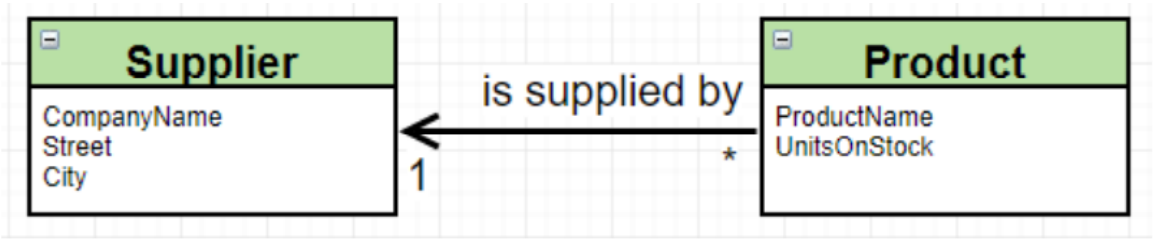
Bazy Danych
Hibernate / JPA

Barbara Wojtarowicz
grupa czw. 14:40 B

Raport z wykonania ćwiczenia

1. Zadanie II

Zmodyfikowano model, wprowadzając pojęcie Dostawcy zgodnie z poniższym schematem:



1. Dodałam klasę Supplier.

```
package com.company;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Supplier {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   public int SupplierID;
   public String CompanyName;
   public String Street;
   public String City;

   public Supplier(){

   }

   public Supplier(String companyName, String street, String city){
       CompanyName = companyName;
       Street = street;
       City = city;
   }
}
```

2. Dodałam pole Supplier w klasie Product.

```
package com.company;

import javax.persistence.*;
```

```
@Entity
public class Product {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   private int productID;

   private String ProductName;
   private Integer UnitsOnStock;

   @ManyToOne
   public Supplier Supplier;

   public Product(){

   }

   public Product(String productName, Integer unitsOnStock){
       this.ProductName = productName;
       this.UnitsOnStock = unitsOnStock;
   }

   public Product(String productName, Integer unitsOnStock, Supplier
supplier){
       this.ProductName = productName;
       this.UnitsOnStock = unitsOnStock;
       this.Supplier = supplier;
   }

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

}
```

3. Zmodyfikowałam hibernate.cfg.xml, dodając mapping klasy Supplier.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD//EN"
   "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
   <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</prope
```

```
rty>
   <property
name="connection.url">jdbc:derby://127.0.0.1/BarbaraWojtarowiczJPA;creat
e=true</property>
   <property
name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
   <property name="show_sql">true</property>
   <property name="format_sql">true</property>
   <property name="use_sql_comments">true</property>
   <property name="hbm2ddl.auto">update</property>
   <mapping class="com.company.Product"></mapping>
   <mapping class="com.company.Supplier"></mapping>
 </session-factory>
</hibernate-configuration>
```

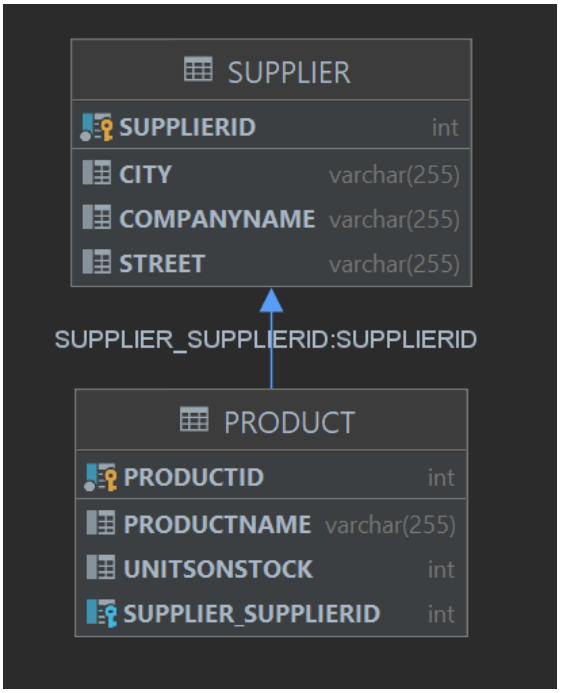
4. Zmodyfikowałam klasę Main tak, by dodała nowego dostawcę oraz w ostatnio dodanym produkcie ustawiła pole producenta na tego nowo dodanego.

```
package com.company;

import org.hibernate.*;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.NativeQuery;

import javax.persistence.metamodel.EntityType;

public class Main {
   private static final SessionFactory ourSessionFactory;

   static {
       try {
           Configuration configuration = new Configuration();
           configuration.configure();

           ourSessionFactory = configuration.buildSessionFactory();
       } catch (Throwable ex) {
           throw new ExceptionInInitializerError(ex);
       }
   }

   public static Session getSession() throws HibernateException {
       return ourSessionFactory.openSession();
   }

   public static void main(final String[] args) throws Exception {
       final Session session = getSession();
```

```
        Supplier supplier = new Supplier("Dostawca1", "Wrocławska 10",
"Kraków");
        Product foundProduct = session.get(Product.class, 1);
        foundProduct.setSupplier(supplier);

        try {
            Transaction tx = session.beginTransaction();
            session.save(supplier);
            session.save(foundProduct);
            tx.commit();

            System.out.println("Querying all the entities being
managed:");
            final Metamodel metamodel =
session.getSessionFactory().getMetamodel();
            for (EntityType<?> entityType : metamodel.getEntities()){
                final String entityName = entityType.getName();
                //final Query query = session.createQuery("from" +
entityName);
                final NativeQuery query =
session.createNativeQuery("SELECT * FROM " + entityName);
                System.out.println("...executing: " +
query.getQueryString() + "...");
                for (Object o : query.list()){
                    System.out.println(" " + o);
                }
            }

        } finally {
            session.close();
        }
    }
}
```

W wyniku ww. działań zaszły następujące zmiany:

5. Schemat bazy przyjął postać:



6. Pojawili się nowi dostawcy (dwa razy uruchomiłam klasę Main :)):

```
select * from Supplier
```

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	3	Kraków	Dostawca1	Wrocławska 10
2	4	Kraków	Dostawca1	Wrocławska 10

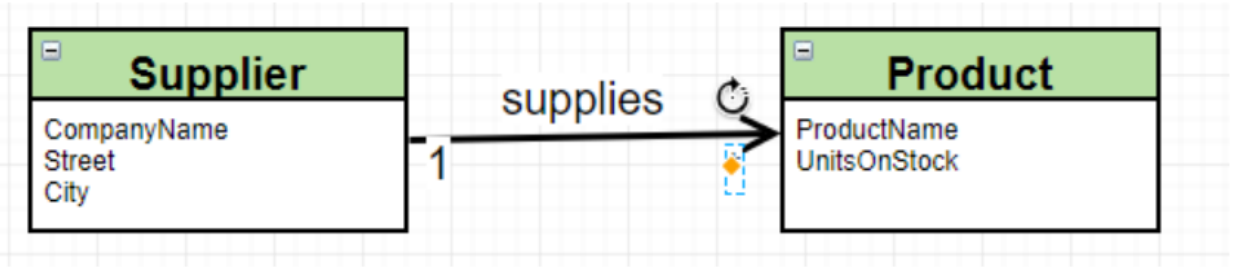
7. Ostatnio dodany produkt (produktów w sumie było dwa) przyjął pole dostawcy o SupplierID równym 4:

```
select * from Product
```

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	1	Krzeslo	111	4
2	2	Krzeslo	111	<null>

2. Zadanie III

Zmodyfikowano model zgodnie z poniższym schematem:



1. Wariant z tabelą łącznikową
   1.1. Z klasy Product usunęłam pole Supplier i związane z nim metody.

```
package com.company;

import javax.persistence.*;

@Entity
public class Product {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   private int productID;

   private String ProductName;
   private Integer UnitsOnStock;

   public Product(){

   }

   public Product(String productName, Integer unitsOnStock){
       this.ProductName = productName;
       this.UnitsOnStock = unitsOnStock;
   }
}
```

   1.2. Do klasy Supplier dodałam HashSet produktów, reprezentujący relację @OneToMany oraz metodę dodającą produkt do tego zbioru.

```
package com.company;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Supplier {
```

```
    @Id
    @GeneratedValue(
            strategy = GenerationType.AUTO
    )
    public int SupplierID;
    public String CompanyName;
    public String Street;
    public String City;

    @OneToMany
    public Set<Product> suppliedProducts = new HashSet<>();

    public Supplier(){

    }

    public Supplier(String companyName, String street, String city){
        CompanyName = companyName;
        Street = street;
        City = city;
    }

    public Supplier(String companyName, String street, String city,
Product product){
        CompanyName = companyName;
        Street = street;
        City = city;
        this.suppliedProducts.add(product);
    }

    public void addProduct(Product product){
        this.suppliedProducts.add(product);
    }
}
```

1.3. W klasie Main dodałam dostawcę oraz kilka produktów, które następnie dodałam do zbioru produktów przez niego dostarczanych.

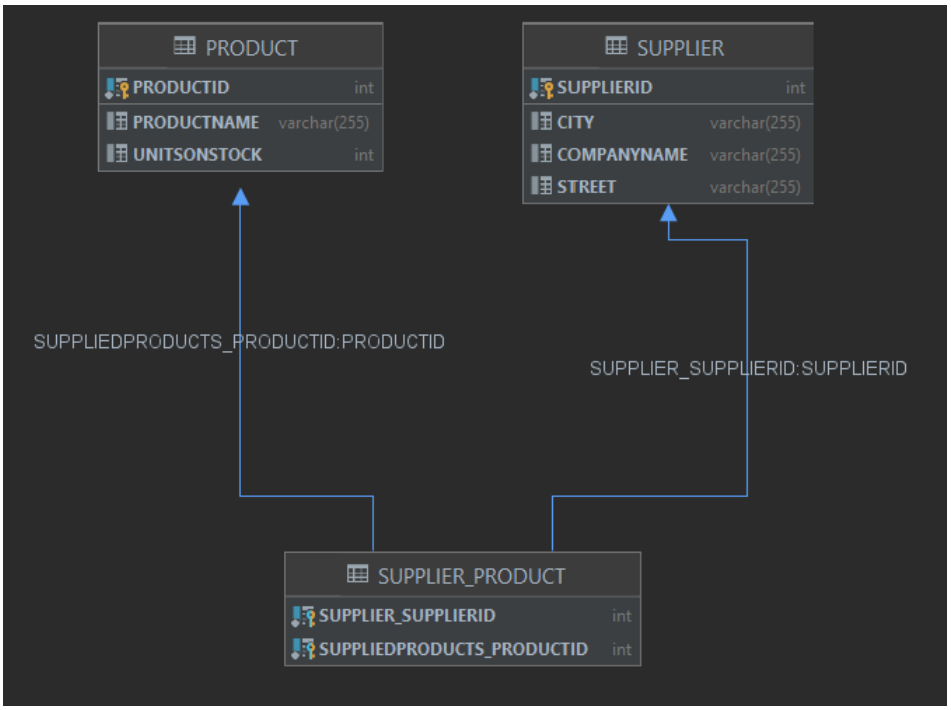
```
package com.company;

import org.hibernate.*;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.NativeQuery;

import javax.persistence.metamodel.EntityType;
```

```
public class Main {
   private static final SessionFactory ourSessionFactory;

   static {
       try {
           Configuration configuration = new Configuration();
           configuration.configure();

           ourSessionFactory = configuration.buildSessionFactory();
       } catch (Throwable ex) {
           throw new ExceptionInInitializerError(ex);
       }
   }

   public static Session getSession() throws HibernateException {
       return ourSessionFactory.openSession();
   }

   public static void main(final String[] args) throws Exception {
       final Session session = getSession();

