

Hibernate, JPA

Aleksandra Mazur

Zadanie II. Basics

a) - h) Konfiguracja

Uruchomiono i podpięto się do serwera *Derby* oraz założono bazę *AMazurJPA*.

Polecenie *show tables* zwróciło poniższe dane.

```
C:\WINDOWS\system32\cmd.exe
ij> connect 'jdbc:derby://127.0.0.1/AMazurJPA;create=true';
ij> show tables;
TABLE_SCHEM      | TABLE_NAME          | REMARKS
-----+-----+-----+
SYS           | SYSALIASES
SYS           | SYSCHECKS
SYS           | SYSCOLPERMS
SYS           | SYSCOLUMNS
SYS           | SYSCONGLOMERATES
SYS           | SYSCONSTRAINTS
SYS           | SYSDEPENDS
SYS           | SYSFILES
SYS           | SYSFOREIGNKEYS
SYS           | SYSKEYS
SYS           | SYSPERMS
SYS           | SYSROLES
SYS           | SYSROUTINEPERMS
SYS           | SYSSCHEMAS
SYS           | SYSSEQUENCES
SYS           | SYSSTATEMENTS
SYS           | SYSSTATISTICS
SYS           | SYSTABLEPERMS
SYS           | SYSTABLES
SYS           | SYSTRIGGERS
SYS           | SYSUSERS
SYS           | SYSVIEWS
SYSIBM         | SYSUMMY1
23 wierszy wybranych
ij>
```

i) Projekt Javowy

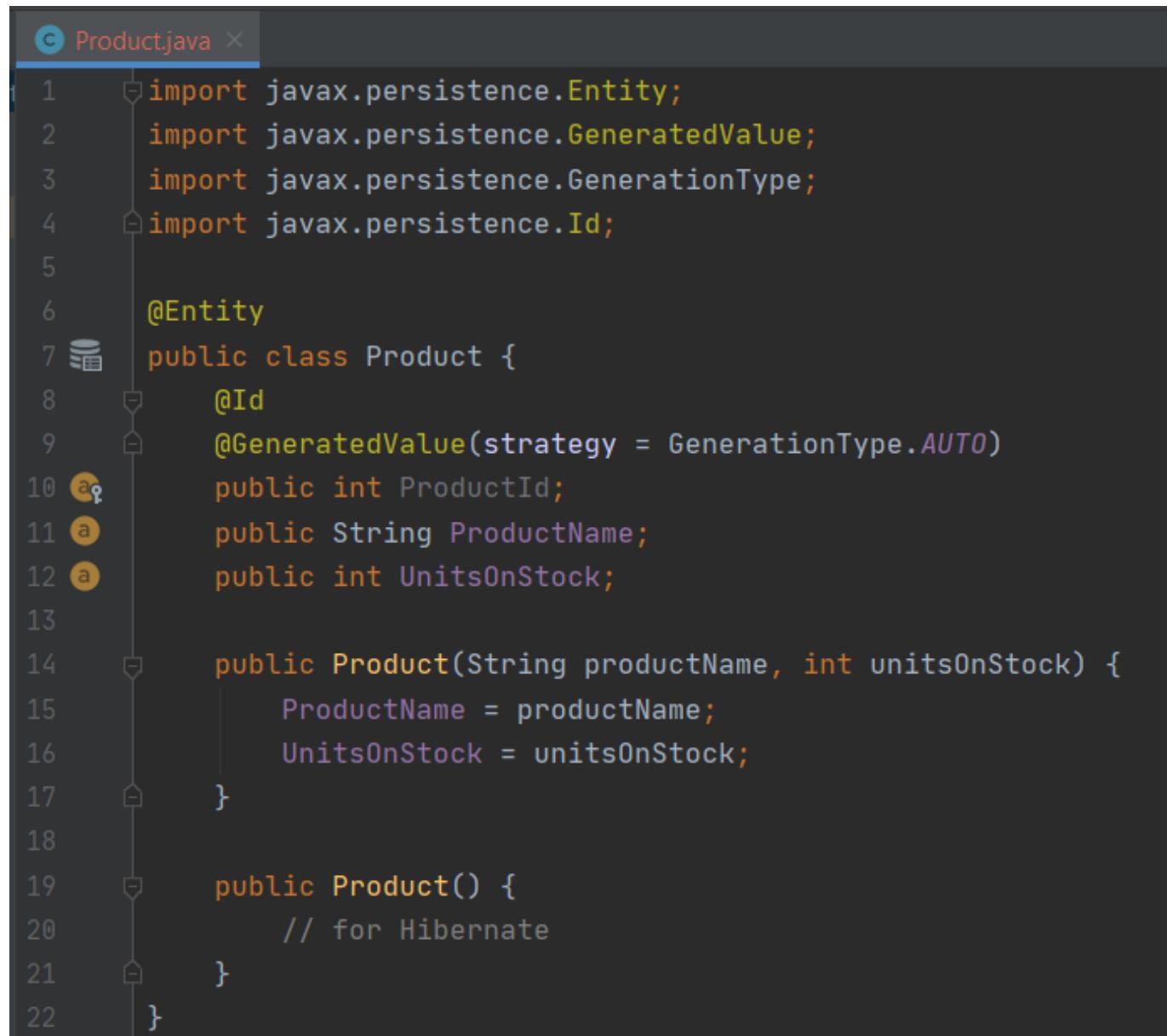
Utworzono projekt w IntelliJ o nazwie **AMazurJPAPractice**.

