laptop", 20);
                    Product product2 = new Product("Smartphone", 50);
                    Product product3 = new Product("Tablet", 30);
                    Product product4 = new Product("Books", 15);
                    Product product5 = new Product("Car", 10);

                    Category category1 = new Category("Devices");
                    Category category2 = new Category("School");
                    Category category3 = new Category("Vehicles");

                    product1.setSupplier(supplier1);
                    product1.setCategory(category1);

                    product2.setSupplier(supplier1);
                    product2.setCategory(category1);

                    product3.setSupplier(supplier2);
                    product3.setCategory(category1);

                    product4.setSupplier(supplier2);
                    product4.setCategory(category2);

                    product5.setSupplier(supplier3);
                    product5.setCategory(category3);


                    supplier1.addProduct(product1);
                    supplier1.addProduct(product2);

                    supplier2.addProduct(product3);
                    supplier2.addProduct(product4);

                    supplier3.addProduct(product5);
```

```
                category1.addProducts(product1);
                category1.addProducts(product2);
                category1.addProducts(product3);

                category2.addProducts(product4);

                category3.addProducts(product5);


                session.save(supplier1);
                session.save(supplier2);
                session.save(supplier3);

                session.save(category1);
                session.save(category2);
                session.save(category3);

                session.save(product1);
                session.save(product2);
                session.save(product3);
                session.save(product4);
                session.save(product5);

                tx.commit();
                System.out.println("Product added successfully.");
                break;
            case 1:
                tx = session.beginTransaction();
                System.out.print("Enter product name: ");
                String productName = scanner.nextLine();
                System.out.print("Enter units in stock: ");
                int unitsInStock = scanner.nextInt();
                Product product = new Product(productName, unitsInStock);
                session.save(product);
                tx.commit();
                System.out.println("Product added successfully.");
                break;

            case 2:
                tx = session.beginTransaction();
                System.out.print("Enter supplier name: ");
                String supplierName = scanner.nextLine();
                System.out.print("Enter supplier street: ");
                String supplierRoad = scanner.nextLine();
                System.out.print("Enter supplier city: ");
                String supplierCity = scanner.nextLine();

                Supplier supplier = new Supplier(supplierName, supplierRoad, supplierCity);
                session.save(supplier);
                tx.commit();
                System.out.println("Supplier added successfully.");
                break;
            case 3:
                tx = session.beginTransaction();
                List<Product> products = session.createQuery("FROM Product", Product.class).list();
                if (!products.isEmpty()) {
                    for (Product p : products) {
                        System.out.print("Product ID: " + p.getProductID());
                        System.out.print(", Name: " + p.getProductName());
                        System.out.println(", Supplier: " + p.getSupplier());
                        System.out.println(", Category: " + p.getCategory());
                    }
                } else {
                    System.out.println("No products found.");
                }
                tx.commit();
                break;

            case 4:
                tx = session.beginTransaction();
                List<Supplier> suppliers = session.createQuery("FROM Supplier", Supplier.class).list();
```

```
                    if (!suppliers.isEmpty()) {
                        for (Supplier s : suppliers) {
                            System.out.print("Supplier ID: " + s.getSupplierID());
                            System.out.print(", Name: " + s.getCompanyName());
                            System.out.print(", Street: " + s.getStreet());
                            System.out.println(", City: " + s.getCity());
                            System.out.println("Products: " + s.getProducts());
                        }
                    } else {
                        System.out.println("No suppliers found.");
                    }
                    tx.commit();
                    break;
                case 5:
                    tx = session.beginTransaction();
                    List<Category> categories = session.createQuery("FROM Category ", Category.class).list();
                    if (!categories.isEmpty()) {
                        for (Category c : categories) {
                            System.out.print("Category ID: " + c.getCategoryID());
                            System.out.print(", Name: " + c.getName());
                            System.out.println(", Products: " + c.getProducts());
                        }
                    } else {
                        System.out.println("No categories found.");
                    }
                    tx.commit();
                    break;
                case 6:
                    tx = session.beginTransaction();
                    System.out.println("Podaj kategorię: ");
                    String categoryName = scanner.nextLine();
                    List<Product> filteredProducts = session.createQuery("FROM Product p WHERE p.category.name =
:categoryName", Product.class).setParameter("categoryName", categoryName).list();
                    if (!filteredProducts.isEmpty()) {
                        for (Product p : filteredProducts) {
                            System.out.print("Product ID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.println(", Supplier: " + p.getSupplier().getName());
                            System.out.println(", Category: " + p.getCategory().getName());
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;
                case 7:
                    tx = session.beginTransaction();
                    System.out.println("Podaj nazwe produktu którego chcesz poznać kategorię: ");
                    productName = scanner.nextLine();
                    Product specificProduct = session.createQuery("FROM Product p where p.productName =
:productName", Product.class).setParameter("productName", productName).uniqueResult();
                    if (specificProduct == null) {
                        System.out.println("Produkt nie znaleziony");
                    }
                    Category category = specificProduct.getCategory();
                    if (category == null) {
                        System.out.println("Kategoria nie znaleziona");
                    } else {
                        System.out.println("Categoria produktu to: " + category.getName());
                    }
                    tx.commit();
                    break;
                case 8:
                    session.close();
                    scanner.close();
                    System.out.println("Goodbye!");
                    return;

                default:
                    System.out.println("Invalid option, please try again.");
            }
        }
```

```
        }
    }
```

Products:

WHERE ORDER BY

	CATEGORYID	PRODUCTID	SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	1	20	Laptop
2	1	2	1	50	Smartphone
3	3	3	2	30	Tablet
4	2	4	2	15	Books
5	3	5	3	10	Car

Categories:

WHERE ORDER BY

	CATEGORYID	NAME
1	1	Devices
2	2	School
3	3	Vehicles

- SQL Logi po wykonaniu z main case 0:

```
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:
    /* insert for
        zad1.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Supplier */insert
```

```
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Category */insert
    into
        Category (name, categoryID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Category */insert
    into
        Category (name, categoryID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Category */insert
    into
        Category (name, categoryID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Product added successfully.
```

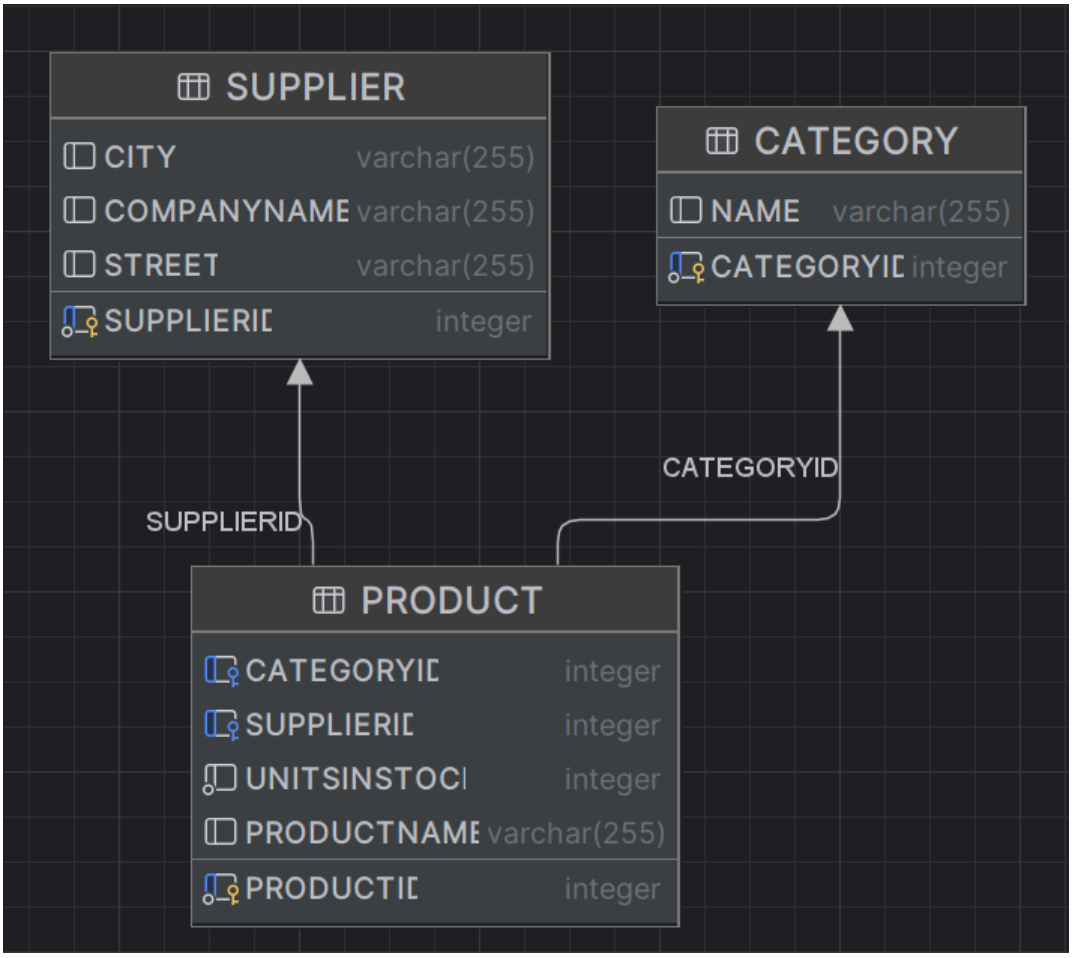
Produkty z wybranej kategorii:

```
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show suppliers
6. Show products with Category
7. Show Product Category
8. Exit
--------------------
6
Podaj kategorię:
Devices
Hibernate:
    /*
FROM
    Product p
WHERE
    p.category.name = :categoryName */ select
        p1_0.productID,
        p1_0.categoryID,
        p1_0.productName,
        p1_0.supplierID,
        p1_0.unitsInStock
    from
        Product p1_0
    join
        Category c1_0
            on c1_0.categoryID=p1_0.categoryID
    where
        c1_0.name=?
Product ID: 1, Name: Laptop, Supplier: Acme Corp, Category:
Devices
Product ID: 2, Name: Smartphone, Supplier: Acme Corp, Category:
Devices
Product ID: 3, Name: Tablet, Supplier: Tech Solutions,
Category: Vehicles
--------------------
```

Kategoria danego produktu:

```
--------------------
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show suppliers
6. Show products with Category
7. Show Product Category
8. Exit
--------------------
7
Podaj nazwe produktu którego chcesz poznać kategorię:
Car
Hibernate:
    /*
FROM
    Product p
where
    p.productName = :productName */ select
        p1_0.productID,
        p1_0.categoryID,
        p1_0.productName,
        p1_0.supplierID,
        p1_0.unitsInStock
    from
        Product p1_0
    where
        p1_0.productName=?
Categoria produktu to: Vehicles
```

Zoptymalizowany schemat bazy danych:



Zadanie 5 - modelowanie relacji wiele-do-wielu

- Tworzymy klasę Invoice reprezentującą fakturę konkretnego zamówienia, a następnie ustawiamy realcję wiele do wielu z tabelą Products.
- Modelujemy relacje dodając do klasy Products HashSet z adnotacją ManyToMany reprezentujący w jakich fakturach znajduje się dany produkt
- Product:

```
@Entity
@SequenceGenerator(name = "Products_SEQ")
class Product{

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Products_SEQ")
    private int productID;

    private String productName;
    private int unitsInStock;

    @ManyToOne
    @JoinColumn(name = "supplierID")
    private Supplier supplier;

    @ManyToOne
    @JoinColumn(name = "categoryID")
    private Category category;