       Supplier supplier2 = new Supplier("Dostawca2", "Pomorksa 12",
"Gdańsk");
       Product product1 = new Product("Milk", 10);
       Product product2 = new Product("Banana", 200);
       Product product3 = new Product("Book", 2);
       supplier2.addProduct(product1);
       supplier2.addProduct(product2);
       supplier2.addProduct(product3);

       try {
           Transaction tx = session.beginTransaction();
           session.save(product1);
           session.save(product2);
           session.save(product3);
           session.save(supplier2);
           tx.commit();

           System.out.println("Querying all the entities being
managed:");
           final Metamodel metamodel =
session.getSessionFactory().getMetamodel();
           for (EntityType<?> entityType : metamodel.getEntities()){
               final String entityName = entityType.getName();
               //final Query query = session.createQuery("from" +
entityName);
```

```
                final NativeQuery query =
session.createNativeQuery("SELECT * FROM " + entityName);
                System.out.println("...executing: " +
query.getQueryString() + "...");
                for (Object o : query.list()){
                    System.out.println(" " + o);
                }
            }

        } finally {
            session.close();
        }
    }
}
```

W wyniku ww. działań zaszły następujące zmiany:

1.4. W bazie pojawiła się tabela łącznikowa SUPPLIER_PRODUCT:



1.5. W tabeli Supplier pojawił się nowy dostawca:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Gdańsk	Dostawca2	Pomorksa 12

1.6. W tabeli Products pojawiły się produkty:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Milk	10
2	2	Banana	200
3	3	Book	2

1.7. W tabeli SUPPLIER_PRODUCT pojawiły się relacje zachodzące między dostawcą a produktami:

	SUPPLIER_SUPPLIERID	SUPPLIEDPRODUCTS_PRODUCTID
1	4	1
2	4	2
3	4	3

2. Wariant bez tabeli łącznikowej

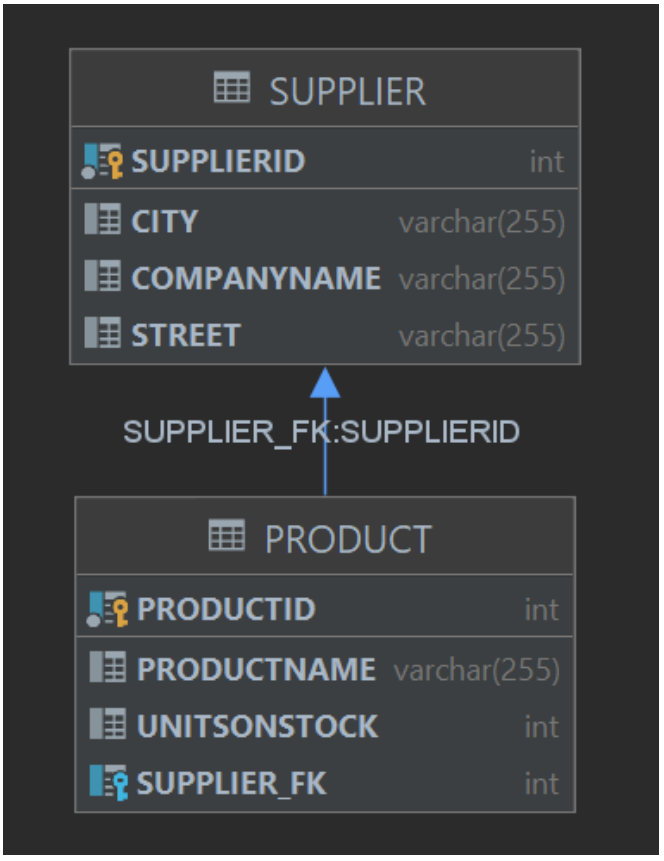
2.1. Zmodyfikowałam klasę Supplier, dodając adnotację @JoinColumn przy adnotowaniu relacji @OneToMany:

```
package com.company;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Supplier {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   public int SupplierID;
   public String CompanyName;
   public String Street;
   public String City;

   @OneToMany
   @JoinColumn(name="Supplier_FK")
   public Set<Product> suppliedProducts = new HashSet<>();

   public Supplier(){

   }

   public Supplier(String companyName, String street, String city){
```

```
        CompanyName = companyName;
        Street = street;
        City = city;
    }

    public Supplier(String companyName, String street, String city,
Product product){
        CompanyName = companyName;
        Street = street;
        City = city;
        this.suppliedProducts.add(product);
    }

    public void addProduct(Product product){
        this.suppliedProducts.add(product);
    }
}
```

Reszta kodu pozostała bez zmian. Uzyskano następujące efekty:

2.2. Do bazy nie została dodana tabela łącznikowa. Zamiast tego, w tabeli Product pojawiła się kolumna Supplier_FK (taką nazwę nadałam jej w adnotacji @JoinColumn(name="Supplier_FK"), która zawiera klucz obcy do tabeli Supplier:



2.3. Tabela Supplier wygląda tak, jak wcześniej:

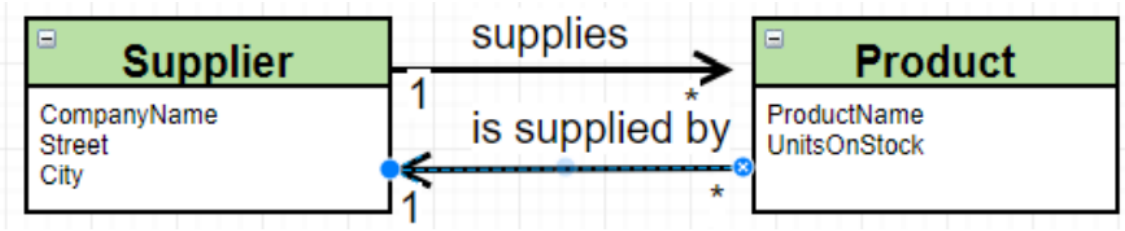
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Gdańsk	Dostawca2	Pomorksa 12

2.4. Tabela Products zawiera teraz kolumnę z kluczem obcym:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK
1	1	Milk	10	4
2	2	Banana	200	4
3	3	Book	2	4

3. Zadanie IV

Zamodelowano relację dwustronną zgodnie ze schematem:



1. W klasie Supplier metoda addProduct dodaje produkt do zbioru SuppliedProducts, a także modyfikuje pole Supplier tego produktu. Metoda suppliesProduct zwraca info., czy dany produkt figuruje w zbiorze SuppliedProducts danej instancji klasy Supplier. Relacja @OneToMany pozostaje bez zmian.

```
package com.company;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Supplier {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   public int SupplierID;
   public String CompanyName;
   public String Street;
   public String City;

   @OneToMany
```

```
    @JoinColumn(name="SUPPLIED_BY")
    public Set<Product> suppliedProducts = new HashSet<>();

    public Supplier(){

    }

    public Supplier(String companyName, String street, String city){
        CompanyName = companyName;
        Street = street;
        City = city;
    }

    public Supplier(String companyName, String street, String city,
Product product){
        CompanyName = companyName;
        Street = street;
        City = city;
        this.suppliedProducts.add(product);
    }

    public void addProduct(Product product){
        this.suppliedProducts.add(product);
        product.setSupplier(this);
    }

    public boolean suppliesProduct(Product product){
        return suppliedProducts.contains(product);
    }
}
```

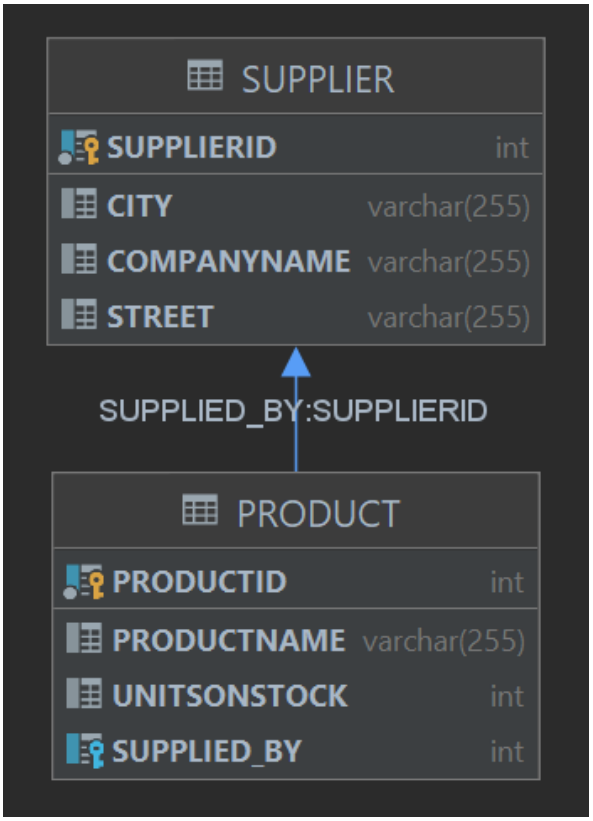
2. Do klasy Product wróciło pole Supplier oraz związana z nim metoda setSupplier. Pole występuje z adnotacją relacji @ManyToOne poprzez kolumnę “SUPPLIER_BY”.

```
package com.company;

import javax.persistence.*;

@Entity
public class Product {
    @Id
    @GeneratedValue(
            strategy = GenerationType.AUTO
    )
    private int productID;
```

```
    private String ProductName;
    private Integer UnitsOnStock;

    @ManyToOne
    @JoinColumn(name="SUPPLIED_BY")
    public Supplier Supplier;

    public Product(){

    }

    public Product(String productName, Integer unitsOnStock){
        this.ProductName = productName;
        this.UnitsOnStock = unitsOnStock;
    }

    public Product(String productName, Integer unitsOnStock, Supplier
supplier){
        this.ProductName = productName;
        this.UnitsOnStock = unitsOnStock;
        this.Supplier = supplier;
    }

    public void setSupplier(Supplier supplier){
        this.Supplier = supplier;
        if (!Supplier.suppliesProduct(this))
            Supplier.addProduct(this);
    }
}
```

W wyniku ww. działań zaszły następujące zmiany:

3. Schemat bazy przyjął poniższą postać:



4. Dodany został dostawca:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Gdańsk	Dostawca2	Pomorksa 12

5. W tabeli Product widać pole SUPPLIED_BY z kluczem obcym dostawcy:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIED_BY
1	1	Milk	10	4
2	2	Banana	200	4
3	3	Book	2	4

4. Zadanie V

1. Dodałam klasę Category. Występuje w niej relacja @OneToMany w postaci listy produktów tej kategorii. Metoda addProduct dodaje produkt do listy produktów oraz wywołuje na tym produkcie metodę setCategory, która modyfikuje pole kategorii w tym produkcie.