j), k) Klasa Product

Stworzono klasę **Product** z polami:

- public int ProductId
- public String ProductName
- public int UnitsOnStock

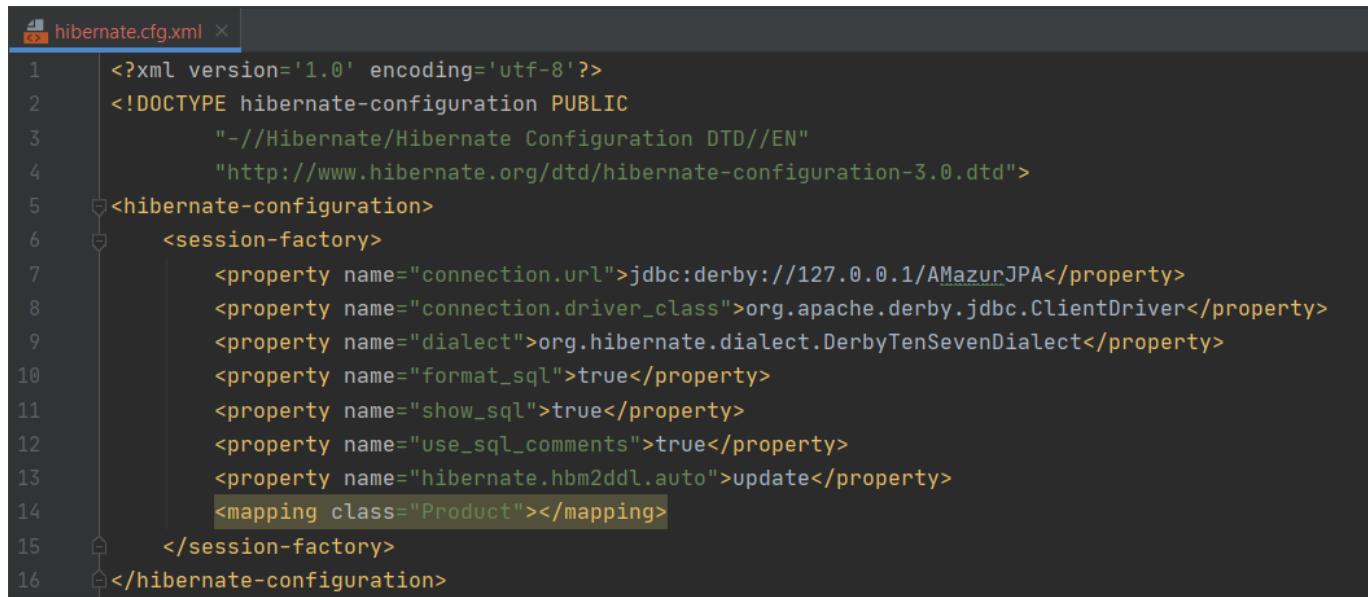
Uzupełniono w klasie elementy potrzebne do zmapowania jej do bazy danych.



```
Product.java
1 import javax.persistence.Entity;
2 import javax.persistence.GeneratedValue;
3 import javax.persistence.GenerationType;
4 import javax.persistence.Id;
5
6 @Entity
7 public class Product {
8     @Id
9     @GeneratedValue(strategy = GenerationType.AUTO)
10    public int ProductId;
11    public String ProductName;
12    public int UnitsOnStock;
13
14    public Product(String productName, int unitsOnStock) {
15        ProductName = productName;
16        UnitsOnStock = unitsOnStock;
17    }
18
19    public Product() {
20        // for Hibernate
21    }
22}
```