    @ManyToMany(mappedBy = "products")
    private Set<Invoice> invoices = new HashSet<>();

    public Product() {}

    Product(String productName, int unitsInStock) {
        this.productName = productName;
        this.unitsInStock = unitsInStock;
    }

    int getProductID() {
        return productID;
    }

    String getProductName() {
        return productName;
    }

    int getUnitsInStock() {
        return unitsInStock;
    }

    public Supplier getSupplier() {
        return supplier;
    }

    public void setSupplier(Supplier supplier) {
        this.supplier = supplier;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    public Category getCategory() {
        return category;
    }

    @Override
    public String toString() {
        return "Product{" +
```

```
                "productName='" + productName + '\'' +
                '}';
    }

    public Set<Invoice> getInvoices() {
        return invoices;
    }

    public void sell(Invoice invoice, int quantity) {
        if (unitsInStock < quantity) {
            System.out.println("Unable to sell" + quantity + " products");
            return;
        }
        unitsInStock -= quantity;
        invoice.addProducts(this, quantity);
        invoices.add(invoice);
    }
}
```

Z kolei w klasie Invoice tworzymy HashSet z adnotacjami ManyToMany oraz JoinTable aby baza danych poprawnie powiązała relację ManyToMany tworząc pomocniczą tabelę Invoice_Products (jest to łącznik pomiędzy dwoma tabelami, który w relacjach ManyToMany jest niezbędny)

- Invoice

```
@Entity
@SequenceGenerator(name = "Invoice_SEQ")
public class Invoice {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "Invoice_SEQ")
    private int invoiceID;
    private int invoiceNumber;
    private int quantity = 0;

    @ManyToMany
    @JoinTable(
            name = "Invoice_Products",
            joinColumns = @JoinColumn(name = "invoiceID"),
            inverseJoinColumns = @JoinColumn(name = "productID")
    )
    private Set<Product> products = new HashSet<>();

    public Invoice() {
    }

    public Invoice(int invoiceNumber) {
        this.invoiceNumber = invoiceNumber;
    }

    public Set<Product> getProducts() {
        return products;
    }

    public void addProducts(Product product, int quantity) {
        products.add(product);
        this.quantity += quantity;
    }

    public int getInvoiceNumber() {
        return invoiceNumber;
    }

    public int getQuantity() {
        return quantity;
    }

    @Override
    public String toString() {
        return "Invoice{" +
                "invoiceNumber=" + invoiceNumber +
```

```
                '}';
    }
}
```

W klasie main dodaliśmy kilka Invoices, w ramach których zostałe sprzedane produkty.

- Main

```
class Main {
private static SessionFactory sessionFactory = null;

    private static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            Configuration configuration = new Configuration();
            sessionFactory = configuration.configure().buildSessionFactory();
        }
        return sessionFactory;
    }

    public static void main(String[] args) {

        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();

        Transaction tx;
        Scanner scanner = new Scanner(System.in);
        int option;

        while (true) {
            System.out.println("--------------------");
            System.out.println("What do you want to do?: ");
            System.out.println("0. Add template");
            System.out.println("1. Add product");
            System.out.println("2. Add supplier");
            System.out.println("3. Show products");
            System.out.println("4. Show suppliers");
            System.out.println("5. Show category");
            System.out.println("6. Show products with Category");
            System.out.println("7. Show Sold products on InvoiceID: ");
            System.out.println("8. Show Invoices that product had been sold: ");
            System.out.println("9. Show Product Category");
            System.out.println("10. Exit");
            System.out.println("--------------------");
            option = scanner.nextInt();
            scanner.nextLine();

            switch (option) {
                case 0:
                    tx = session.beginTransaction();

                    Supplier supplier1 = new Supplier("Acme Corp", "123 Main St", "Springfield");
                    Supplier supplier2 = new Supplier("Tech Solutions", "456 Elm St", "Shelbyville");
                    Supplier supplier3 = new Supplier("Global Traders", "789 Oak St", "Capital City");

                    Product product1 = new Product("Laptop", 20);
                    Product product2 = new Product("Smartphone", 50);
                    Product product3 = new Product("Tablet", 30);
                    Product product4 = new Product("Books", 15);
                    Product product5 = new Product("Car", 10);

                    Category category1 = new Category("Devices");
                    Category category2 = new Category("School");
                    Category category3 = new Category("Vehicles");

                    Invoice invoice1 = new Invoice(1001);
                    Invoice invoice2 = new Invoice(1002);
                    int soldNumber1 = 4;
                    int soldNumber2 = 6;
                    int soldNumber3 = 8;
```

```
                product1.sell(invoice1, soldNumber1);
                product2.sell(invoice1, soldNumber2);
                product3.sell(invoice1, soldNumber3);
                product4.sell(invoice2, soldNumber3);
                product3.sell(invoice1, soldNumber1);


                product1.setSupplier(supplier1);
                product1.setCategory(category1);

                product2.setSupplier(supplier1);
                product2.setCategory(category1);

                product3.setSupplier(supplier2);
                product3.setCategory(category1);

                product4.setSupplier(supplier2);
                product4.setCategory(category2);

                product5.setSupplier(supplier3);
                product5.setCategory(category3);


                supplier1.addProduct(product1);
                supplier1.addProduct(product2);

                supplier2.addProduct(product3);
                supplier2.addProduct(product4);

                supplier3.addProduct(product5);

                category1.addProducts(product1);
                category1.addProducts(product2);
                category1.addProducts(product3);

                category2.addProducts(product4);

                category3.addProducts(product5);

                session.save(supplier1);
                session.save(supplier2);
                session.save(supplier3);

                session.save(category1);
                session.save(category2);
                session.save(category3);

                session.save(product1);
                session.save(product2);
                session.save(product3);
                session.save(product4);
                session.save(product5);

                session.save(invoice1);
                session.save(invoice2);


                tx.commit();
                System.out.println("Product added successfully.");
                break;
            case 1:
                tx = session.beginTransaction();
                System.out.print("Enter product name: ");
                String productName = scanner.nextLine();
                System.out.print("Enter units in stock: ");
                int unitsInStock = scanner.nextInt();
                Product product = new Product(productName, unitsInStock);
                session.save(product);
                tx.commit();
                System.out.println("Product added successfully.");
                break;
```

```
                case 2:
                    tx = session.beginTransaction();
                    System.out.print("Enter supplier name: ");
                    String supplierName = scanner.nextLine();
                    System.out.print("Enter supplier street: ");
                    String supplierRoad = scanner.nextLine();
                    System.out.print("Enter supplier city: ");
                    String supplierCity = scanner.nextLine();

                    Supplier supplier = new Supplier(supplierName, supplierRoad, supplierCity);
                    session.save(supplier);
                    tx.commit();
                    System.out.println("Supplier added successfully.");
                    break;
                case 3:
                    tx = session.beginTransaction();
                    List<Product> products = session.createQuery("FROM Product", Product.class).list();
                    if (!products.isEmpty()) {
                        for (Product p : products) {
                            System.out.print("Product ID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.println(", Supplier: " + p.getSupplier());
                            System.out.println(", Category: " + p.getCategory());
                            System.out.println(", Invoices: " + p.getInvoices());
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;

                case 4:
                    tx = session.beginTransaction();
                    List<Supplier> suppliers = session.createQuery("FROM Supplier", Supplier.class).list();
                    if (!suppliers.isEmpty()) {
                        for (Supplier s : suppliers) {
                            System.out.print("Supplier ID: " + s.getSupplierID());
                            System.out.print(", Name: " + s.getCompanyName());
                            System.out.print(", Street: " + s.getStreet());
                            System.out.println(", City: " + s.getCity());
                            System.out.println("Products: " + s.getProducts());
                        }
                    } else {
                        System.out.println("No suppliers found.");
                    }
                    tx.commit();
                    break;
                case 5:
                    tx = session.beginTransaction();
                    List<Category> categories = session.createQuery("FROM Category ", Category.class).list();
                    if (!categories.isEmpty()) {
                        for (Category c : categories) {
                            System.out.print("Category ID: " + c.getCategoryID());
                            System.out.print(", Name: " + c.getName());
                            System.out.println(", Products: " + c.getProducts());
                        }
                    } else {
                        System.out.println("No categories found.");
                    }
                    tx.commit();
                    break;
                case 6:
                    tx = session.beginTransaction();
                    System.out.println("Category Name: ");
                    String categoryName = scanner.nextLine();
                    List<Product> filteredProducts = session.createQuery("FROM Product p WHERE p.category.name =
:categoryName", Product.class).setParameter("categoryName", categoryName).list();
                    if (!filteredProducts.isEmpty()) {
                        for (Product p : filteredProducts) {
                            System.out.print("Product ID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.println(", Supplier: " + p.getSupplier());
```

```
                            System.out.println(", Category: " + p.getCategory());
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;
                case 7:
                    tx = session.beginTransaction();
                    System.out.println("Invoice Number: ");
                    String invoiceNumber = scanner.nextLine();
                    Invoice invoice = session.createQuery("FROM Invoice inv where inv.invoiceNumber =
:invoiceNumber", Invoice.class).setParameter("invoiceNumber", invoiceNumber).uniqueResult();
                    if (invoice != null) {
                        for (Product p : invoice.getProducts()) {
                            System.out.print("ProductID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.print(", Supplier: " + p.getSupplier());
                            System.out.println(", Category: " + p.getCategory());
                        }
                    } else {
                        System.out.println("No invoice found.");
                    }
                    tx.commit();
                    break;
                case 8:
                    tx = session.beginTransaction();
                    System.out.println("Product Name: ");
                    productName = scanner.nextLine();
                    products = session.createQuery("FROM Product p WHERE p.productName = :productName",
Product.class)
                            .setParameter("productName", productName)
                            .getResultList();
                    if (!products.isEmpty()) {
                        for (Product p : products) {
                            System.out.println("Product: " + p.getProductName() + " appears in the following
invoices:");
                            for (Invoice inv : p.getInvoices()) {
                                System.out.println("Invoice Number: " + inv.getInvoiceNumber());
                            }
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;
                case 9:
                    tx = session.beginTransaction();
                    System.out.println("Product name you want to know category: ");
                    productName = scanner.nextLine();
                    Product specificProduct = session.createQuery("FROM Product p where p.productName =
:productName", Product.class).setParameter("productName", productName).uniqueResult();
                    if (specificProduct == null) {
                        System.out.println("Product not found");
                    }
                    Category category = specificProduct.getCategory();
                    if (category == null) {
                        System.out.println("Category not found");
                    } else {
                        System.out.println("Product Category: : " + category.getName());
                    }
                    tx.commit();
                    break;
                case 10:
                    session.close();
                    scanner.close();
                    System.out.println("Goodbye!");
                    return;

                default:
                    System.out.println("Invalid option, please try again.");
            }
```

```
        }
    }
}
```

- SQL Logi:

```
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:
    /* insert for
        zad1.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Category */insert
    into
        Category (name, categoryID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Category */insert
    into
```

```
        Category (name, categoryID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Category */insert
    into
        Category (name, categoryID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Product */insert
    into
        Product (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    /* insert for
        zad1.Invoice */insert
    into
        Invoice (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        zad1.Invoice */insert
    into
        Invoice (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        zad1.Invoice.products */insert
    into
        Invoice_Products (invoiceID, productID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Invoice.products */insert
    into
        Invoice_Products (invoiceID, productID)
    values
```