```
package com.company;

import javax.persistence.*;
import java.util.ArrayList;
```

```java
import java.util.List;

@Entity
public class Category {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   private int CategoryID;

   private String Name;

   @OneToMany
   @JoinColumn(name="IN_CATEGORY")
   public List<Product> Products = new ArrayList<>();

   public Category(){

   }

   public Category(String name){
       Name = name;
   }

   public Category(String name, Product product){
       Name = name;
       addProduct(product);
   }

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

   public void addProduct(Product product){
       this.Products.add(product);
       product.setCategory(this);
   }
}
```

2. Zmodyfikowałam klasę Product, dodając pole Category w postaci relacji @ManyToOne, a także związaną z nią metodę setCategory, która ustawia kategorię tego produktu, a dodatkowo, jeśli produkt ten jeszcze nie jest zawarty w liście produktów danej kategorii, dodaje go tam.

```java
package com.company;
```

```
import javax.persistence.*;

@Entity
public class Product {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   private int productID;

   private String ProductName;
   private Integer UnitsOnStock;

   @ManyToOne
   @JoinColumn(name="SUPPLIED_BY")
   public Supplier Supplier;

   @ManyToOne
   @JoinColumn(name="IN_CATEGORY")
   public Category Category;

   public Product(){

   }

   public Product(String productName, Integer unitsOnStock){
       this.ProductName = productName;
       this.UnitsOnStock = unitsOnStock;
   }

   public Product(String productName, Integer unitsOnStock, Category
category){
       this(productName, unitsOnStock);
       this.Category = category;
   }

   public Product(String productName, Integer unitsOnStock, Supplier
supplier){
       this(productName, unitsOnStock);
       this.Supplier = supplier;
   }

   public Product(String productName, Integer unitsOnStock, Supplier
supplier, Category category){
       this(productName, unitsOnStock, supplier);
```

```
        this.Category = category;
    }

    public void setSupplier(Supplier supplier){
        this.Supplier = supplier;
        if (!Supplier.suppliesProduct(this))
            Supplier.addProduct(this);
    }

    public void setCategory(Category category) {
        this.Category = category;
        if (!category.getProducts().contains(this))
            category.addProduct(this);
    }
}
```

3. W klasie Main dodałam 2 dostawców, kilka produktów oraz 2 kategorie.

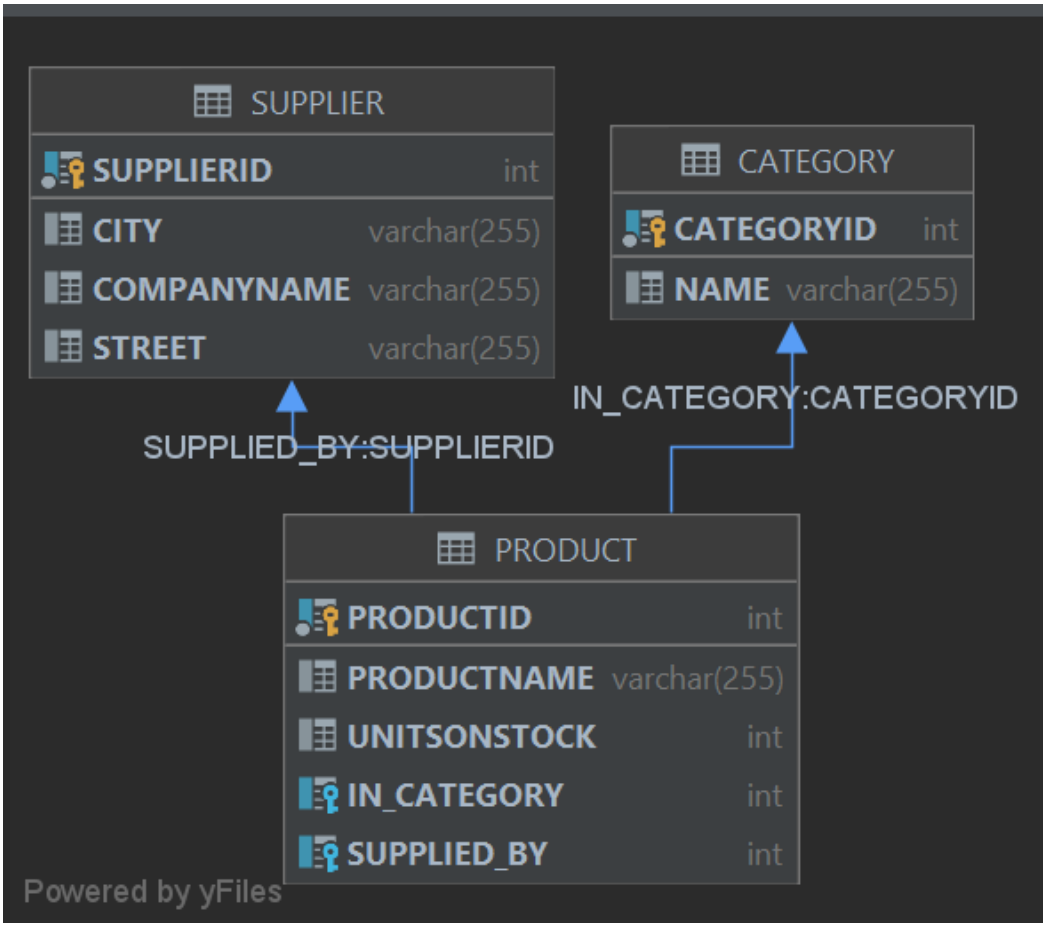
```
package com.company;

import org.hibernate.*;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.NativeQuery;

import javax.persistence.metamodel.EntityType;

public class Main {
    private static final SessionFactory ourSessionFactory;

    static {
        try {
            Configuration configuration = new Configuration();
            configuration.configure();

            ourSessionFactory = configuration.buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static Session getSession() throws HibernateException {
        return ourSessionFactory.openSession();
    }

    public static void main(final String[] args) throws Exception {
```

```
        final Session session = getSession();

        Supplier supplier1 = new Supplier("Supplier1", "Main Street 5",
"New York");
        Supplier supplier2 = new Supplier("Supplier2", "Main Road 10",
"London");

        Product product1 = new Product("Milk", 10);
        Product product2 = new Product("Banana", 200);
        Product product3 = new Product("Book", 2);

        Category food = new Category("Food");
        Category literature = new Category("Literature");


        supplier1.addProduct(product1);
        supplier2.addProduct(product2);
        supplier2.addProduct(product3);

        food.addProduct(product1);
        food.addProduct(product2);
        literature.addProduct(product3);

        try {
            Transaction tx = session.beginTransaction();
            session.save(supplier1);
            session.save(supplier2);
            session.save(product1);
            session.save(product2);
            session.save(product3);
            session.save(food);
            session.save(literature);
            tx.commit();

            System.out.println("Querying all the entities being
managed:");
            final Metamodel metamodel =
session.getSessionFactory().getMetamodel();
            for (EntityType<?> entityType : metamodel.getEntities()){
                final String entityName = entityType.getName();
                //final Query query = session.createQuery("from" +
entityName);
                final NativeQuery query =
session.createNativeQuery("SELECT * FROM " + entityName);
                System.out.println("...executing: " +
query.getQueryString() + "...");
```

```
                for (Object o : query.list()){
                    System.out.println(" " + o);
                }
            }

        } finally {
            session.close();
        }
    }
}
```

W wyniku ww. działań zaszły następujące zmiany:

4. Schemat bazy przyjął postać:



5. Dodano dostawców:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	New York	Supplier1	Main Street 5
2	2	London	Supplier2	Main Road 10

6. Dodano kategorie:

	CATEGORYID	NAME
1	6	Food
2	7	Literature

7. Dodano produkty z uwzględnieniem kategorii, do których należą:

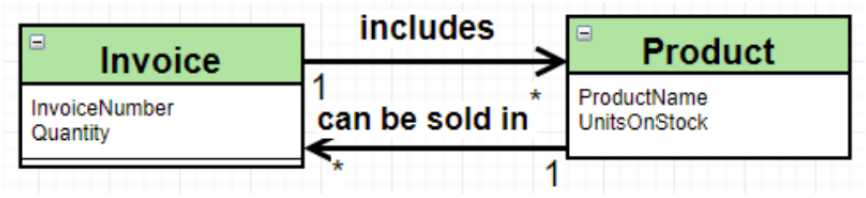
	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	IN_CATEGORY	SUPPLIED_BY
1	3	Milk	10	6	1
2	4	Banana	200	6	2
3	5	Book	2	7	2

8. Skrypt generujący tabele wygląda następująco:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table SUPPLIER
(
   SUPPLIERID  INTEGER not null
       primary key,
   CITY        VARCHAR(255),
   COMPANYNAME VARCHAR(255),
   STREET      VARCHAR(255)
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY,
   SUPPLIED_BY  INTEGER
       constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
           references SUPPLIER
);
```

5. Zadanie VI

Zamodelowano poniższą relację:



1. Stworzyłam klasę Invoice, zawierającą zbiór produktów zawartych na fakturze includedProducts, co reprezentuje relację typu @ManyToMany.

```
package com.company;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Invoice {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   private int InvoiceID;

   private String InvoiceNumber;
   private Integer Quantity;

   @ManyToMany
   private Set<Product> includedProducts = new HashSet<>();

   public Invoice() {}

   public Invoice(String invoiceNumber, Integer quantity){
       InvoiceNumber = invoiceNumber;
       Quantity = quantity;
   }

   public Invoice(String invoiceNumber, Integer quantity, Product
product){
       this(invoiceNumber, quantity);
       addProduct(product);
   }

   public void addProduct(Product product){
```

```
        includedProducts.add(product);
        product.getCanBeSoldIn().add(this);
        this.Quantity += 1;
    }
}
```

2. Do klasy Product dodałam zbiór faktur, do których należy dany produkt canBeSoldIn. Reprezentuje on relację @ManyToMany, w której canBeSoldIn mapowany jest przez includedProducts z klasy Invoice, w wyniku czego powstaje tabela INVOICE_PRODUCT opisująca tę relację.

Dla zachowania czytelności kodu poniżej zamieszczam tylko nowy fragment klasy Product:

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

dodano również getter dla tego Setu.