I), m) Hibernate config

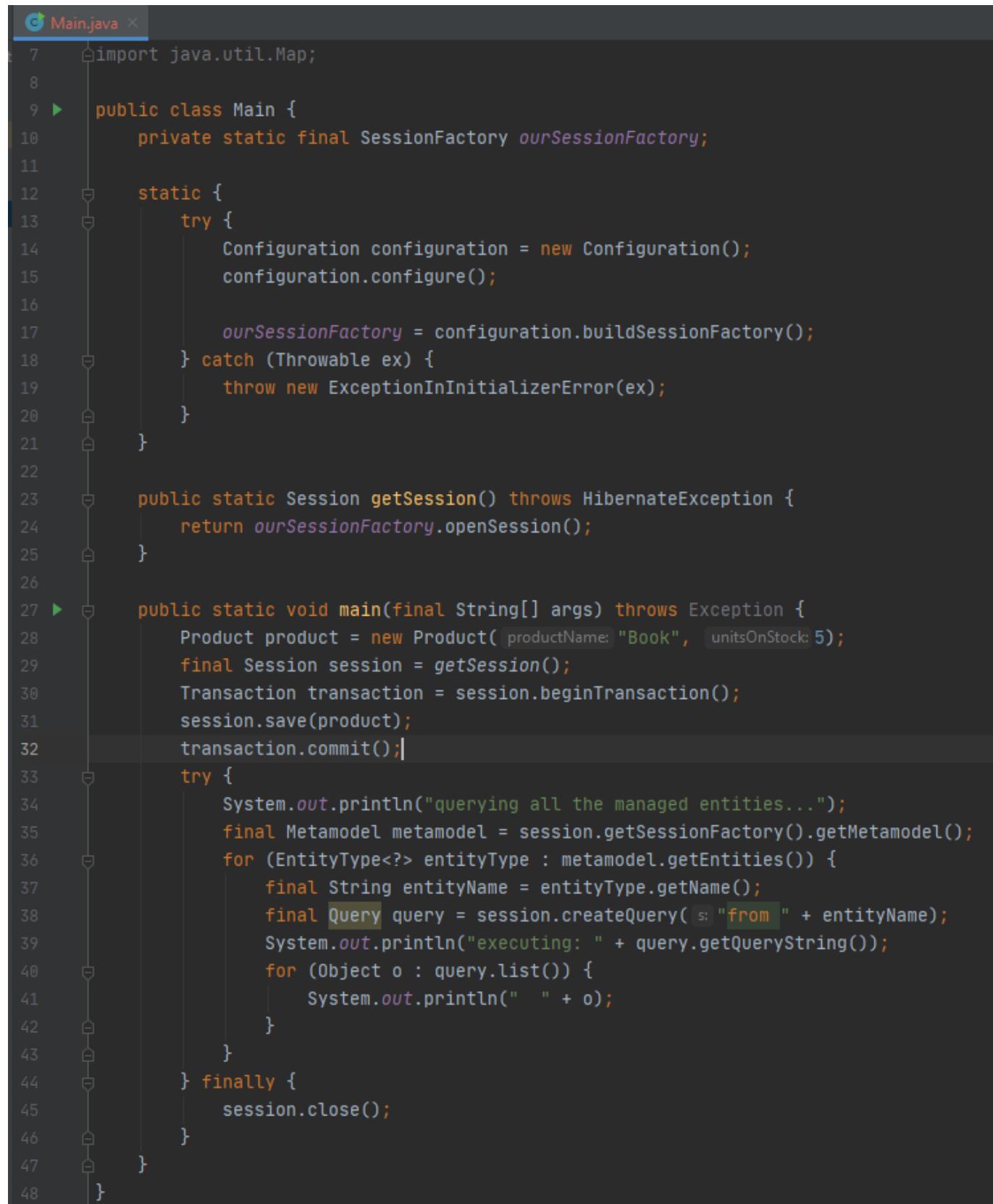
W pliku `hibernate.cfg.xml` uzupełniono potrzebne property.



```
<?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.url">jdbc:derby://127.0.0.1/AMazurJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
    </session-factory>
</hibernate-configuration>
```