```
        (?, ?)
Hibernate:
    /* insert for
        zad1.Invoice.products */insert
    into
        Invoice_Products (invoiceID, productID)
    values
        (?, ?)
Hibernate:
    /* insert for
        zad1.Invoice.products */insert
    into
        Invoice_Products (invoiceID, productID)
    values
        (?, ?)
Product added successfully.
```

Products:

WHERE ORDER BY

	CATEGORYID	PRODUCTID	SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	1	16	Laptop
2	1	2	1	44	Smartphone
3	1	3	2	18	Tablet
4	2	4	2	7	Books
5	3	5	3	10	Car

Invoices:

	INVOICEID	INVOICENUMBER	QUANTITY
1	1	1001	22
2	2	1002	8

InvoiceProducts:

	INVOICEID	PRODUCTID
1	1	1
2	1	2
3	1	3
4	2	4

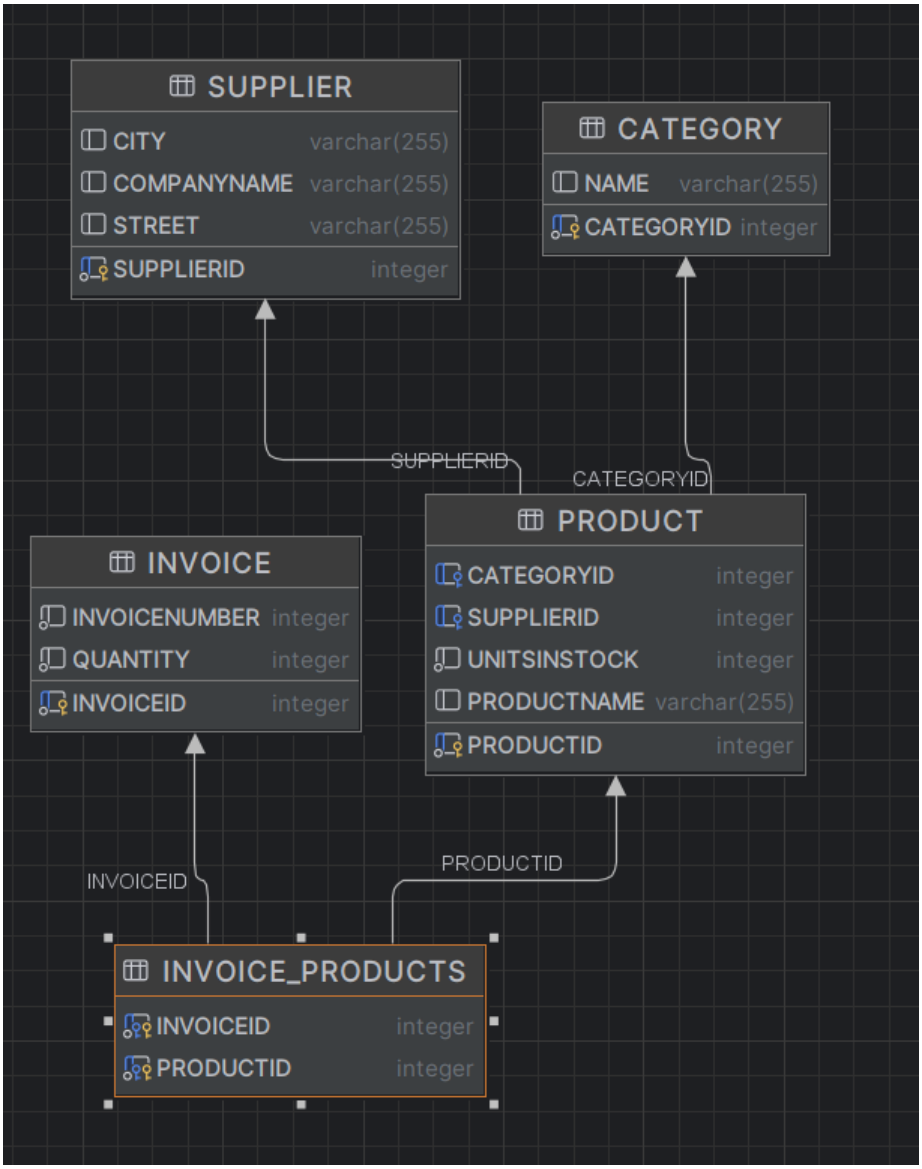
Produkty na fakturze o konkretnym numerze:

```
--------------------
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
7
Invoice Number:
1001
Hibernate:
    /*
FROM
    Invoice inv
where
    inv.invoiceNumber = :invoiceNumber */ select
        i1_0.invoiceID,
        i1_0.invoiceNumber,
        i1_0.quantity
    from
        Invoice i1_0
    where
        i1_0.invoiceNumber=?
ProductID: 1, Name: Laptop, Supplier: Acme Corp, Category: Devices
ProductID: 3, Name: Tablet, Supplier: Tech Solutions, Category: Devices
ProductID: 2, Name: Smartphone, Supplier: Acme Corp, Category: Devices
```

Faktury na których został sprzedany dany produkt:

```
--------------------
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
8
Product Name:
Tablet
Hibernate:
    /*
FROM
    Product p
WHERE
    p.productName = :productName */ select
        p1_0.productID,
        p1_0.categoryID,
        p1_0.productName,
        p1_0.supplierID,
        p1_0.unitsInStock
    from
        Product p1_0
    where
        p1_0.productName=?
Product: Tablet appears in the following invoices:
Invoice Number: 1001
--------------------
```

Schemat bazy danych:



Zadanie 6 - modelowanie relacji wiele-do-wielu: JPA

Stworzyliśmy nowego Maina i nieco zmieniliśmy kod poszczególnych klas i konfiguracji, aby mogły one współpracować z technologią JPA. Klasy Category oraz Supplier pozostają bez zmian.

- Invoice (fragment starego kodu)

```
...
    @ManyToMany
    @JoinTable(
            name = "Invoice_Products",
            joinColumns = @JoinColumn(name = "invoiceID"),
            inverseJoinColumns = @JoinColumn(name = "productID")
    )
    private Set<Product> products = new HashSet<>();
...
```

- Inovice (fragment nowego kodu)

```
...
    @ManyToMany(cascade = CascadeType.PERSIST)
    private Set<Product> products = new HashSet<>();
...
```

- Product (fragment starego kodu)

```
...
    @ManyToOne
    @JoinColumn(name = "supplierID")
    private Supplier supplier;

    @ManyToOne
    @JoinColumn(name = "categoryID")
    private Category category;

    @ManyToMany(mappedBy = "products")
    private Set<Invoice> invoices = new HashSet<>();
...
```

- Product (fragment nowego kodu)

```
    @ManyToOne(cascade = CascadeType.PERSIST)
    private Supplier supplier;

    @ManyToOne(cascade = CascadeType.PERSIST)
    private Category category;

    @ManyToMany(mappedBy = "products")
    private Set<Invoice> invoices = new HashSet<>();
```

- MainJPA

```
package zad1;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Persistence;
import org.hibernate.Transaction;
```

```java
import org.hibernate.cfg.Configuration;

import java.util.List;
import java.util.Scanner;
import java.util.Set;


class MainJPA {
    private static final EntityManagerFactory emf;

    static {
        try {
            emf = Persistence.createEntityManagerFactory("derby");
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public static void main(String[] args) {
        final EntityManager entityManager = getEntityManager();
        EntityTransaction tx;

        Scanner scanner = new Scanner(System.in);
        int option;

        while (true) {
            System.out.println("--------------------");
            System.out.println("What do you want to do?: ");
            System.out.println("0. Add template");
            System.out.println("1. Add product");
            System.out.println("2. Add supplier");
            System.out.println("3. Show products");
            System.out.println("4. Show suppliers");
            System.out.println("5. Show category");
            System.out.println("6. Show products with Category");
            System.out.println("7. Show Sold products on InvoiceID: ");
            System.out.println("8. Show Invoices that product had been sold: ");
            System.out.println("9. Show Product Category");
            System.out.println("10. Exit");
            System.out.println("--------------------");
            option = scanner.nextInt();
            scanner.nextLine();

            switch (option) {
                case 0:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    Supplier supplier1 = new Supplier("Acme Corp", "123 Main St", "Springfield");
                    Supplier supplier2 = new Supplier("Tech Solutions", "456 Elm St", "Shelbyville");
                    Supplier supplier3 = new Supplier("Global Traders", "789 Oak St", "Capital City");

                    Product product1 = new Product("Laptop", 20);
                    Product product2 = new Product("Smartphone", 50);
                    Product product3 = new Product("Tablet", 30);
                    Product product4 = new Product("Books", 15);
                    Product product5 = new Product("Car", 10);

                    Category category1 = new Category("Devices");
                    Category category2 = new Category("School");
                    Category category3 = new Category("Vehicles");

                    Invoice invoice1 = new Invoice(1001);
                    Invoice invoice2 = new Invoice(1002);
                    int soldNumber1 = 4;
                    int soldNumber2 = 6;
                    int soldNumber3 = 8;

                    product1.sell(invoice1, soldNumber1);
```

```
                    product2.sell(invoice1, soldNumber2);
                    product3.sell(invoice1, soldNumber3);
                    product4.sell(invoice2, soldNumber3);
                    product3.sell(invoice1, soldNumber1);


                    product1.setSupplier(supplier1);
                    product1.setCategory(category1);

                    product2.setSupplier(supplier1);
                    product2.setCategory(category1);

                    product3.setSupplier(supplier2);
                    product3.setCategory(category1);

                    product4.setSupplier(supplier2);
                    product4.setCategory(category2);

                    product5.setSupplier(supplier3);
                    product5.setCategory(category3);


                    supplier1.addProduct(product1);
                    supplier1.addProduct(product2);

                    supplier2.addProduct(product3);
                    supplier2.addProduct(product4);

                    supplier3.addProduct(product5);

                    category1.addProducts(product1);
                    category1.addProducts(product2);
                    category1.addProducts(product3);

                    category2.addProducts(product4);

                    category3.addProducts(product5);

                    entityManager.persist(supplier1);
                    entityManager.persist(supplier2);
                    entityManager.persist(supplier3);

                    entityManager.persist(category1);
                    entityManager.persist(category2);
                    entityManager.persist(category3);

                    entityManager.persist(product1);
                    entityManager.persist(product2);
                    entityManager.persist(product3);
                    entityManager.persist(product4);
                    entityManager.persist(product5);

                    entityManager.persist(invoice1);
                    entityManager.persist(invoice2);


                    tx.commit();
                    System.out.println("Product added successfully.");
                    break;
                case 1:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    System.out.print("Enter product name: ");
                    String productName = scanner.nextLine();
                    System.out.print("Enter units in stock: ");
                    int unitsInStock = scanner.nextInt();
                    Product product = new Product(productName, unitsInStock);
                    entityManager.persist(product);
                    tx.commit();
                    System.out.println("Product added successfully.");
                    break;
```

```
                case 2:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    System.out.print("Enter supplier name: ");
                    String supplierName = scanner.nextLine();
                    System.out.print("Enter supplier street: ");
                    String supplierRoad = scanner.nextLine();
                    System.out.print("Enter supplier city: ");
                    String supplierCity = scanner.nextLine();

                    Supplier supplier = new Supplier(supplierName, supplierRoad, supplierCity);
                    entityManager.persist(supplier);
                    tx.commit();
                    System.out.println("Supplier added successfully.");
                    break;
                case 3:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    List<Product> products = entityManager.createQuery("FROM Product",
Product.class).getResultList();
                    if (!products.isEmpty()) {
                        for (Product p : products) {
                            System.out.println("Product ID: " + p.getProductID());
                            System.out.println("Name: " + p.getProductName());
                            System.out.println("Supplier: " + p.getSupplier());
                            System.out.println("Category: " + p.getCategory());
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;

                case 4:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    List<Supplier> suppliers = entityManager.createQuery("FROM Supplier",
Supplier.class).getResultList();
                    if (!suppliers.isEmpty()) {
                        for (Supplier s : suppliers) {
                            System.out.print("Supplier ID: " + s.getSupplierID());
                            System.out.print(", Name: " + s.getCompanyName());
                            System.out.print(", Street: " + s.getStreet());
                            System.out.println(", City: " + s.getCity());
                            System.out.println("Products: " + s.getProducts());
                        }
                    } else {
                        System.out.println("No suppliers found.");
                    }
                    tx.commit();
                    break;
                case 5:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    List<Category> categories = entityManager.createQuery("FROM Category ",
Category.class).getResultList();
                    if (!categories.isEmpty()) {
                        for (Category c : categories) {
                            System.out.print("Category ID: " + c.getCategoryID());
                            System.out.print(", Name: " + c.getName());
                            System.out.println(", Products: " + c.getProducts());
                        }
                    } else {
                        System.out.println("No categories found.");
                    }
                    tx.commit();
                    break;
```