3. W klasie Main dodałam kilka produktów, 2 dostawców i 3 produkty.

Analogicznie, dla zachowania czytelności zamieszczam jedynie nowe fragmenty kodu:

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();

    Supplier supplier1 = new Supplier("Supplier1", "Main Street 5", "New
York");
    Supplier supplier2 = new Supplier("Supplier2", "Main Road 10",
"London");

    Product product1 = new Product("Milk", 10);
    Product product2 = new Product("Banana", 200);
    Product product3 = new Product("Book", 2);

    Category food = new Category("Food");
    Category literature = new Category("Literature");

    Invoice invoice1 = new Invoice("00001", 5);
    Invoice invoice2 = new Invoice("00002", 12);
    Invoice invoice3 = new Invoice("00099", 100);

    supplier1.addProduct(product1);
    supplier2.addProduct(product2);
    supplier2.addProduct(product3);
```

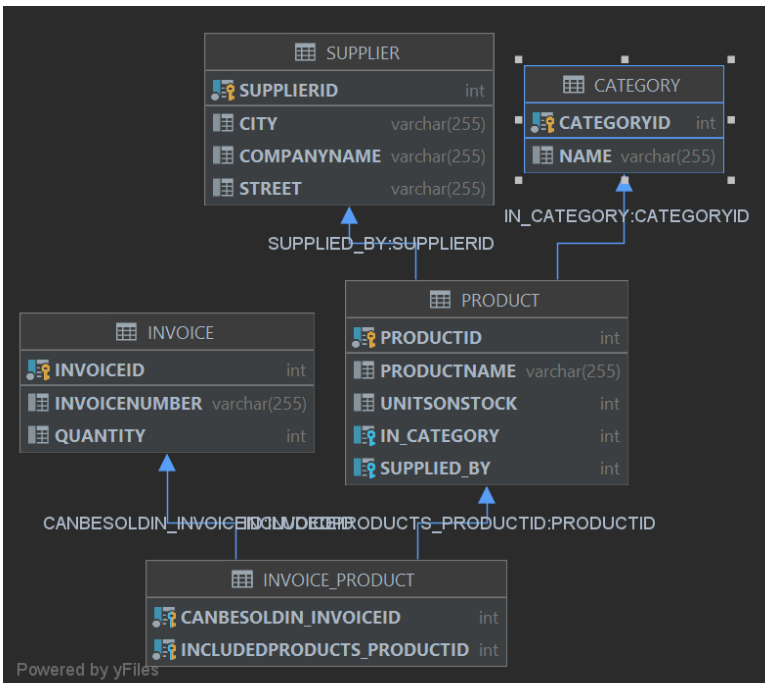
```
    food.addProduct(product1);
    food.addProduct(product2);
    literature.addProduct(product3);

    invoice1.addProduct(product1);
    invoice1.addProduct(product1);
    invoice1.addProduct(product1);
    invoice1.addProduct(product2);
    invoice2.addProduct(product2);
    invoice3.addProduct(product3);

    try {
        Transaction tx = session.beginTransaction();
        session.save(supplier1);
        session.save(supplier2);
        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(food);
        session.save(literature);
        session.save(invoice1);
        session.save(invoice2);
        session.save(invoice3);
        tx.commit();
```

4. Konieczne również było dodanie odpowiedniego mapowania w hibernate.cfg.xml:

```
<mapping class="com.company.Invoice"></mapping>
```

W wyniku ww. działań nastąpiły niżej opisane zmiany:

5. Schemat bazy przyjął postać:



6. Dodano kategorie:

	CATEGORYID	NAME
1	6	Food
2	7	Literature

7. Dodano dostawców:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	New York	Supplier1	Main Street 5
2	2	London	Supplier2	Main Road 10

8. Dodano produkty:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	IN_CATEGORY	SUPPLIED_BY
1	3	Milk	10	6	1
2	4	Banana	200	6	2
3	5	Book	2	7	2

9. Dodano faktury, a wartości Quantity zostały odpowiednio zmodyfikowane w wyniku działania metody addProduct w klasie Invoice:

	INVOICEID	INVOICENUMBER	QUANTITY
1	8	00001	9
2	9	00002	13
3	10	00099	101

10. Tabela INVOICE_PRODUCT prawidłowo zarejestrowała relację @ManyToMany między fakturami a produktami:

	CANBESOLDIN_INVOICEID	INCLUDEDPRODUCTS_PRODUCTID
1	8	3
2	8	4
3	9	4
4	10	5

11. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);

create table SUPPLIER
(
   SUPPLIERID  INTEGER not null
       primary key,
   CITY        VARCHAR(255),
   COMPANYNAME VARCHAR(255),
   STREET      VARCHAR(255)
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY (CATEGORYID),
   SUPPLIED_BY  INTEGER
```

```
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
        constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
            references INVOICE,
   INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
        constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
            references PRODUCT (PRODUCTID),
   primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```

6. Zadanie VII - standard JPA

1. Stworzyłam nowy plik konfiguracyjny dla standardu JPA: src\META-INF\persistence.xml.

```
<?xml version="1.0"?>

<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="myDatabaseConfig"
                     transaction-type="RESOURCE_LOCAL">
       <properties>
           <property name="hibernate.connection.driver_class"
                     value="org.apache.derby.jdbc.ClientDriver"/>
           <property name="hibernate.connection.url"

value="jdbc:derby://127.0.0.1/BarbaraWojtarowiczJPA"/>
           <property name="hibernate.show_sql" value="true" />
           <property name="hibernate.format_sql" value="true" />
           <property name="hibernate.hbm2ddl.auto" value="create" />
       </properties>
   </persistence-unit>
</persistence>
```

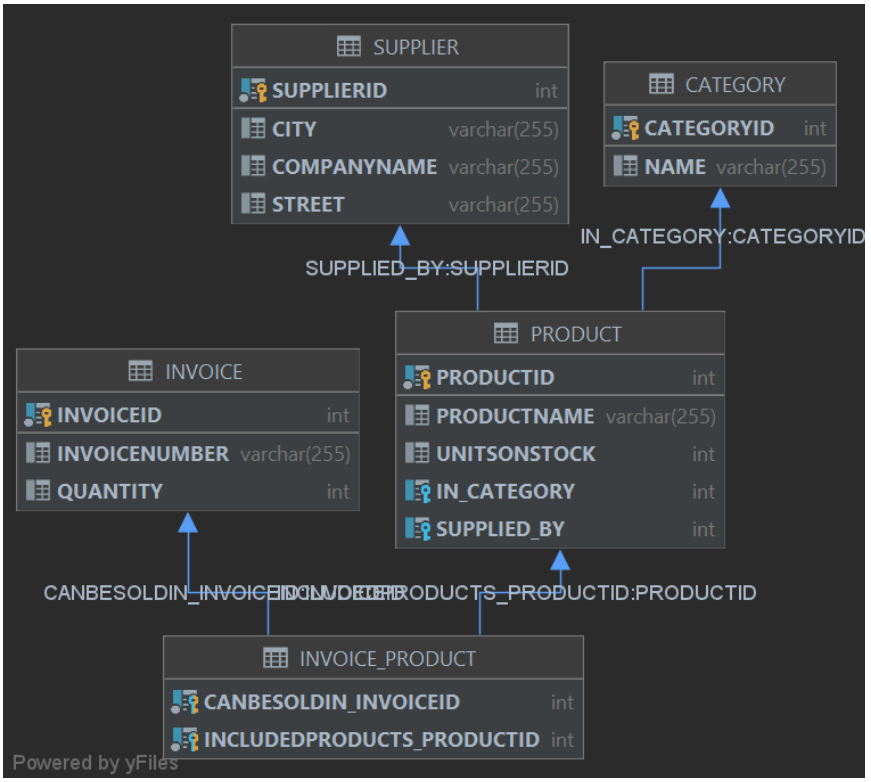
2. Stworzyłam klasę JPAMain, dodawane w niej obiekty nazywałam stosując prefiks “JPA ”, by móc odróżnić wynik działania tej klasy od działania klasy Main.

```
package com.company;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class JPAMain {
   public static void main(String[] args) {

       EntityManagerFactory emf = Persistence.
               createEntityManagerFactory("myDatabaseConfig");
       EntityManager em = emf.createEntityManager();
       EntityTransaction etx = em.getTransaction();
       etx.begin();

       Supplier supplier1 = new Supplier("JPA Supplier1", "Main Street
5", "New York");
       Supplier supplier2 = new Supplier("JPA Supplier2", "Main Road
10", "London");

       Product product1 = new Product("JPA Milk", 10);
       Product product2 = new Product("JPA Banana", 200);
       Product product3 = new Product("JPA Book", 2);

       Category food = new Category("JPA Food");
       Category literature = new Category("JPA Literature");

       Invoice invoice1 = new Invoice("JPA 00001", 5);
       Invoice invoice2 = new Invoice("JPA 00002", 12);
       Invoice invoice3 = new Invoice("JPA 00099", 100);

       supplier1.addProduct(product1);
       supplier2.addProduct(product2);
       supplier2.addProduct(product3);

       food.addProduct(product1);
       food.addProduct(product2);
       literature.addProduct(product3);

       invoice1.addProduct(product1);
       invoice1.addProduct(product1);
       invoice1.addProduct(product1);
```

```
        invoice1.addProduct(product2);
        invoice2.addProduct(product2);
        invoice3.addProduct(product3);

        em.persist(supplier1);
        em.persist(supplier2);
        em.persist(product1);
        em.persist(product2);
        em.persist(product3);
        em.persist(food);
        em.persist(literature);
        em.persist(invoice1);
        em.persist(invoice2);
        em.persist(invoice3);

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

W wyniku ww. działań zaszły następujące zmiany:

3. Schemat bazy był taki sam, jak w Zadaniu VI:



4. Dostawcy “JPA Supplier_” dodali się prawidłowo:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	New York	JPA Supplier1	Main Street 5
2	2	London	JPA Supplier2	Main Road 10

5. Kategorie “JPA _” dodały się prawidłowo:

	CATEGORYID	NAME
1	6	JPA Food
2	7	JPA Literature

6. Produkty “JPA _” dodały się prawidłowo:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	IN_CATEGORY	SUPPLIED_BY
1	3	JPA Milk	10	6	1
2	4	JPA Banana	200	6	2
3	5	JPA Book	2	7	2

7. Faktury “JPA _” dodały się prawidłowo, Quantity również ma odpowiednio większe wartości w zależności od liczby dodanych do faktury w kodzie JPAMain produktów:

	INVOICEID	INVOICENUMBER	QUANTITY
1	8	JPA 00001	9
2	9	JPA 00002	13
3	10	JPA 00099	101

8. Tabela INVOICE_PRODUCT prawidłowo zarejestrowała relację @ManyToMany pomiędzy fakturami i produktami:

	CANBESOLDIN_INVOICEID	INCLUDEDPRODUCTS_PRODUCTID
1	8	3
2	8	4
3	9	4
4	10	5

9. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);
```

```
create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);

create table SUPPLIER
(
   SUPPLIERID  INTEGER not null
       primary key,
   CITY        VARCHAR(255),
   COMPANYNAME VARCHAR(255),
   STREET      VARCHAR(255)
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY (CATEGORYID),
   SUPPLIED_BY  INTEGER
       constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
           references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
       constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
           references INVOICE (INVOICEID),
   INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
       constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
           references PRODUCT (PRODUCTID),
   primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```

7. Zadanie VIII - Kaskady

Zmodyfikowałam model w taki sposób aby było możliwe kaskadowe tworzenie faktur wraz z nowymi produktami, oraz produktów wraz z nową faktur.