Zadanie III. Dodanie produktu

W klasie `Main` stworzono przykładowy produkt i zapisano go do bazy danych.



The screenshot shows a Java code editor with a dark theme. The file is named `Main.java`. The code implements a `Main` class with static methods for managing a session factory and persisting a product entity.

```
1 import java.util.Map;
2
3 public class Main {
4     private static final SessionFactory ourSessionFactory;
5
6     static {
7         try {
8             Configuration configuration = new Configuration();
9             configuration.configure();
10
11             ourSessionFactory = configuration.buildSessionFactory();
12         } catch (Throwable ex) {
13             throw new ExceptionInInitializerError(ex);
14         }
15     }
16
17     public static Session getSession() throws HibernateException {
18         return ourSessionFactory.openSession();
19     }
20
21     public static void main(final String[] args) throws Exception {
22         Product product = new Product(productName: "Book", unitsOnStock: 5);
23         final Session session = getSession();
24         Transaction transaction = session.beginTransaction();
25         session.save(product);
26         transaction.commit();
27         try {
28             System.out.println("querying all the managed entities...");
29             final Metamodel metamodel = session.getSessionFactory().getMetamodel();
30             for (EntityType<?> entityType : metamodel.getEntities()) {
31                 final String entityName = entityType.getName();
32                 final Query query = session.createQuery(s: "from " + entityName);
33                 System.out.println("executing: " + query.getQueryString());
34                 for (Object o : query.list()) {
35                     System.out.println(" " + o);
36                 }
37             }
38         } finally {
39             session.close();
40         }
41     }
42 }
```

```
Main ×  
↑ kwi 28, 2020 8:56:12 AM org.hibernate.engine.transaction.  
↓ INFO: HHH000490: Using JtaPlatform implementation: [or  
Hibernate:  
values  
    next value for hibernate_sequence  
Hibernate:  
    /* insert Product  
        *_ insert  
        into  
            Product  
            (ProductName, UnitsOnStock, ProductId)  
        values  
            (?, ?, ?)  
querying all the managed entities...  
executing: from Product  
Hibernate:  
    /*  
from  
    Product *_ select  
        product0_.ProductId as producti1_0_,  
        product0_.ProductName as productn2_0_,  
        product0_.UnitsOnStock as unitsons3_0_  
    from  
        Product product0_  
Product@151bf776  
  
Process finished with exit code 0
```

Schemat w bazie danych wygląda następująco:

PRODUCT	
!	PRODUCTID int
!	PRODUCTNAME varchar(255)
!	UNITSONSTOCK int

Skrypt tworzący tabelę **Product**:

```
create table PRODUCT
(
    PRODUCTID      INTEGER not null
        constraint "SQL00000000081-120fc126-0171-bf75-6e31-fffffc71c8945"
        primary key,
    PRODUCTNAME    VARCHAR(255),
    UNITSONSTOCK  INTEGER not null
);
```

Jak widać produkt dodał się poprawnie:

The screenshot shows a database interface with two main sections: a top pane for running SQL queries and a bottom pane for viewing the results.