```
                case 6:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    System.out.println("Category Name: ");
                    String categoryName = scanner.nextLine();
                    List<Product> filteredProducts = entityManager.createQuery("FROM Product p WHERE
p.category.name = :categoryName", Product.class).setParameter("categoryName", categoryName).getResultList();
                    if (!filteredProducts.isEmpty()) {
                        for (Product p : filteredProducts) {
                            System.out.print("Product ID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.println(", Supplier: " + p.getSupplier());
                            System.out.println(", Category: " + p.getCategory());
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;
                case 7:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    System.out.println("Invoice Number: ");
                    String invoiceNumber = scanner.nextLine();
                    Invoice invoice = entityManager.createQuery("FROM Invoice inv where inv.invoiceNumber =
:invoiceNumber", Invoice.class).setParameter("invoiceNumber", invoiceNumber).getSingleResult();
                    if (invoice != null) {
                        for (Product p : invoice.getProducts()) {
                            System.out.print("ProductID: " + p.getProductID());
                            System.out.print(", Name: " + p.getProductName());
                            System.out.print(", Supplier: " + p.getSupplier());
                            System.out.println(", Category: " + p.getCategory());
                        }
                    } else {
                        System.out.println("No invoice found.");
                    }
                    tx.commit();
                    break;
                case 8:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    System.out.println("Product Name: ");
                    productName = scanner.nextLine();
                    products = entityManager.createQuery("FROM Product p WHERE p.productName = :productName",
Product.class)
                            .setParameter("productName", productName)
                            .getResultList();
                    if (!products.isEmpty()) {
                        for (Product p : products) {
                            System.out.println("Product: " + p.getProductName() + " appears in the following
invoices:");
                            for (Invoice inv : p.getInvoices()) {
                                System.out.println("Invoice Number: " + inv.getInvoiceNumber());
                            }
                        }
                    } else {
                        System.out.println("No products found.");
                    }
                    tx.commit();
                    break;
                case 9:
                    tx = entityManager.getTransaction();
                    tx.begin();

                    System.out.println("Product name you want to know category: ");
                    productName = scanner.nextLine();
                    Product specificProduct = entityManager.createQuery("FROM Product p where p.productName =
:productName", Product.class).setParameter("productName", productName).getSingleResult();
                    if (specificProduct == null) {
```

```
                        System.out.println("Product not found");
                    }
                    Category category = specificProduct.getCategory();
                    if (category == null) {
                        System.out.println("Category not found");
                    } else {
                        System.out.println("Product Category: : " + category.getName());
                    }
                    tx.commit();
                    break;
                case 10:
                    entityManager.close();
                    scanner.close();
                    System.out.println("Goodbye!");
                    return;

                default:
                    System.out.println("Invalid option, please try again.");
            }
        }
    }
}
```

Do JPA również był potrzebny nowy plik konfiguracyjny:

- persistence.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2)0.xsd" version="2.0">
    <persistence-unit name="derby" transaction-type="RESOURCE_LOCAL">
        <properties>
            <property name="hibernate.dialect"
                      value="org.hibernate.dialect.DerbyTenSevenDialect" />
            <property name="hibernate.connection.driver_class"
                      value="org.apache.derby.jdbc.ClientDriver"/>
            <property name="hibernate.connection.url"
                      value="jdbc:derby://127.0.0.1/StasKochevenkoJPA"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
        </properties>
    </persistence-unit>
</persistence>
```

- SQL Logi:

```
Hibernate:
    create sequence Categories_SEQ start with 1 increment by 50
Hibernate:
    create sequence Invoice_SEQ start with 1 increment by 50
Hibernate:
    create sequence Products_SEQ start with 1 increment by 50
Hibernate:
    create sequence Supplier_SEQ start with 1 increment by 50
Hibernate:
    create table Category (
        categoryID integer not null,
        name varchar(255),
        primary key (categoryID)
    )
Hibernate:
    create table Invoice (
        invoiceID integer not null,
        invoiceNumber integer not null,
        quantity integer not null,
```

```
        primary key (invoiceID)
    )
Hibernate:
    create table Invoice_Product (
        invoices_invoiceID integer not null,
        products_productID integer not null,
        primary key (invoices_invoiceID, products_productID)
    )
Hibernate:
    create table Product (
        category_categoryID integer,
        productID integer not null,
        supplier_supplierID integer,
        unitsInStock integer not null,
        productName varchar(255),
        primary key (productID)
    )
Hibernate:
    create table Supplier (
        supplierID integer not null,
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        primary key (supplierID)
    )
Hibernate:
    alter table Invoice_Product
       add constraint FK2mn08nt19nrqagr12grh5uho0
       foreign key (products_productID)
       references Product
Hibernate:
    alter table Invoice_Product
       add constraint FKthlx5t7fv55fgsfp7ivt4fj6u
       foreign key (invoices_invoiceID)
       references Invoice
Hibernate:
    alter table Product
       add constraint FK987q0koesbyk7oqky7lg431xr
       foreign key (category_categoryID)
       references Category
Hibernate:
    alter table Product
       add constraint FK6em1ebdcyqbgxmglei6wanchp
       foreign key (supplier_supplierID)
       references Supplier
--------------------
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
0
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Categories_SEQ
```

```
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
```

```
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Product added successfully.
```

Products:

	CATEGORY_CATEGORYID	PRODUCTID	SUPPLIER_SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	1	16	Laptop
2	1	2	1	44	Smartphone
3	1	3	2	18	Tablet
4	2	4	2	7	Books
5	3	5	3	10	Car

Invoices:

	INVOICEID	INVOICENUMBER	QUANTITY
1	1	1001	22
2	2	1002	8

InvoiceProducts:

	INVOICES_INVOICEID	PRODUCTS_PRODUCTID
1	1	1
2	1	2
3	1	3
4	2	4

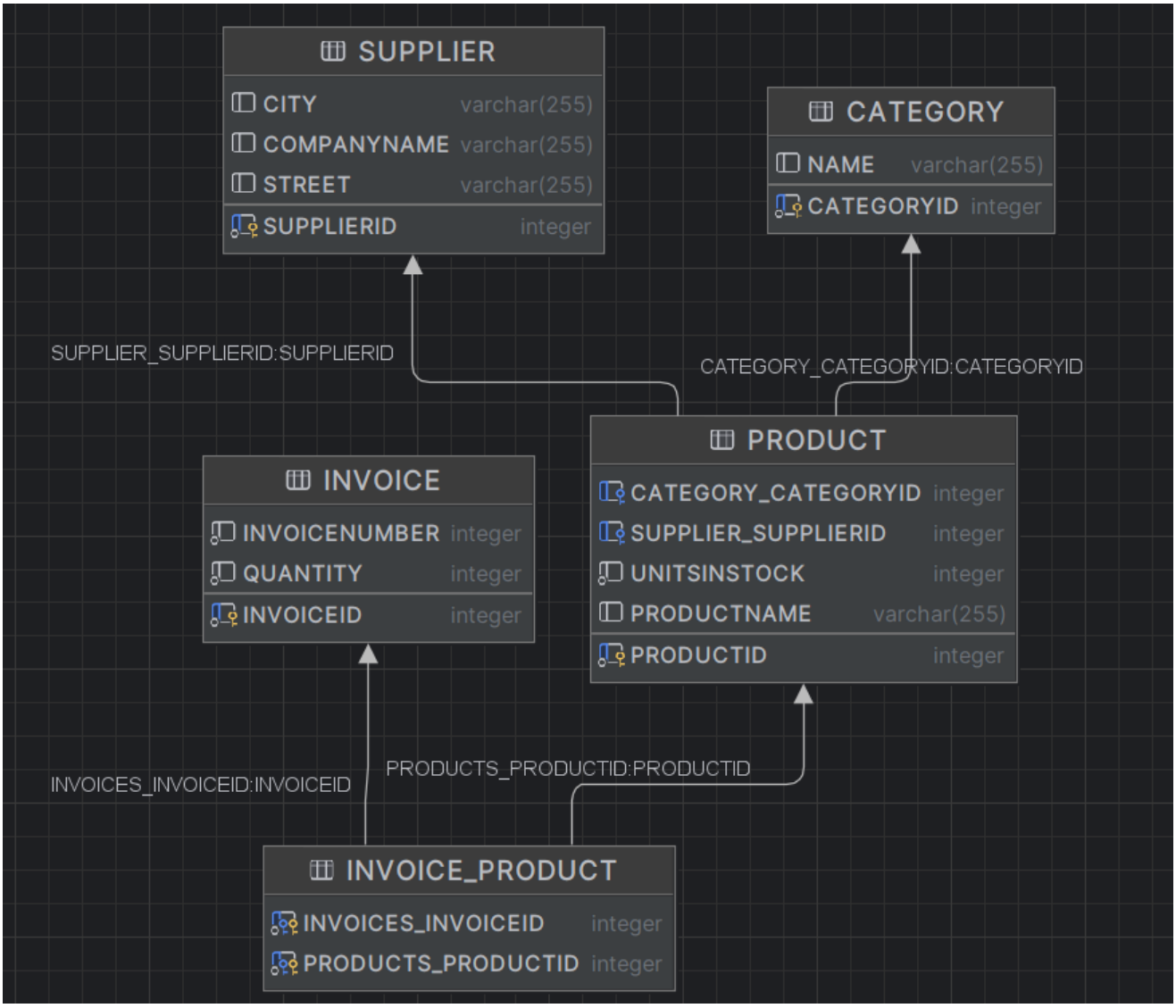
Produkty na fakturze o konkretnym numerze:

```
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
7
Invoice Number:
1001
Hibernate:
    select
        i1_0.invoiceID,
        i1_0.invoiceNumber,
        i1_0.quantity
    from
        Invoice i1_0
    where
        i1_0.invoiceNumber=?
ProductID: 3, Name: Tablet, Supplier: Tech Solutions, Category: Devices
ProductID: 1, Name: Laptop, Supplier: Acme Corp, Category: Devices
ProductID: 2, Name: Smartphone, Supplier: Acme Corp, Category: Devices
```

Faktury na których został sprzedany dany produkt:

```
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
8
Product Name:
Tablet
Hibernate:
    select
        p1_0.productID,
        p1_0.category_categoryID,
        p1_0.productName,
        p1_0.supplier_supplierID,
        p1_0.unitsInStock
    from
        Product p1_0
    where
        p1_0.productName=?
Product: Tablet appears in the following invoices:
Invoice Number: 1001
```

Schemat bazy danych:



Jak widać, schemat nie różni się od pokazanego w zadaniu 5 (chyba że nazwami niektórych elementów).

Zadanie 7 - kaskady

Kaskadowe operację przewidywaliśmy jeszcze w rozwiązaniu poprzedniego zadania. W danym punkcie do zamodelowania kaskadowego tworzenia faktur wraz z nowymi produktami, oraz produktów wraz z nową fakturą zmieniliśmy jedynie dekorator @ManyToMany w obydwu klasach i przystosowaliśmy metodę sprzedającą oraz metodę Main pod nowy model.