1. W klasie Product wprowadziłam operację kaskadową:

```
@ManyToMany(mappedBy = "includedProducts", cascade =
{CascadeType.PERSIST})
private Set<Invoice> canBeSoldIn = new HashSet<>();
```

2. W klasie Invoice również wprowadziłam kaskadę:

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

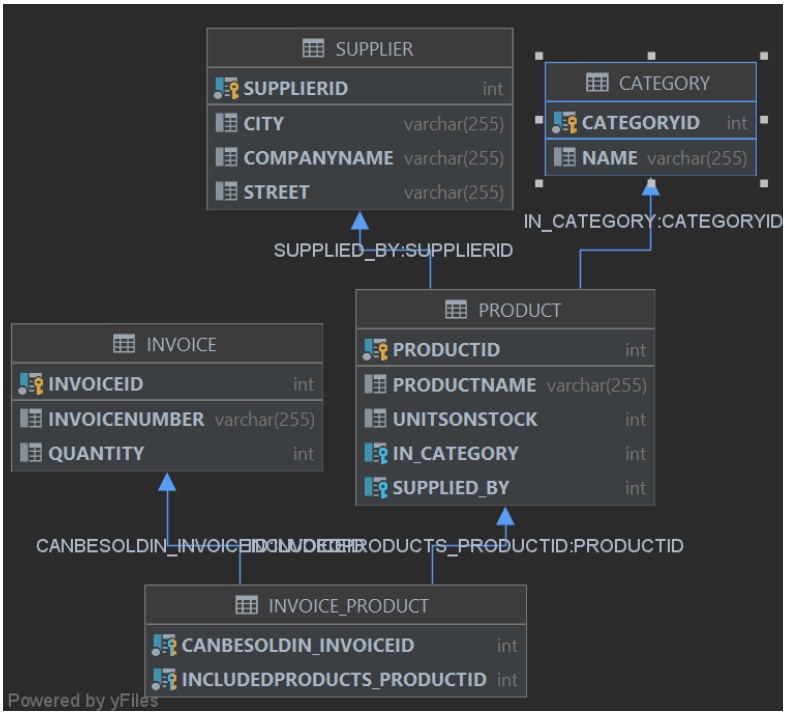
3. W klasie JPAMain dodałam kod odpowiedzialny za dodawanie nowych obiektów do bazy, nazywałam je z przedrostkiem “Cascade _”. Nie zapisywałam faktur w EntityManagerze, by sprawdzić, czy samo zapisanie produktów wystarczy, a mechanizm kaskad sam doda faktury do bazy. Tak się stało, co widać w następnych punktach.

```
public class JPAMain {
   public static void main(String[] args) {

       EntityManagerFactory emf = Persistence.
               createEntityManagerFactory("myDatabaseConfig");
       EntityManager em = emf.createEntityManager();
       EntityTransaction etx = em.getTransaction();
       etx.begin();

       Supplier supplier = new Supplier("Cascade Supplier", "Main Street
5", "New York");

       Product product1 = new Product("Cascade Milk", 10);
       Product product2 = new Product("Cascade Banana", 200);
       Product product3 = new Product("Cascade Book", 2);

       Category food = new Category("Cascade Food");
       Category literature = new Category("Cascade Literature");

       Invoice invoice1 = new Invoice("Cascade 00001", 5);
       Invoice invoice2 = new Invoice("Cascade 00002", 12);
       Invoice invoice3 = new Invoice("Cascade 00099", 100);
```

```
        supplier.addProduct(product1);
        supplier.addProduct(product2);
        supplier.addProduct(product3);

        food.addProduct(product1);
        food.addProduct(product2);
        literature.addProduct(product3);

        invoice1.addProduct(product1);
        invoice1.addProduct(product1);
        invoice1.addProduct(product1);
        invoice1.addProduct(product2);
        invoice2.addProduct(product2);
        invoice3.addProduct(product3);

        em.persist(supplier);
        em.persist(product1);
        em.persist(product2);
        em.persist(product3);
        em.persist(food);
        em.persist(literature);

        //em.persist(invoice1);
        //em.persist(invoice2);
        //em.persist(invoice3);

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

Mechanizm zadziałał poprawnie:

4. Schemat bazy:



5. Dostawca dodał się prawidłowo:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	New York	Cascade Supplier	Main Street 5

6. Kategorie dodały się prawidłowo:

	CATEGORYID	NAME
1	8	Cascade Food
2	9	Cascade Literature

7. Produkty dodały się prawidłowo:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	IN_CATEGORY	SUPPLIED_BY
1	2	Cascade Milk	10	8	1
2	4	Cascade Banana	200	8	1
3	6	Cascade Book	2	9	1

8. Faktury zostały dodane prawidłowo => kaskady działają prawidłowo, Quantity prawidłowo zwiększone w wyniku dodawania produktów:

	INVOICEID	INVOICENUMBER	QUANTITY
1	3	Cascade 00001	9
2	5	Cascade 00002	13
3	7	Cascade 00099	101

9. Relacja produktów i faktur została prawidłowo zmapowana do tabeli

INVOICE_PRODUCT => kaskady działają prawidłowo:

	CANBESOLDIN_INVOICEID	INCLUDEDPRODUCTS_PRODUCTID
1	3	2
2	3	4
3	5	4
4	7	6

10. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);

create table SUPPLIER
(
   SUPPLIERID  INTEGER not null
       primary key,
   CITY        VARCHAR(255),
   COMPANYNAME VARCHAR(255),
   STREET      VARCHAR(255)
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY (CATEGORYID),
   SUPPLIED_BY  INTEGER
       constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
           references SUPPLIER (SUPPLIERID)
```

```
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
       constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
           references INVOICE (INVOICEID),
   INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
       constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
           references PRODUCT (PRODUCTID),
   primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```

8. Zadanie IX - Embedded

1. Embedded classes
   1.1. Stworzyłam klasę @Embeddable Address:

```
package com.company;

import javax.persistence.Embeddable;

@Embeddable
public class Address {
   public String Country;
   public String City;
   public String Street;
   public String ZipCode;

   public Address(){}

   public Address(String country, String city, String street, String
zipCode){
       Country = country;
       City = city;
       Street = street;
       ZipCode = zipCode;
   }
}
```

1.2. Zmodyfikowałam klasę Supplier, zmieniając pola typu String na pole typu Address oraz konstruktory z nim związane:

```
@Embedded
public Address Address;
```

```
public Supplier(String companyName, Address address){
   CompanyName = companyName;
   Address = address;
}

public Supplier(String companyName, Address address, Product product){
   this(companyName, address);
   this.suppliedProducts.add(product);
```

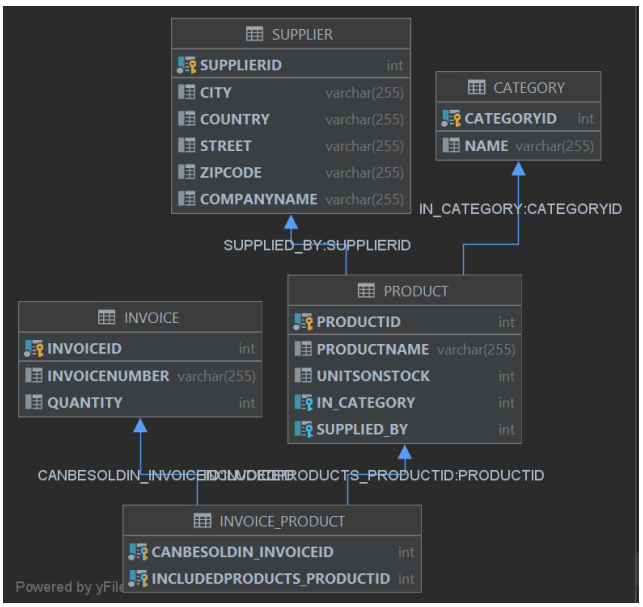
1.3. W klasie JPAMain dodałam dwa adresy i dostawców z nimi związanych. Celowo nie zapisałam w EntityManagerze faktur, by dalej korzystać z mechanizmu kaskad:

```
package com.company;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class JPAMain {
   public static void main(String[] args) {

       EntityManagerFactory emf = Persistence.
               createEntityManagerFactory("myDatabaseConfig");
       EntityManager em = emf.createEntityManager();
       EntityTransaction etx = em.getTransaction();
       etx.begin();

       Address address1 = new Address("UK", "London", "Downing Street
21", "87-111");
       Address address2 = new Address("Poland", "Kraków", "Kawiory 21",
"12-120");

       Supplier supplier1 = new Supplier("English Supplier", address1);
       Supplier supplier2 = new Supplier("Polish Supplier", address2);

       Product product1 = new Product("Embedded Milk", 10);
       Product product2 = new Product("Embedded Banana", 200);
       Product product3 = new Product("Embedded Book", 2);

       Category food = new Category("Embedded Food");
```

```
        Category literature = new Category("Embedded Literature");

        Invoice invoice1 = new Invoice("Embedded 00001", 5);
        Invoice invoice2 = new Invoice("Embedded 00002", 12);
        Invoice invoice3 = new Invoice("Embedded 00099", 100);

        supplier1.addProduct(product1);
        supplier1.addProduct(product2);
        supplier2.addProduct(product3);

        food.addProduct(product1);
        food.addProduct(product2);
        literature.addProduct(product3);

        invoice1.addProduct(product1);
        invoice1.addProduct(product1);
        invoice1.addProduct(product1);
        invoice1.addProduct(product2);
        invoice2.addProduct(product2);
        invoice3.addProduct(product3);

        em.persist(supplier1);
        em.persist(supplier2);
        em.persist(product1);
        em.persist(product2);
        em.persist(product3);
        em.persist(food);
        em.persist(literature);

        //em.persist(invoice1);
        //em.persist(invoice2);
        //em.persist(invoice3);

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

W wyniku ww. działań zaszły następujące zmiany:

1.4. Schemat bazy:



1.5. Kategorie dodały się prawidłowo:

	CATEGORYID	NAME
1	9	Embedded Food
2	10	Embedded Literature

1.6. Dostawcy dodali się prawidłowo:

	SUPPLIERID	CITY	COUNTRY	STREET	ZIPCODE	COMPANYNAME
1	1	London	UK	Downing Street 21	87-111	English Supplier
2	2	Kraków	Poland	Kawiory 21	12-120	Polish Supplier