In the top pane, there is a toolbar with various icons (play, stop, refresh, etc.) and dropdown menus. Below the toolbar, a status bar indicates "Tx: Auto". A green checkmark icon is present, followed by the text "SELECT * FROM PRODUCT".

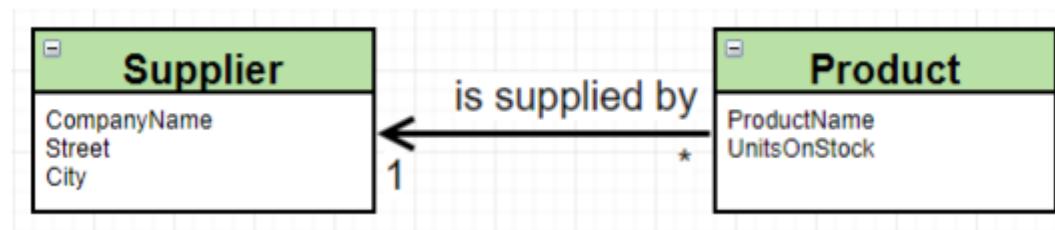
In the bottom pane, there is another toolbar with icons for navigating between tabs and switching between "Output" and "APP.PRODUCT" (which is currently selected). Below this is another status bar with "Tx: Auto" and other buttons like "DDL".

The main result area displays the data from the "APP.PRODUCT" table:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Book	5

Zadanie IV. Klasa Supplier

Zmodyfikowano model wprowadzając pojęcie dostawcy jak poniżej.



Klasa **Supplier** zawiera pola:

- public int SupplierId
- public String CompanyName
- public String Street
- public String City

A screenshot of an IDE showing the **Supplier.java** code. The code defines an Entity class with attributes: SupplierId (auto-generated), CompanyName, Street, and City. It includes a constructor and two empty constructors.

```
1 import javax.persistence.Entity;
2 import javax.persistence.GeneratedValue;
3 import javax.persistence.GenerationType;
4 import javax.persistence.Id;
5
6 @Entity
7 public class Supplier {
8     @Id
9     @GeneratedValue(strategy = GenerationType.AUTO)
10    public int SupplierId;
11    public String CompanyName;
12    public String Street;
13    public String City;
14
15    public Supplier(String companyName, String street, String city) {
16        CompanyName = companyName;
17        Street = street;
18        City = city;
19    }
20
21    public Supplier() {
22    }
23}
```