Wykorzystanie mechanizmu kaskadowego spowoduje, ze przy próbie utrwalenia Invoice'a kaskadowo zostaną utrwaleni wszyscy nieutrwaleni jeszcze Produkty powiązani relacją z tym Invoicem, i odwrotnie (skoro ustawiliśmy to w kodzie klasie).

- Invoice (fragment starego kodu)

```
...
@ManyToMany(cascade = CascadeType.PERSIST)
private Set<Product> products = new HashSet<>();
...
```

- Invoice (fragment nowego kodu)

```
...
@ManyToMany(cascade = CascadeType.PERSIST, fetch = FetchType.EAGER)
private Set<Product> products = new HashSet<>();
...
```

- Product (fragment starego kodu)

```
...
@ManyToMany(mappedBy = "products")
private Set<Invoice> invoices = new HashSet<>();
...
```

- Product (fragment nowego kodu)

```
...
@ManyToMany(
    cascade = CascadeType.PERSIST,
    fetch = FetchType.EAGER,
    mappedBy = "products"
)
private Set<Invoice> invoices = new HashSet<>();
...
```

- Main (fragment starego kodu)

```
entityManager.persist(product1);
entityManager.persist(product2);
entityManager.persist(product3);
entityManager.persist(product4);
entityManager.persist(product5);

entityManager.persist(invoice1);
entityManager.persist(invoice2);
```

Spróbujmy przetestować zapisywanie do bazy nowych Invoce`ów nie ex plicite, a przez dodawanie ich w ramach Products

Kod dla takiego testu:

- Main (fragment nowego kodu)

```
        entityManager.persist(product1);
        entityManager.persist(product2);
        entityManager.persist(product3);
        entityManager.persist(product4);
        entityManager.persist(product5);
```

Products:

	CATEGORY_CATEGORYID	PRODUCTID	SUPPLIER_SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	1	16	Laptop
2	1	2	1	44	Smartphone
3	1	3	2	18	Tablet
4	2	4	2	7	Books
5	3	5	3	10	Car

Invoices:

	INVOICEID	INVOICENUMBER	QUANTITY
1	1	1001	22
2	2	1002	8

InvoiceProducts:

	INVOICES_INVOICEID	PRODUCTS_PRODUCTID
1	1	1
2	1	2
3	1	3
4	2	4

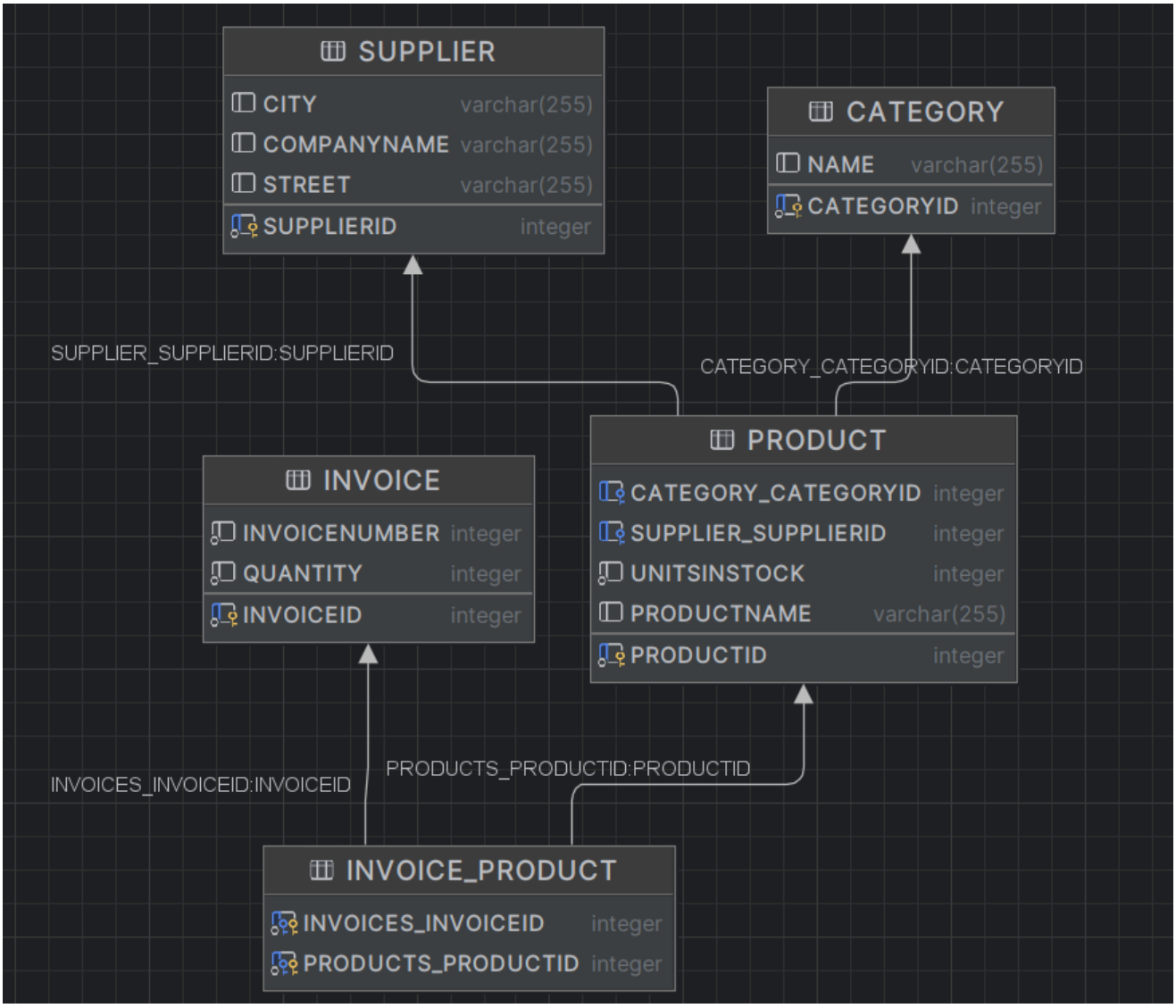
Produkty na fakturze o konkretnym numerze:

```
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
7
Invoice Number:
1001
Hibernate:
    select
        i1_0.invoiceID,
        i1_0.invoiceNumber,
        i1_0.quantity
    from
        Invoice i1_0
    where
        i1_0.invoiceNumber=?
ProductID: 3, Name: Tablet, Supplier: Tech Solutions, Category: Devices
ProductID: 1, Name: Laptop, Supplier: Acme Corp, Category: Devices
ProductID: 2, Name: Smartphone, Supplier: Acme Corp, Category: Devices
```

Faktury na których został sprzedany dany produkt:

```
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
8
Product Name:
Tablet
Hibernate:
    select
        p1_0.productID,
        p1_0.category_categoryID,
        p1_0.productName,
        p1_0.supplier_supplierID,
        p1_0.unitsInStock
    from
        Product p1_0
    where
        p1_0.productName=?
Product: Tablet appears in the following invoices:
Invoice Number: 1001
```

Schemat bazy danych:



Spróbujmy przetestować zapisywanie do bazy nowych produktów nie ex plicite, a przez dodawanie ich do Invoices

Kod dla takiego testu:

- Main (fragment nowego kodu)

```
entityManager.persist(invoice1);
entityManager.persist(invoice2);
```

Products:

	CATEGORY_CATEGORYID	PRODUCTID	SUPPLIER_SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	1	16	Laptop
2	1	2	1	44	Smartphone
3	1	3	2	18	Tablet
4	2	4	2	7	Books
5	3	5	3	10	Car

Invoices:

	INVOICEID	INVOICENUMBER	QUANTITY
1	1	1001	22
2	2	1002	8

InvoiceProducts:

	INVOICES_INVOICEID	PRODUCTS_PRODUCTID
1	1	1
2	1	2
3	1	3
4	2	4

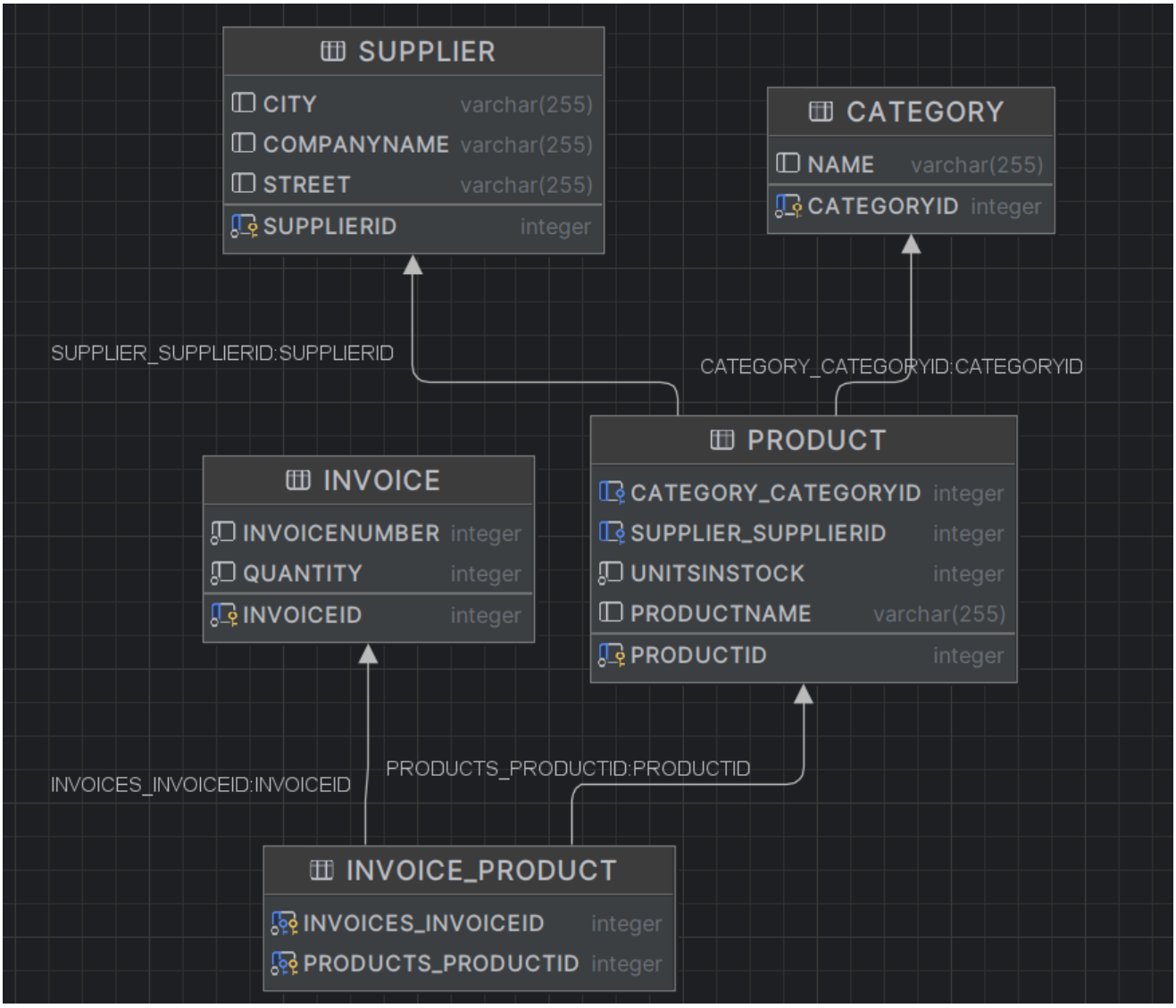
Produkty na fakturze o konkretnym numerze:

```
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
7
Invoice Number:
1001
Hibernate:
    select
        i1_0.invoiceID,
        i1_0.invoiceNumber,
        i1_0.quantity
    from
        Invoice i1_0
    where
        i1_0.invoiceNumber=?
ProductID: 3, Name: Tablet, Supplier: Tech Solutions, Category: Devices
ProductID: 1, Name: Laptop, Supplier: Acme Corp, Category: Devices
ProductID: 2, Name: Smartphone, Supplier: Acme Corp, Category: Devices
```

Faktury na których został sprzedany dany produkt:

```
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
8
Product Name:
Tablet
Hibernate:
    select
        p1_0.productID,
        p1_0.category_categoryID,
        p1_0.productName,
        p1_0.supplier_supplierID,
        p1_0.unitsInStock
    from
        Product p1_0
    where
        p1_0.productName=?
Product: Tablet appears in the following invoices:
Invoice Number: 1001
```

Schemat bazy danych:



W wyniku obydwu testów dostaliśmy takie same dane pozycje, jak w poprzednim zadaniu.