1.7. Produkty dodały się prawidłowo:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	IN_CATEGORY	SUPPLIED_BY
1	3	Embedded Milk	10	9	1
2	5	Embedded Banana	200	9	1
3	7	Embedded Book	2	10	2

1.8. Faktury dodały się prawidłowo:

	INVOICEID	INVOICENUMBER	QUANTITY
1	4	Embedded 00001	9
2	6	Embedded 00002	13
3	8	Embedded 00099	101

1.9. Tabela INVOICE_PRODUCTS prawidłowo przechowuje relacje między fakturami a produktami:

	CANBESOLDIN_INVOICEID	INCLUDEDPRODUCTS_PRODUCTID
1	4	3
2	4	5
3	6	5
4	8	7

1.10. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);

create table SUPPLIER
(
   SUPPLIERID  INTEGER not null
       primary key,
   CITY        VARCHAR(255),
   COUNTRY     VARCHAR(255),
   STREET      VARCHAR(255),
   ZIPCODE     VARCHAR(255),
   COMPANYNAME VARCHAR(255)
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY (CATEGORYID),
   SUPPLIED_BY  INTEGER
       constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
```

```
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
       constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
           references INVOICE (INVOICEID),
   INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
       constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
           references PRODUCT (PRODUCTID),
   primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```

2. SecondaryTable

2.1. Zmodyfikowałam klasę Supplier, dodając @SecondaryTable. Dane adresowe znajdują się w klasie dostawców:

```
package com.company;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@SecondaryTable(name="ADDRESS")
public class Supplier {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   public int SupplierID;
   public String CompanyName;

   @Column(table="ADDRESS")
   private String Country;
   @Column(table="ADDRESS")
   private String City;
   @Column(table="ADDRESS")
   private String Street;
   @Column(table="ADDRESS")
   private String ZipCode;

   @OneToMany
```

```
    @JoinColumn(name="SUPPLIED_BY")
    public Set<Product> suppliedProducts = new HashSet<>();

    public Supplier(){

    }

    public Supplier(String companyName, String country, String city,
String street, String zipCode){
        CompanyName = companyName;
        Country = country;
        City = city;
        Street = street;
        ZipCode = zipCode;
    }

    public Supplier(String companyName, String country, String city,
String street, String zipCode, Product product){
        this(companyName, country, city, street, zipCode);
        this.suppliedProducts.add(product);
    }

    public void addProduct(Product product){
        this.suppliedProducts.add(product);
        product.setSupplier(this);
    }

    public boolean suppliesProduct(Product product){
        return suppliedProducts.contains(product);
    }
}
```

2.2. W klasie JPAMain dodałam i zapisałam dostawców:

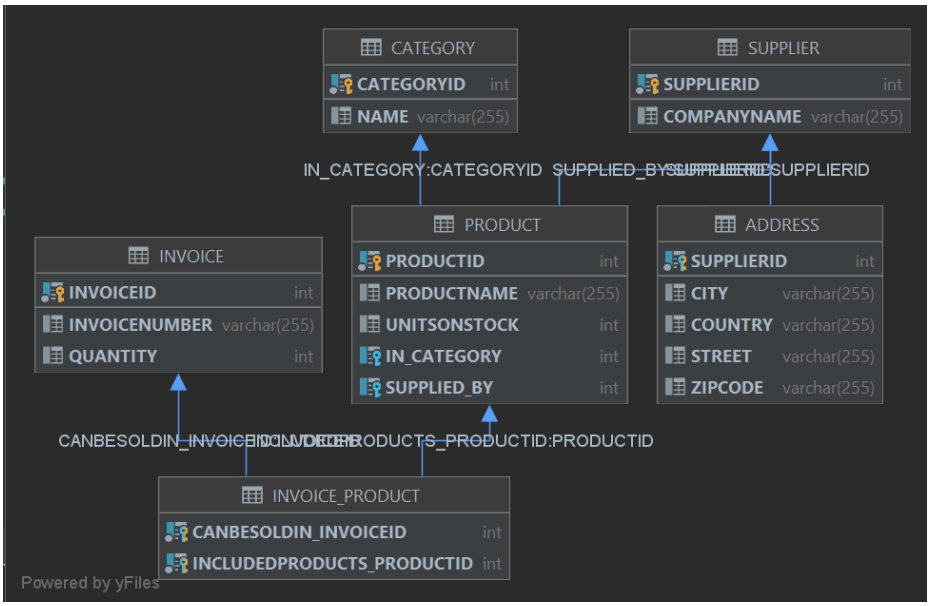
```
Supplier supplier1 = new Supplier("@SecondaryTable English Supplier",
"UK", "London", "Downing Street 21", "87-111");
Supplier supplier2 = new Supplier("@SecondaryTable Polish Supplier",
"Poland", "Kraków", "Kawiory 21", "12-120");
```

```
em.persist(supplier1);
em.persist(supplier2);
```

Reszta kodu w tej klasie pozostała bez zmian (patrz pkt 1.3), zmieniłam jedynie nazwy dodawanych produktów.

W wyniku ww. działań zaszły następujące zmiany:

2.3. Schemat bazy:



2.4. Tabela Address:

	CITY	COUNTRY	STREET	ZIPCODE	SUPPLIERID
1	London	UK	Downing Street 21	87-111	1
2	Kraków	Poland	Kawiory 21	12-120	2

2.5. Tabele Supplier, Category, Product:

	SUPPLIERID	COMPANYNAME
1	1	@SecondaryTable English Supplier
2	2	@SecondaryTable Polish Supplier

2.6. Tabela Invoice:

	INVOICEID	INVOICENUMBER	QUANTITY
1	4	@SecondaryTable 00001	9
2	6	@SecondaryTable 00002	13
3	8	@SecondaryTable 00099	101

2.7. Tabela INVOICE_PRODUCT:

	CANBESOLDIN_INVOICEID	INCLUDEDPRODUCTS_PRODUCTID
1	4	3
2	4	5
3	6	5
4	8	7

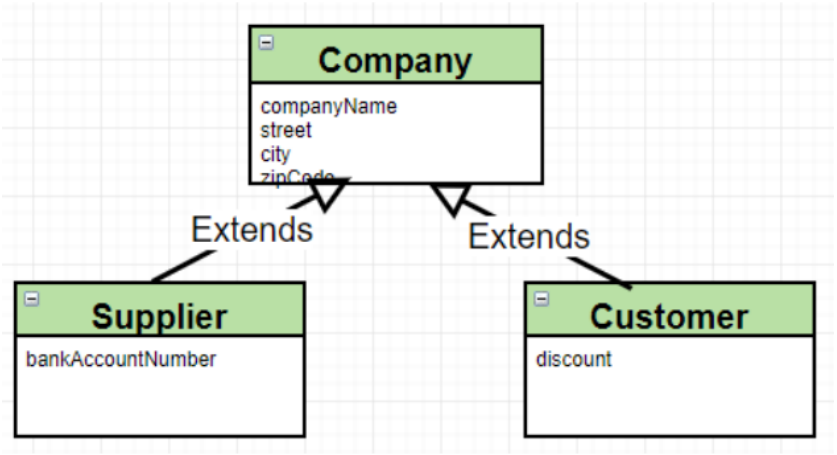
2.8. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);

create table SUPPLIER
(
   SUPPLIERID  INTEGER not null
       primary key,
   COMPANYNAME VARCHAR(255)
);

create table ADDRESS
(
   CITY       VARCHAR(255),
   COUNTRY    VARCHAR(255),
   STREET     VARCHAR(255),
   ZIPCODE    VARCHAR(255),
   SUPPLIERID INTEGER not null
       primary key
       constraint FK46ITY49K53NOSWNL4AQUODOI1
           references SUPPLIER
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY,
   SUPPLIED_BY  INTEGER
```

```
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
        constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
            references INVOICE,
   INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
        constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
            references PRODUCT,
   primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```

9. Zadanie X - Dziedziczenie

Wprowadzono do modelu następującą hierarchię:



1. Strategia SINGLE TABLE
   1.1. Dodałam klasę Company z wbudowaną @SecondaryTable tabelą ADDRESS:

```
package com.company;

import javax.persistence.*;

@Entity
@SecondaryTable(name="ADDRESS")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company {
   @Id
```

```
    @GeneratedValue(
            strategy = GenerationType.AUTO
    )
    public int CompanyID;
    public String CompanyName;

    @Column(table="ADDRESS")
    private String Country;
    @Column(table="ADDRESS")
    private String City;
    @Column(table="ADDRESS")
    private String Street;
    @Column(table="ADDRESS")
    private String ZipCode;

    public Company(){}

    public Company(String companyName, String country, String city,
String street, String zipCode){
        CompanyName = companyName;
        Country = country;
        City = city;
        Street = street;
        ZipCode = zipCode;
    }
}
```

1.2. Zmieniłam klasę Supplier tak, by dziedziczyła z Company, dodałam pole BankAccountNumber:

```
package com.company;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Supplier extends Company {
    private String BankAccountNumber;

    @OneToMany
    @JoinColumn(name="SUPPLIED_BY")
    public Set<Product> suppliedProducts = new HashSet<>();
    public Supplier(){
```

```
    }
    public Supplier(String bankAccountNumber){
        BankAccountNumber = bankAccountNumber;
    }
    public Supplier(String companyName, String country, String city,
String street, String zipCode){
        super(companyName, country, city, street, zipCode);
    }

    public Supplier(String companyName, String country, String city,
String street, String zipCode, Product product){
        this(companyName, country, city, street, zipCode);
        this.suppliedProducts.add(product);
    }

    public Supplier(String companyName, String country, String city,
String street, String zipCode, String bankAccountNumber){
        this(companyName, country, city, street, zipCode);
        setBankAccountNumber(bankAccountNumber);
    }

    public void addProduct(Product product){
        this.suppliedProducts.add(product);
        product.setSupplier(this);
    }

    public boolean suppliesProduct(Product product){
        return suppliedProducts.contains(product);
    }

    public String getBankAccountNumber() { return BankAccountNumber; }

    public void setBankAccountNumber(String bankAccountNumber) {
BankAccountNumber = bankAccountNumber; }

    public Set<Product> getSuppliedProducts() { return suppliedProducts;
}

    public void setSuppliedProducts(Set<Product> suppliedProducts) {
this.suppliedProducts = suppliedProducts; }
}
```

1.3. Dodałam klasę Customer dziedziczącą z Company, posiada pole Discount:

```
package com.company;
```

```
import javax.persistence.Entity;

@Entity
public class Customer extends Company {
   private double Discount;

   public Customer(){}

   public Customer(String companyName, String country, String city,
String street, String zipCode){
       super(companyName, country, city, street, zipCode);
   }

   public Customer(String companyName, String country, String city,
String street, String zipCode, double discount){
       this(companyName, country, city, street, zipCode);
       setDiscount(discount);
   }

   public void setDiscount(double discount) { Discount = discount; }
   public double getDiscount() { return Discount; }
}
```

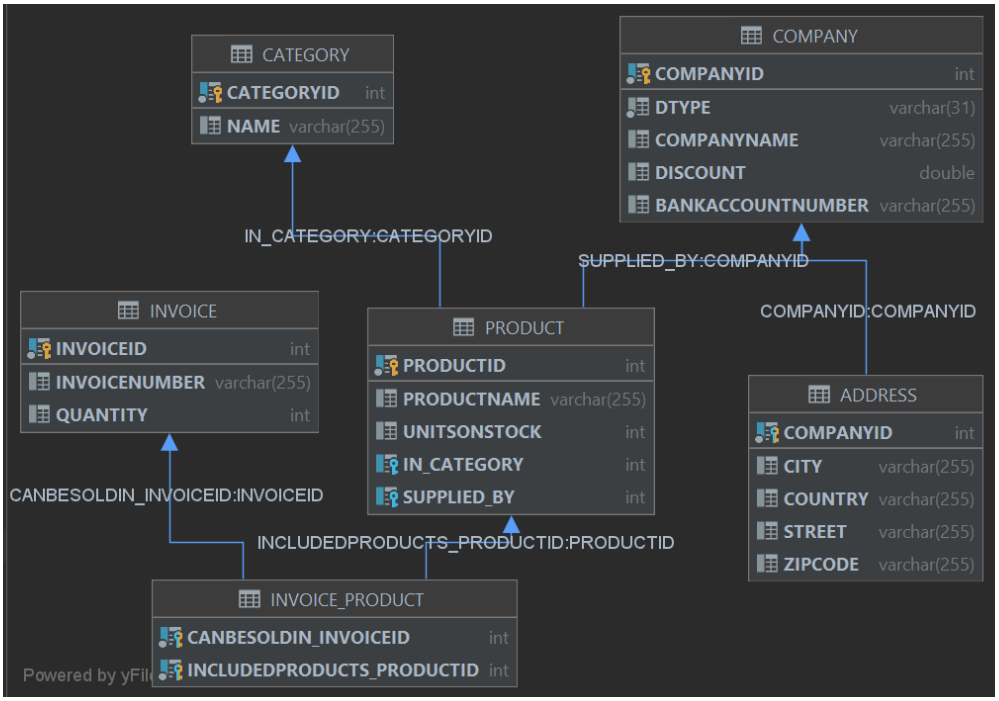
1.4. Klasa JPAMain odpowiada za dodanie kilku klientów, dostawców, produktów, faktur oraz kategorii.

```
public class JPAMain {
   public static void main(String[] args) {

       EntityManagerFactory emf = Persistence.
               createEntityManagerFactory("myDatabaseConfig");
       EntityManager em = emf.createEntityManager();
       EntityTransaction etx = em.getTransaction();
       etx.begin();

       Supplier supplier1 = new Supplier("SINGLE TABLE English
Supplier", "UK", "London", "Downing Street 21", "87-111",
"201010XXX438599424403");
       Supplier supplier2 = new Supplier("SINGLE TABLE Polish Supplier",
"Poland", "Kraków", "Kawiory 21", "12-120", "93582572X5798207438754");

       Customer customer1 = new Customer("SINGLE TABLE English
Customer", "UK", "London", "Downing Street 22", "90-119", 0.01);
       Customer customer2 = new Customer("SINGLE TABLE Polish Customer",
```

```
"Poland", "Kraków", "Kawiory 22", "13-150", 0.0015);

        Product product1 = new Product("SINGLE TABLE Milk", 10);
        Product product2 = new Product("SINGLE TABLE Banana", 200);
        Product product3 = new Product("SINGLE TABLE Book", 2);

        Category food = new Category("SINGLE TABLE Food");
        Category literature = new Category("SINGLE TABLE Literature");

        Invoice invoice1 = new Invoice("SINGLE TABLE 00001", 5);
        Invoice invoice2 = new Invoice("SINGLE TABLE 00002", 12);
        Invoice invoice3 = new Invoice("SINGLE TABLE 00099", 100);

        supplier1.addProduct(product1);
        supplier1.addProduct(product2);
        supplier2.addProduct(product3);

        food.addProduct(product1);
        food.addProduct(product2);
        literature.addProduct(product3);

        invoice1.addProduct(product1);
        invoice1.addProduct(product1);
        invoice1.addProduct(product1);
        invoice1.addProduct(product2);
        invoice2.addProduct(product2);
        invoice3.addProduct(product3);

        em.persist(supplier1);
        em.persist(supplier2);
        em.persist(customer1);
        em.persist(customer2);
        em.persist(product1);
        em.persist(product2);
        em.persist(product3);
        em.persist(food);
        em.persist(literature);

        //em.persist(invoice1);
        //em.persist(invoice2);
        //em.persist(invoice3);

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

W wyniku ww. działań zaszły następujące zmiany:

1.5. Schemat bazy jest zgodny z oczekiwanym, wynikającym z nazwy strategii - Single Table - Supplier i Customer zostali bowiem umieszczeni w tabeli Company:



1.6. Tabele Products, Category:

```
SELECT PRODUCTID, PRODUCTNAME, UNITSONSTOCK, NAME, SUPPLIED_BY
FROM PRODUCT INNER JOIN CATEGORY C on C.CATEGORYID = PRODUCT.IN_CATEGORY
```

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	NAME	SUPPLIED_BY
1	5	SINGLE TABLE Milk	10	SINGLE TABLE Food	1
2	7	SINGLE TABLE Banana	200	SINGLE TABLE Food	1
3	9	SINGLE TABLE Book	2	SINGLE TABLE Literature	2

1.7. Tabele Invoice, INVOICE_PRODUCT:

```
SELECT INVOICEID, INVOICENUMBER, QUANTITY, CANBESOLDIN_INVOICEID
FROM INVOICE INNER JOIN INVOICE_PRODUCT IP on INVOICE.INVOICEID =
IP.CANBESOLDIN_INVOICEID
```

	INVOICEID	INVOICENUMBER	QUANTITY	CANBESOLDIN_INVOICEID
1	6	SINGLE TABLE 00001	9	6
2	6	SINGLE TABLE 00001	9	6
3	8	SINGLE TABLE 00002	13	8
4	10	SINGLE TABLE 00099	101	10

1.8. Tabela Company:

	DTYPE	COMPANYID	COMPANYNAME	DISCOUNT	BANKACCOUNTNUMBER
1	Supplier	1	SINGLE TABLE English Supplier	<null>	201010XXX438599424403
2	Supplier	2	SINGLE TABLE Polish Supplier	<null>	93582572X5798207438754
3	Customer	3	SINGLE TABLE English Customer	0.01	<null>
4	Customer	4	SINGLE TABLE Polish Customer	0.0015	<null>

1.9. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table COMPANY
(
   DTYPE             VARCHAR(31) not null,
   COMPANYID         INTEGER     not null
       primary key,
   COMPANYNAME       VARCHAR(255),
   DISCOUNT          DOUBLE,
   BANKACCOUNTNUMBER VARCHAR(255)
);

create table ADDRESS
(
   CITY      VARCHAR(255),
   COUNTRY   VARCHAR(255),
   STREET    VARCHAR(255),
   ZIPCODE   VARCHAR(255),
   COMPANYID INTEGER not null
       primary key
       constraint FK80GNA9ASWW4EX18HFYUR1RK7M
           references COMPANY
);

create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);
```

```
create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY,
   SUPPLIED_BY  INTEGER
       constraint FKORWD7TI86M4OHR0CRO5GFGLHG
           references COMPANY
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
       constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
           references INVOICE,
   INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
       constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
           references PRODUCT,
   primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```

2. Strategia JOINED

2.1. W klasie Company wprowadziłam następującą zmianę dot. typu strategii dziedziczenia:

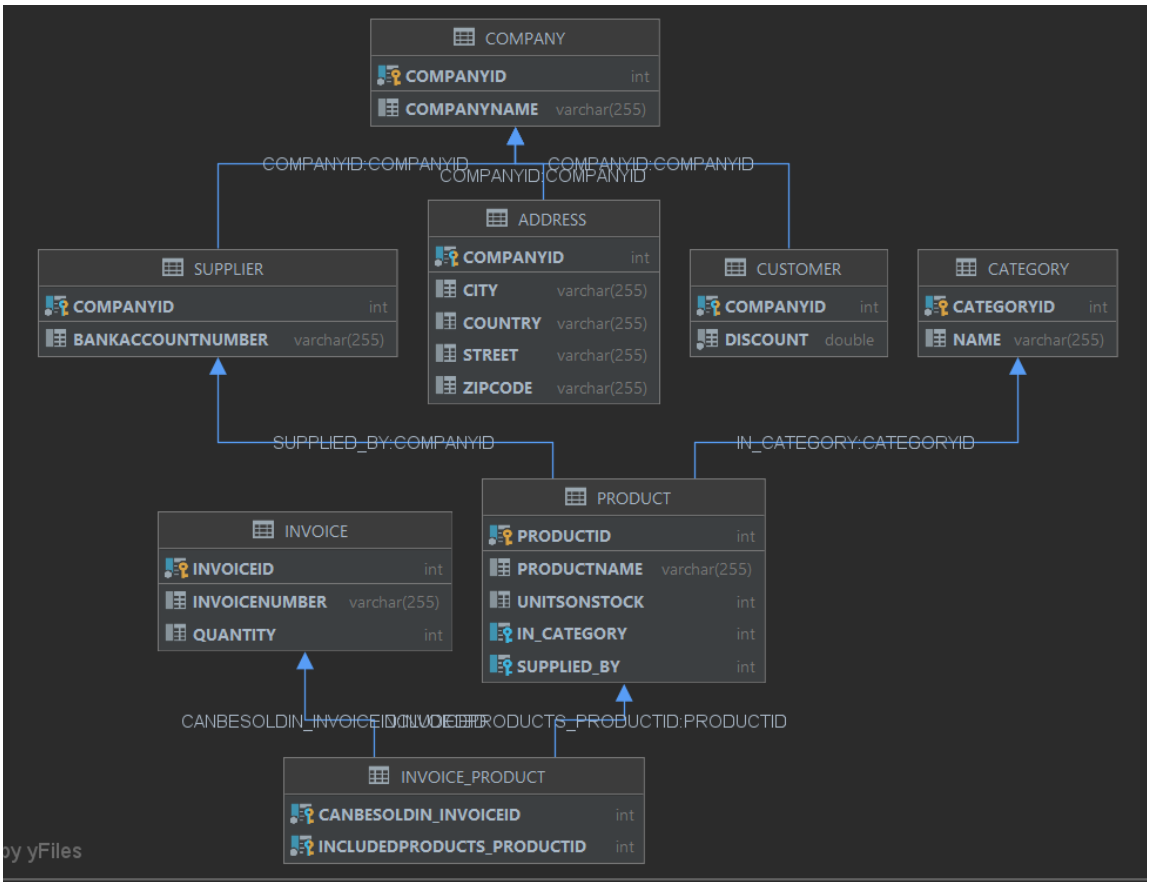
```
@Entity
@SecondaryTable(name="ADDRESS")
@Inheritance(strategy = InheritanceType.JOINED)
public class Company {
   @Id
   @GeneratedValue(
           strategy = GenerationType.AUTO
   )
   public int CompanyID;
   public String CompanyName;