- SQL logi

```
Hibernate:
    create sequence Categories_SEQ start with 1 increment by 50
Hibernate:
    create sequence Invoice_SEQ start with 1 increment by 50
Hibernate:
    create sequence Products_SEQ start with 1 increment by 50
Hibernate:
    create sequence Supplier_SEQ start with 1 increment by 50
Hibernate:
    create table Category (
        categoryID integer not null,
        name varchar(255),
        primary key (categoryID)
    )
Hibernate:
    create table Invoice (
        invoiceID integer not null,
        invoiceNumber integer not null,
        quantity integer not null,
        primary key (invoiceID)
    )
Hibernate:
    create table Invoice_Product (
        invoices_invoiceID integer not null,
```

```
        products_productID integer not null,
        primary key (invoices_invoiceID, products_productID)
    )
Hibernate:
    create table Product (
        category_categoryID integer,
        productID integer not null,
        supplier_supplierID integer,
        unitsInStock integer not null,
        productName varchar(255),
        primary key (productID)
    )
Hibernate:
    create table Supplier (
        supplierID integer not null,
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        primary key (supplierID)
    )
Hibernate:
    alter table Invoice_Product
       add constraint FK2mn08nt19nrqagr12grh5uho0
       foreign key (products_productID)
       references Product
Hibernate:
    alter table Invoice_Product
       add constraint FKthlx5t7fv55fgsfp7ivt4fj6u
       foreign key (invoices_invoiceID)
       references Invoice
Hibernate:
    alter table Product
       add constraint FK987q0koesbyk7oqky7lg431xr
       foreign key (category_categoryID)
       references Category
Hibernate:
    alter table Product
       add constraint FK6em1ebdcyqbgxmglei6wanchp
       foreign key (supplier_supplierID)
       references Supplier
--------------------
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
0
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:
```

```
values
    next value for Invoice_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
```

```
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Product added successfully.
--------------------
```

Zadanie 8 - embedded class

a-b) dodanie "wbudowanego" adresu w klasę dostawcy

Została stworzona klasa Address, którą "wbudowaliśmy" do klasy Supplier przy pomocy podejścia Embedded classes.

- Address

```
package zad1;

import jakarta.persistence.*;

@Embeddable
class Address {

    private String street;
    private String city;

    public Address() {}

    Address(String street, String city) {
        this.street = street;
        this.city = city;
    }

    String getStreet() {
        return street;
    }
    String getCity() {
        return city;
    }

    public void setStreet(String street) {
        this.street = street;
    }
    public void setCity(String city) {
        this.city = city;
    }
    @Override
    public String toString() { return getCity() + ", " + getStreet(); }
}
```

- Supplier

```
package zad1;

import jakarta.persistence.*;

import java.util.ArrayList;
import java.util.List;

@Entity
@SequenceGenerator(name = "Supplier_SEQ")
class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Supplier_SEQ")
    private int supplierID;

    private String companyName;
    @Embedded
    private Address address;

    @OneToMany(mappedBy = "supplier")
    private List<Product> products = new ArrayList<>();
```

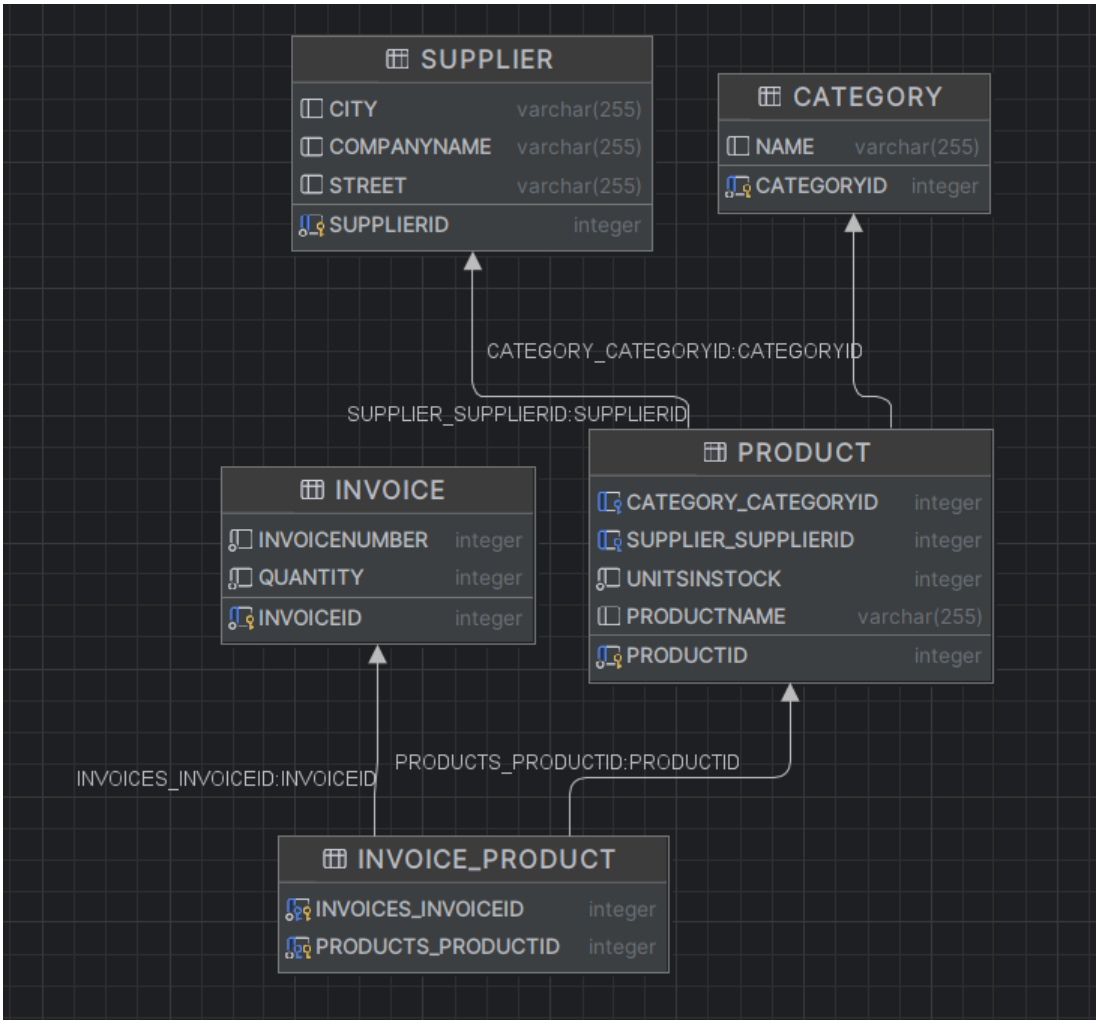
```java
    public Supplier() {}

    Supplier(String companyName, String street, String city) {
        this.companyName = companyName;
        this.address = new Address(street, city);
    }

    Supplier(String companyName, Address address) {
        this.companyName = companyName;
        this.address = address;
    }

    int getSupplierID() {
        return supplierID;
    }
    String getCompanyName() {
        return companyName;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
    public Address getAddress() {
        return address;
    }

    List<Product> getProducts() {
        return products;
    }
    void addProduct(Product product) {
        this.products.add(product);
    }

    @Override
    public String toString() { return companyName; }
}
```

Jak widać, zmieniliśmy konstruktory dla klasy Supplier, dzięki czemu nie musieliśmy zmieniać funkcji Main. Zobaczmy jak zmiany są reprezentowane w bazie na przykładowym zestawie danych jak wcześniej.

Suppliers:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Springfield	Acme Corp	123 Main St
2	2	Shelbyville	Tech Solutions	456 Elm St
3	3	Capital City	Global Traders	789 Oak St

Schemat bazy danych:



Podobnie jak w poprzednich zadaniach, schemat bazy danych został zoptymalizowany i nie widzimy tabeli Address.

- SQL logi

```
Hibernate:
    create sequence Categories_SEQ start with 1 increment by 50
Hibernate:
    create sequence Invoice_SEQ start with 1 increment by 50
Hibernate:
    create sequence Products_SEQ start with 1 increment by 50
Hibernate:
    create sequence Supplier_SEQ start with 1 increment by 50
Hibernate:
    create table Category (
        categoryID integer not null,
        name varchar(255),
        primary key (categoryID)
    )
Hibernate:
    create table Invoice (
        invoiceID integer not null,
        invoiceNumber integer not null,
        quantity integer not null,
        primary key (invoiceID)
    )
Hibernate:
    create table Invoice_Product (
        invoices_invoiceID integer not null,
        products_productID integer not null,
        primary key (invoices_invoiceID, products_productID)
    )
```

```
Hibernate:
    create table Product (
        category_categoryID integer,
        productID integer not null,
        supplier_supplierID integer,
        unitsInStock integer not null,
        productName varchar(255),
        primary key (productID)
    )
Hibernate:
    create table Supplier (
        supplierID integer not null,
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        primary key (supplierID)
    )
Hibernate:
    alter table Invoice_Product
       add constraint FK2mn08nt19nrqagr12grh5uho0
       foreign key (products_productID)
       references Product
Hibernate:
    alter table Invoice_Product
       add constraint FKthlx5t7fv55fgsfp7ivt4fj6u
       foreign key (invoices_invoiceID)
       references Invoice
Hibernate:
    alter table Product
       add constraint FK987q0koesbyk7oqky7lg431xr
       foreign key (category_categoryID)
       references Category
Hibernate:
    alter table Product
       add constraint FK6em1ebdcyqbgxmglei6wanchp
       foreign key (supplier_supplierID)
       references Supplier
--------------------
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
0
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Products_SEQ
```

```
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:
    insert
    into
        Supplier
        (city, street, companyName, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, street, companyName, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, street, companyName, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Product
```

```
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Product added successfully.
--------------------
```

c-d) mapowanie danych adresowych do osobnej tabeli

Zgodnie z poleceniem, zrezegnowaliśmy z klasy Address i umieściliśmy dane adresowe w klasie Supplier. Wskazaliśmy, że to będzie nowa osobna tabela.