   @Column(table="ADDRESS")
   private String Country;
   @Column(table="ADDRESS")
```

```
    private String City;
    @Column(table="ADDRESS")
    private String Street;
    @Column(table="ADDRESS")
    private String ZipCode;
```

Reszta kodu pozostała bez zmian, w szczególności klasy Supplier i Customer.

Wynik:

2.2. Schemat bazy: wyróżnione zostały osobne tabele Customer oraz Supplier, obecna jest też tabela wspólna Company z polem CompanyName:



2.3. Tabele Company, Customer, Supplier, Address:

	COMPANYID	COMPANYNAME
1	1	SINGLE TABLE English Supplier
2	2	SINGLE TABLE Polish Supplier
3	3	SINGLE TABLE English Customer
4	4	SINGLE TABLE Polish Customer

	DISCOUNT	COMPANYID
1	0.01	3
2	0.0015	4

	BANKACCOUNTNUMBER	COMPANYID
1	201010XXX438599424403	1
2	93582572X5798207438754	2

	CITY	COUNTRY	STREET	ZIPCODE	COMPANYID
1	London	UK	Downing Street 21	87-111	1
2	Kraków	Poland	Kawiory 21	12-120	2
3	London	UK	Downing Street 22	90-119	3
4	Kraków	Poland	Kawiory 22	13-150	4

2.4. Reszta tabel pozostała bez zmian, jak w punkcie 1.

2.5. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table COMPANY
(
   COMPANYID   INTEGER not null
       primary key,
   COMPANYNAME VARCHAR(255)
);

create table ADDRESS
(
   CITY      VARCHAR(255),
   COUNTRY   VARCHAR(255),
   STREET    VARCHAR(255),
   ZIPCODE   VARCHAR(255),
   COMPANYID INTEGER not null
       primary key
       constraint FK80GNA9ASWW4EX18HFYUR1RK7M
```

```
            references COMPANY
);

create table CUSTOMER
(
   DISCOUNT  DOUBLE  not null,
   COMPANYID INTEGER not null
       primary key
       constraint FKO6BOYET0XSRWJCUP3LQK2RC7A
           references COMPANY
);

create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);

create table SUPPLIER
(
   BANKACCOUNTNUMBER VARCHAR(255),
   COMPANYID         INTEGER not null
       primary key
       constraint FK8CHS3V21RG6V4NCJWMLKNN8H1
           references COMPANY
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY,
   SUPPLIED_BY  INTEGER
       constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
           references SUPPLIER
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
```

```
        constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
            references INVOICE,
    INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
        constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
            references PRODUCT,
    primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```

3. Strategia TABLE PER CLASS

3.1. Zmodyfikowałam typ strategii dziedziczenia w klasie Company. Usunęłam też @SecondaryTable ADDRESS i zastąpiłam ją polami typu String opisującymi adres:

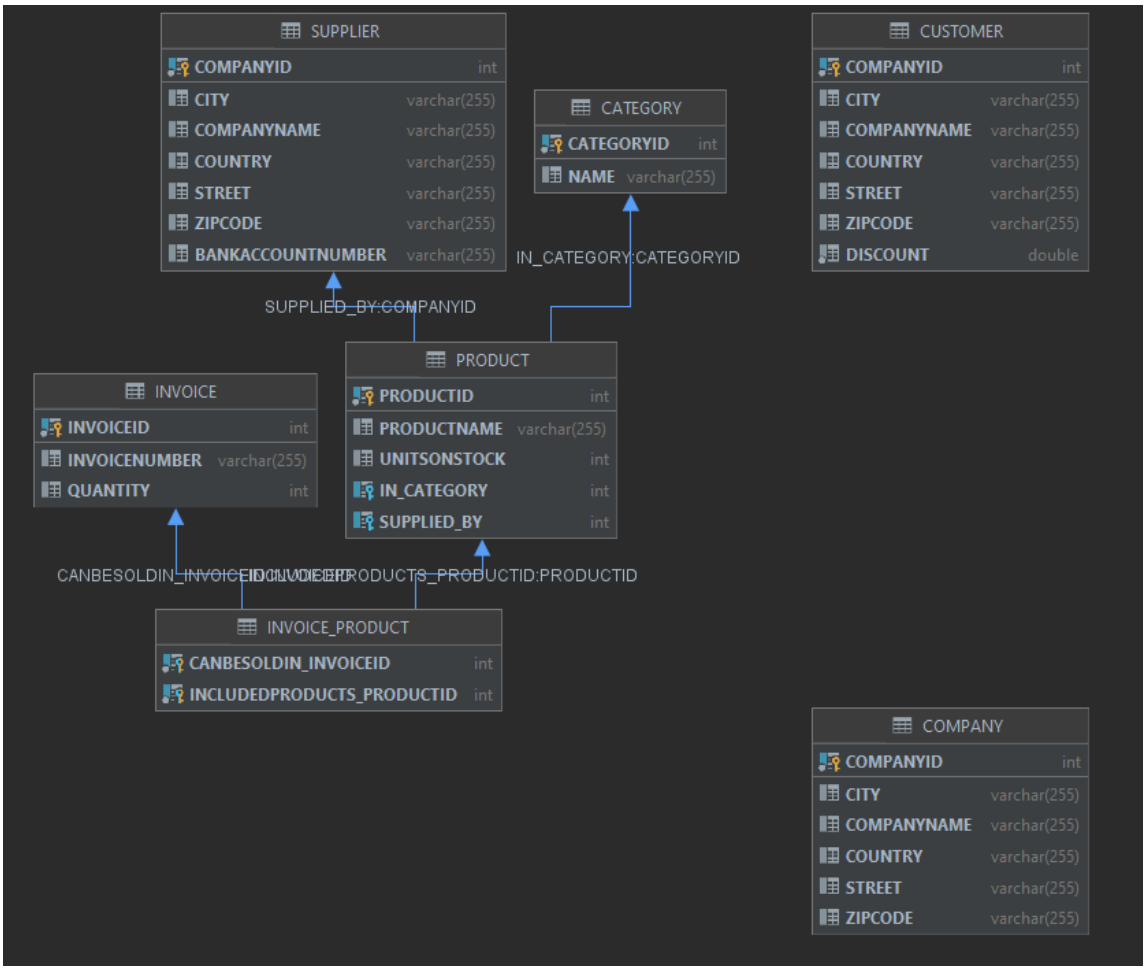
```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Company {
    @Id
    @GeneratedValue(
            strategy = GenerationType.AUTO
    )
    public int CompanyID;
    public String CompanyName;

    private String Country;
    private String City;
    private String Street;
    private String ZipCode;

    public Company(){}

    public Company(String companyName, String country, String city,
String street, String zipCode){
        CompanyName = companyName;
        setCountry(country);
        setCity(city);
        setStreet(street);
        setZipCode(zipCode);
    }

    public String getCountry() { return Country; }
    public void setCountry(String country) { Country = country; }

    public String getCity() { return City; }
```

```
    public void setCity(String city) { City = city; }

    public String getStreet() { return Street; }
    public void setStreet(String street) { Street = street; }

    public String getZipCode() { return ZipCode; }
    public void setZipCode(String zipCode) { ZipCode = zipCode; }
}
```

Reszta kodu pozostała bez zmian, w szczególności klasy Supplier i Customer.

Rezultaty:

3.2. Schemat bazy: zgodnie z oczekiwaniami wynikającymi z nazwy strategii dziedziczenia, każda z klas Supplier oraz Customer pojawiła się w osobnej tabeli, dodatkowo obecna jest tabela Company:



3.3. Tabele Company:

```
SELECT * FROM COMPANY
```

COMPANYID	CITY	COMPANYNAME	COUNTRY	STREET	ZIPCODE

Tabela Company jest pusta.

3.4. Tabela Supplier:

COMPANYID	CITY	COMPANYNAME	COUNTRY	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	London	SINGLE TABLE English Supplier	UK	Downing Street 21	87-111	201010XXX438599424403
2	Kraków	SINGLE TABLE Polish Supplier	Poland	Kawiory 21	12-120	93582572X5798207438754

3.5. Tabela Customer:

COMPANYID	CITY	COMPANYNAME	COUNTRY	STREET	ZIPCODE	DISCOUNT
3	London	SINGLE TABLE English Customer	UK	Downing Street 22	90-119	0.01
4	Kraków	SINGLE TABLE Polish Customer	Poland	Kawiory 22	13-150	0.0015

3.6. Reszta tabel pozostaje bez zmian, jak w punkcie 1.

3.7. Skrypt SQL generujący tabele:

```
create table CATEGORY
(
   CATEGORYID INTEGER not null
       primary key,
   NAME       VARCHAR(255)
);

create table COMPANY
(
   COMPANYID   INTEGER not null
       primary key,
   CITY        VARCHAR(255),
   COMPANYNAME VARCHAR(255),
   COUNTRY     VARCHAR(255),
   STREET      VARCHAR(255),
   ZIPCODE     VARCHAR(255)
);

create table CUSTOMER
(
   COMPANYID   INTEGER not null
       primary key,
```

```
   CITY        VARCHAR(255),
   COMPANYNAME VARCHAR(255),
   COUNTRY     VARCHAR(255),
   STREET      VARCHAR(255),
   ZIPCODE     VARCHAR(255),
   DISCOUNT    DOUBLE  not null
);

create table INVOICE
(
   INVOICEID     INTEGER not null
       primary key,
   INVOICENUMBER VARCHAR(255),
   QUANTITY      INTEGER
);

create table SUPPLIER
(
   COMPANYID         INTEGER not null
       primary key,
   CITY              VARCHAR(255),
   COMPANYNAME       VARCHAR(255),
   COUNTRY           VARCHAR(255),
   STREET            VARCHAR(255),
   ZIPCODE           VARCHAR(255),
   BANKACCOUNTNUMBER VARCHAR(255)
);

create table PRODUCT
(
   PRODUCTID    INTEGER not null
       primary key,
   PRODUCTNAME  VARCHAR(255),
   UNITSONSTOCK INTEGER,
   IN_CATEGORY  INTEGER
       constraint FK6TBNJPVM7I0PL1242U7KA8JCJ
           references CATEGORY,
   SUPPLIED_BY  INTEGER
       constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
           references SUPPLIER
);

create table INVOICE_PRODUCT
(
   CANBESOLDIN_INVOICEID      INTEGER not null
       constraint FKHPFJDVP2OCOR3T6YYLQVVFPP6
```

```
            references INVOICE,
    INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
        constraint FKIYKYA4ARU1S70R4YKTOHXY7C0
            references PRODUCT,
    primary key (CANBESOLDIN_INVOICEID, INCLUDEDPRODUCTS_PRODUCTID)
);
```