- Supplier

```
package zad1;

import jakarta.persistence.*;

import java.util.ArrayList;
import java.util.List;

@Entity
@SecondaryTable(name = "Address")
@SequenceGenerator(name = "Supplier_SEQ")
class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Supplier_SEQ")
    private int supplierID;

    private String companyName;

    @Column(table = "Address")
    private String city;
    @Column(table = "Address")
    private String street;

    @OneToMany(mappedBy = "supplier")
    private List<Product> products = new ArrayList<>();

    public Supplier() {}

    public Supplier(String companyName, String city, String street) {
        this.companyName = companyName;
        this.city = city;
        this.street = street;
    }
    public String getCity() {
        return city;
    }
    public String getStreet() {
        return street;
    }
    int getSupplierID() {
        return supplierID;
    }
    String getCompanyName() {
        return companyName;
    }
    List<Product> getProducts() {
        return products;
    }
    void addProduct(Product product) {
        this.products.add(product);
    }
    @Override
    public String toString() { return companyName; }
}
```

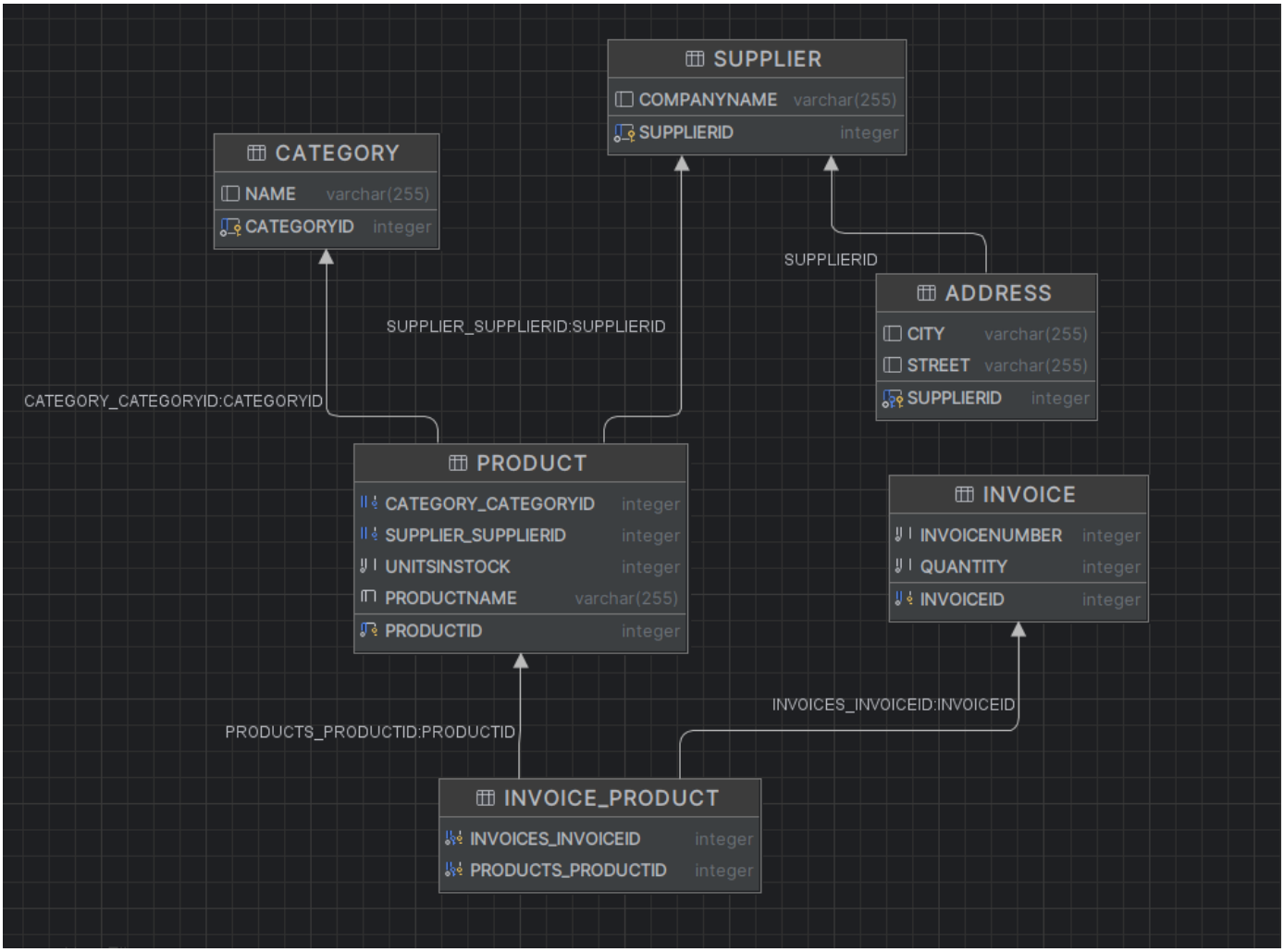
W wyniku takiej zmiany powstała nowa klasa-tabela Address:

	SUPPLIERID	CITY	STREET
1	1	123 Main St	Springfield
2	2	456 Elm St	Shelbyville
3	3	789 Oak St	Capital City

Suppliers mają teraz trochę prostszą strukturę:

	SUPPLIERID	COMPANYNAME
1	1	Acme Corp
2	2	Tech Solutions
3	3	Global Traders

Schemat bazy danych:



Pojawiła się tabela Address, ponieważ zmapowaliśmy do niej jawnie niektóre pola z Supplier.

- SQL logi

```
Hibernate:
    create sequence Categories_SEQ start with 1 increment by 50
Hibernate:
    create sequence Invoice_SEQ start with 1 increment by 50
Hibernate:
    create sequence Products_SEQ start with 1 increment by 50
Hibernate:
    create sequence Supplier_SEQ start with 1 increment by 50
Hibernate:
```

```
    create table Address (
        supplierID integer not null,
        city varchar(255),
        street varchar(255),
        primary key (supplierID)
    )
Hibernate:
    create table Category (
        categoryID integer not null,
        name varchar(255),
        primary key (categoryID)
    )
Hibernate:
    create table Invoice (
        invoiceID integer not null,
        invoiceNumber integer not null,
        quantity integer not null,
        primary key (invoiceID)
    )
Hibernate:
    create table Invoice_Product (
        invoices_invoiceID integer not null,
        products_productID integer not null,
        primary key (invoices_invoiceID, products_productID)
    )
Hibernate:
    create table Product (
        category_categoryID integer,
        productID integer not null,
        supplier_supplierID integer,
        unitsInStock integer not null,
        productName varchar(255),
        primary key (productID)
    )
Hibernate:
    create table Supplier (
        supplierID integer not null,
        companyName varchar(255),
        primary key (supplierID)
    )
Hibernate:
    alter table Address
       add constraint FKsg53al8nvbanq59s3pd6axyit
       foreign key (supplierID)
       references Supplier
Hibernate:
    alter table Invoice_Product
       add constraint FK2mn08nt19nrqagr12grh5uho0
       foreign key (products_productID)
       references Product
Hibernate:
    alter table Invoice_Product
       add constraint FKthlx5t7fv55fgsfp7ivt4fj6u
       foreign key (invoices_invoiceID)
       references Invoice
Hibernate:
    alter table Product
       add constraint FK987q0koesbyk7oqky7lg431xr
       foreign key (category_categoryID)
       references Category
Hibernate:
    alter table Product
       add constraint FK6em1ebdcyqbgxmglei6wanchp
       foreign key (supplier_supplierID)
       references Supplier
--------------------
What do you want to do?:
0. Add template
1. Add product
2. Add supplier
3. Show products
4. Show suppliers
```

```
5. Show category
6. Show products with Category
7. Show Sold products on InvoiceID:
8. Show Invoices that product had been sold:
9. Show Product Category
10. Exit
--------------------
0
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Categories_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:

values
    next value for Products_SEQ
Hibernate:

values
    next value for Invoice_SEQ
Hibernate:
    insert
    into
        Supplier
        (companyName, supplierID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Address
        (city, street, supplierID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (companyName, supplierID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Address
        (city, street, supplierID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Supplier
```

```
        (companyName, supplierID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Address
        (city, street, supplierID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Category
        (name, categoryID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice
        (invoiceNumber, quantity, invoiceID)
    values
```

```
        (?, ?, ?)
Hibernate:
    insert
    into
        Product
        (category_categoryID, productName, supplier_supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Invoice_Product
        (invoices_invoiceID, products_productID)
    values
        (?, ?)
Product added successfully.
--------------------
```

Zadanie 9 - dziedziczenie

W ramach tego zadania rozważyliśmy 3 podejścia dziedziczenia w Hibernate/JPA.

a) Type Per Class

Zaimplementowane klasy:

- Company

```
package zad9;

import jakarta.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Company_GEN")
    @SequenceGenerator(name = "Company_GEN", sequenceName = "Company_SEQ")
    private int companyID;
    private String companyName;
    private String street;
    private String city;
    private String zipCode;
    public Company() {}
    public Company(String companyName, String street, String city, String zipCode)
    {
        this.companyName = companyName;
        this.zipCode = zipCode;
        this.street = street;
        this.city = city;
    }
    @Override
    public String toString() {
        return companyName;
    }
    public String getCompanyName() {
        return companyName;
    }
    public String getStreet() {
        return street;
    }
    public String getCity() {
        return city;
    }
    public String getZipCode() {
        return zipCode;
    }
}
```

- Supplier

```
package zad9;
import jakarta.persistence.*;

@Entity
public class Supplier extends Company {
    private String bankAccountNumber;
    public Supplier() {}
    public Supplier(String companyName, String street, String city, String
            zipCode, String bankAccountNumber) {
        super(companyName, street, city, zipCode);
        this.bankAccountNumber = bankAccountNumber;
    }
```

```
}
```

- Customer

```
package zad9;

import jakarta.persistence.*;

@Entity
public class Customer extends Company {
    private double discount; // %
    public Customer() {}
    public Customer(String companyName, String street, String city, String
            zipCode, double discount) {
        super(companyName, street, city, zipCode);
        this.discount = discount;
    }
}
```

- Main

```
package zad9;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Persistence;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import java.util.List;
import java.util.Scanner;
import java.util.Set;


class Main {
    private static final EntityManagerFactory emf;

    static {
        try {
            emf = Persistence.createEntityManagerFactory("derby");
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public static void main(String[] args) {
        EntityManager em = getEntityManager();
        EntityTransaction etx = em.getTransaction();

        etx.begin();
        Customer customer1 = new Customer("Client #1", "Short street", "Zlin",
                "27-001", 0.0);
        Customer customer2 = new Customer("Client #2", "29 listopada", "Gdynia", "35-693",
                2.0);
        Supplier supplier1 = new Supplier("Supplier #1", "Rondo Kosmiczne",
                "Warsaw", "40-367", "687453214569874536541236");
        Supplier supplier2 = new Supplier("Supplier #2", "Oil street", "Kyiv", "10-234",
                "9874563256463215896345 6");
        em.persist(customer1);
        em.persist(customer2);
        em.persist(supplier1);
        em.persist(supplier2);
```

```
        etx.commit();
        em.close();
    }
}
```

- SQL logi

```
Hibernate:
    create sequence Company_SEQ start with 1 increment by 50
Hibernate:
    create table Customer (
        companyID integer not null,
        discount float(52) not null,
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        zipCode varchar(255),
        primary key (companyID)
    )
Hibernate:
    create table Supplier (
        companyID integer not null,
        bankAccountNumber varchar(255),
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        zipCode varchar(255),
        primary key (companyID)
    )
Hibernate:

values
    next value for Company_SEQ
Hibernate:

values
    next value for Company_SEQ
Hibernate:
    insert
    into
        Customer
        (city, companyName, street, zipCode, discount, companyID)
    values
        (?, ?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Customer
        (city, companyName, street, zipCode, discount, companyID)
    values
        (?, ?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, zipCode, bankAccountNumber, companyID)
    values
        (?, ?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (city, companyName, street, zipCode, bankAccountNumber, companyID)
    values
        (?, ?, ?, ?, ?, ?)
```

W bazie danych zostały utworzone dwie tabeli - Customer i Supplier.

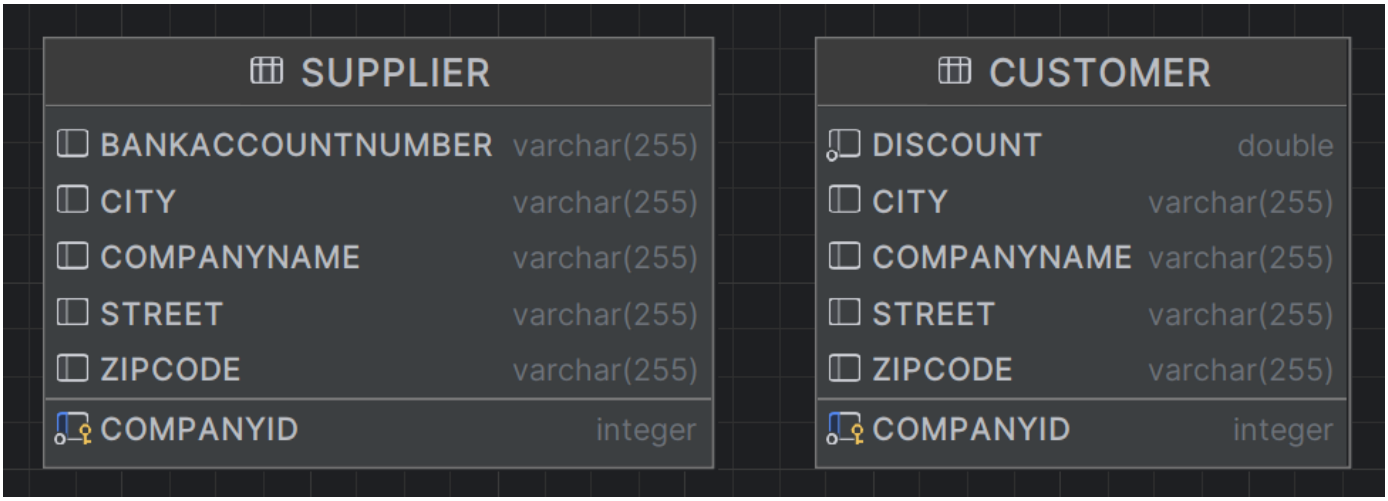
Supplier:

	COMPANYID	BANKACCOUNTNUMBER	CITY	COMPANYNAME	STREET	ZIPCODE
1	3	68745321456987453654123 6	Warsaw	Supplier #1	Rondo Kosmiczne	40-367
2	4	98745632564632158963456	Kyiv	Supplier #2	Oil street	10-234

Customer:

	COMPANYID	DISCOUNT	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	0	Zlin	Client #1	Short street	27-001
2	2	2	Gdynia	Client #2	29 listopada	35-693

Schemat bazy danych:



b) Joined

W tym podejściu zmieniliśmy jedynie dekorator klasy Company na odpowiedni danej strategii.

- Company (fragment starego kodu)

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Company {
...
}
```

- Company (fragment nowego kodu)

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Company {
...
}
```

- SQL logi

```
Hibernate:
    create sequence Company_SEQ start with 1 increment by 50
Hibernate:
    create table Company (
        companyID integer not null,
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
```

```
        zipCode varchar(255),
        primary key (companyID)
    )
Hibernate:
    create table Customer (
        companyID integer not null,
        discount float(52) not null,
        primary key (companyID)
    )
Hibernate:
    create table Supplier (
        companyID integer not null,
        bankAccountNumber varchar(255),
        primary key (companyID)
    )
Hibernate:
    alter table Customer
       add constraint FKn7fvr687iixps0s6i5casr6f3
       foreign key (companyID)
       references Company
Hibernate:
    alter table Supplier
       add constraint FKpinunrb4v5p4aemt2k4fnkjp8
       foreign key (companyID)
       references Company
Hibernate:

values
    next value for Company_SEQ
Hibernate:

values
    next value for Company_SEQ
Hibernate:
    insert
    into
        Company
        (city, companyName, street, zipCode, companyID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Customer
        (discount, companyID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Company
        (city, companyName, street, zipCode, companyID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Customer
        (discount, companyID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Company
        (city, companyName, street, zipCode, companyID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
```

```
        (bankAccountNumber, companyID)
    values
        (?, ?)
Hibernate:
    insert
    into
        Company
        (city, companyName, street, zipCode, companyID)
    values
        (?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        Supplier
        (bankAccountNumber, companyID)
    values
        (?, ?)
```

W bazie danych zostałe utworzone trzy tabele - Company, Customer oraz Supplier, czyli każda tabela przedstawia osobną klasę.

Company:

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	Zlin	Client #1	Short street	27-001
2	2	Gdynia	Client #2	29 listopada	35-693
3	3	Warsaw	Supplier #1	Rondo Kosmiczne	40-367
4	4	Kyiv	Supplier #2	Oil street	10-234

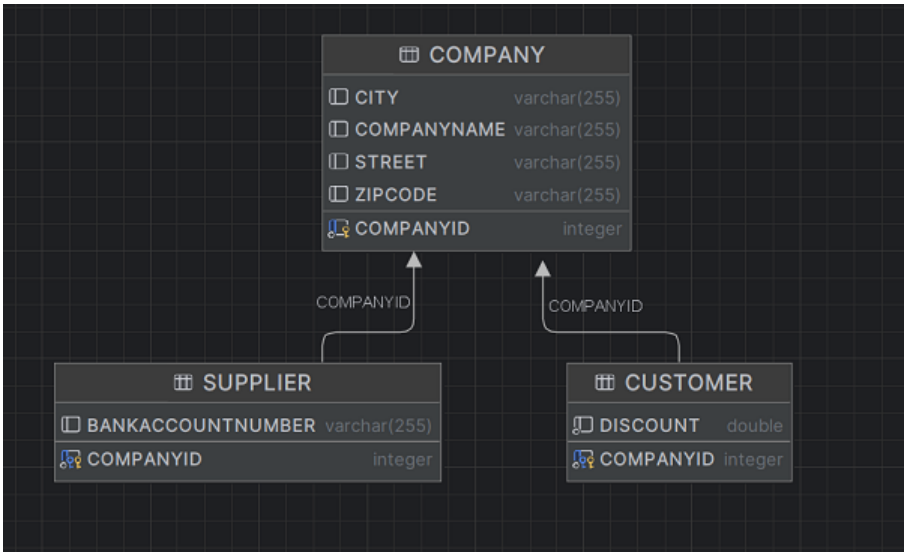
Supplier:

	COMPANYID	BANKACCOUNTNUMBER
1	3	687453214569874536541236
2	4	987456325646321589634 56

Customer:

	COMPANYID	DISCOUNT
1	1	0
2	2	2

Schemat bazy danych:



c) Single Table

W tym podejściu zmieniliśmy jedynie dekorator klasy Company na odpowiedni danej strategii.

- Company (fragment starego kodu)

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Company {
...
}
```

- Company (fragment nowego kodu)

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class Company {
...
}
```

- SQL logi

```
Hibernate:
    create table Company (
        companyID integer not null,
        discount float(52),
        DTYPE varchar(31) not null,
        bankAccountNumber varchar(255),
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        zipCode varchar(255),
        primary key (companyID)
    )
Hibernate:

values
    next value for Company_SEQ
Hibernate:

values
    next value for Company_SEQ
Hibernate:
    insert
    into
        Company
        (city, companyName, street, zipCode, discount, DTYPE, companyID)
    values
        (?, ?, ?, ?, ?, 'Customer', ?)
Hibernate:
    insert
    into
        Company
        (city, companyName, street, zipCode, discount, DTYPE, companyID)
    values
        (?, ?, ?, ?, ?, 'Customer', ?)
Hibernate:
    insert
    into
        Company
        (city, companyName, street, zipCode, bankAccountNumber, DTYPE, companyID)
    values
        (?, ?, ?, ?, ?, 'Supplier', ?)
Hibernate:
    insert
```

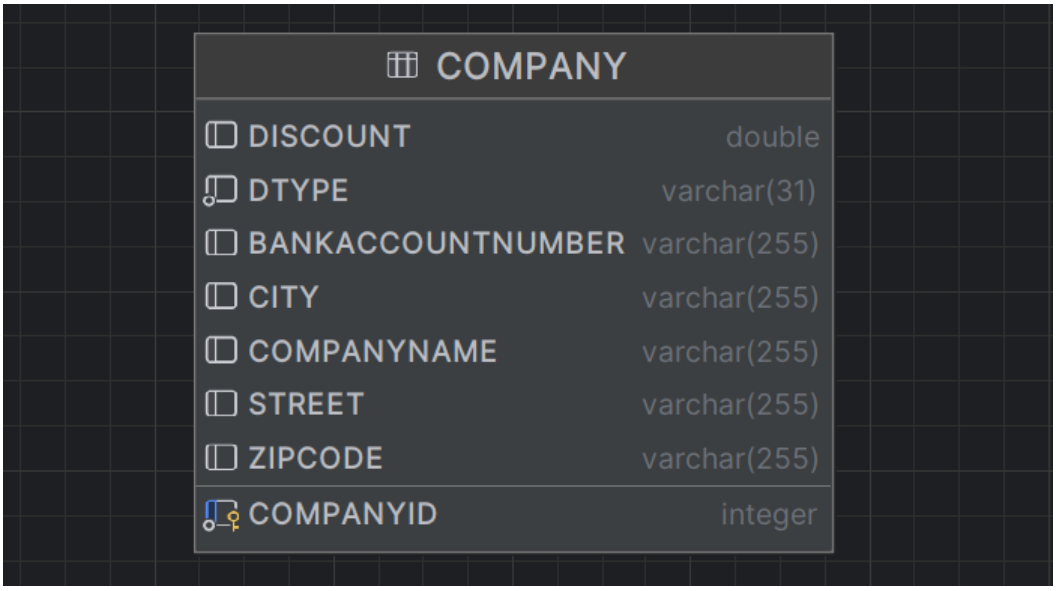
```
    into
        Company
        (city, companyName, street, zipCode, bankAccountNumber, DTYPE, companyID)
    values
        (?, ?, ?, ?, ?, 'Supplier', ?)
```

W bazie danych została utworzona jedna tabela - Company, czyli ma ona nulle na polach Supplier`a, jeśli dany object jest Customer'em i tak samo w drugą stronę.

Company:

	COMPANYID	DISC…	DTYPE	BANKACCOUN…	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	0	Customer	<null>	Zlin	Client #1	Short street	27-001
2	2	2	Customer	<null>	Gdynia	Client #2	29 listopada	35-693
3	3	<null>	Supplier	687453214569874536…	Warsaw	Supplier #1	Rondo Kosmiczne	40-367
4	4	<null>	Supplier	987456325646321589…	Kyiv	Supplier #2	Oil street	10-234

Schemat bazy danych:



Każde z pokazanych podejść dziedziczenia ma swoje wade i zalety. Type per class - to sposób, gdy nie jest tworzona żadna macierzysta struktura, a więc może się okazać że baza będzie miała bardzo dużo powielanych danych. Jest to wygodna struktura dostępu do konkretnych danych, ale słabo nadaje się do polimorfizmu. Natomiast, Joined Table ma prejżystą stukturę tabel, ponieważ w tym podejściu są tworzone tabeli dla każdej klasy hierarchii, włącznie z abstrakcyjnymi. Nie jest to najlepszy wybór do polimorfizmu, utrudnia się również dostęp do danych poszczególnych klas. Z kolei Single Table jest lepszym wyborem dla dostępu do klas dziedziczących i modelowania polimorfizmu, natomiast schemat takiej bazy w przypadku rozszerzenia może zawierać ogromną ilość nulli w specyficznych dla danego typu dziedziczącego.