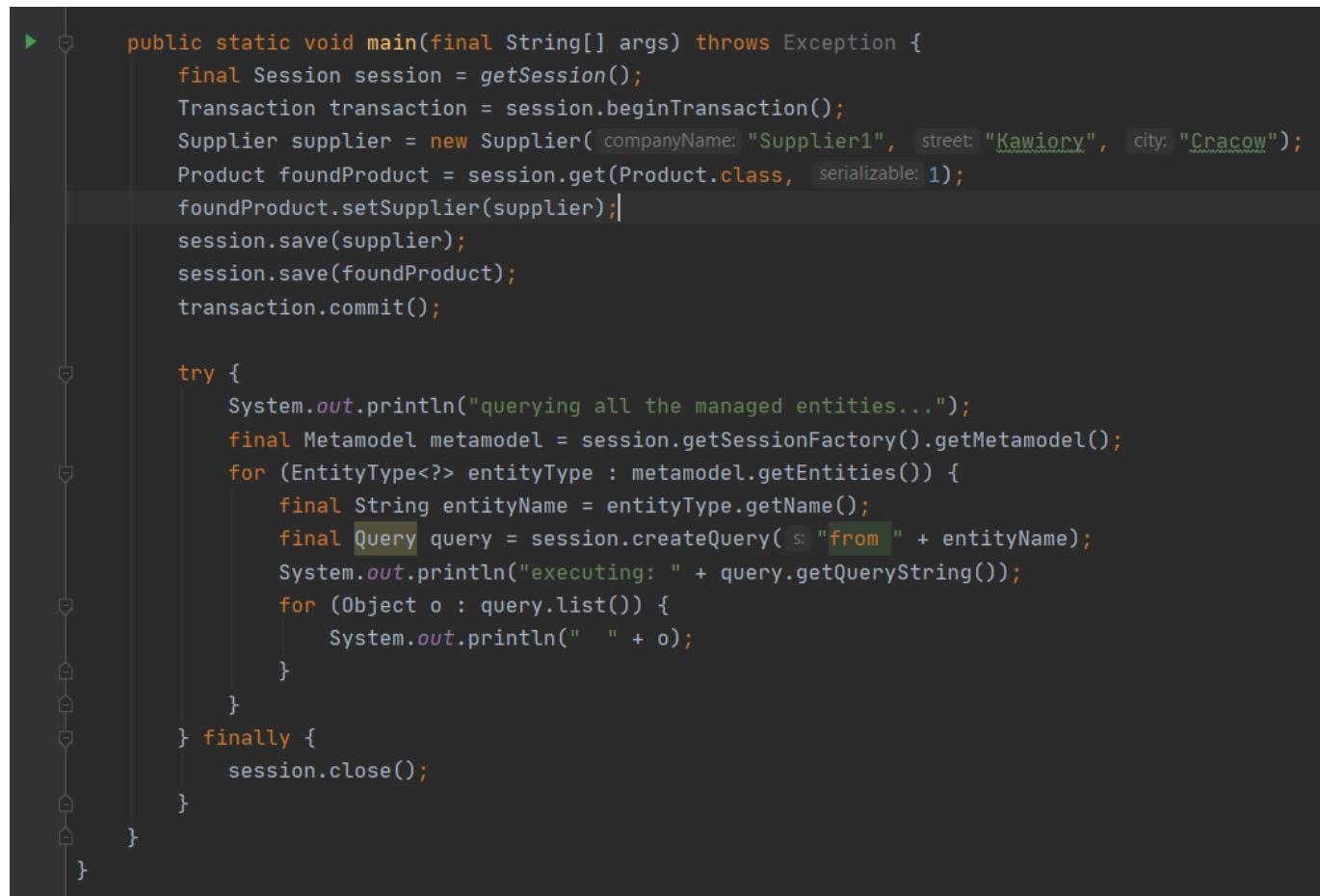
Zmodyfikowano klasę `Product`, dodając do niej pole `Supplier` i funkcję `setSupplier(Supplier supplier)` przypisującą danego dostawcę do produktu.

```
>Main.java ×  Supplier.java ×  Product.java ×
1  public class Product {
2
3      @Id
4      @GeneratedValue(strategy = GenerationType.AUTO)
5      public int ProductId;
6
7      @a
8      public String ProductName;
9
10     @a
11     public int UnitsOnStock;
12
13
14     @ManyToOne
15     public Supplier supplier;
16
17
18     public Product(String productName, int unitsOnStock) {
19
20         ProductName = productName;
21         UnitsOnStock = unitsOnStock;
22     }
23
24     public Product() {
25
26         // for Hibernate
27     }
28
29     public Product(String productName, int unitsOnStock, Supplier supplier) {
30
31         ProductName = productName;
32         UnitsOnStock = unitsOnStock;
33         this.supplier = supplier;
34     }
35
36     public void setSupplier(Supplier supplier) {
37
38         this.supplier = supplier;
39     }
40 }
```

Do pliku `hibernate.cfg.xml` dodano Suppliera.

```
<?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.url">jdbc:derby://127.0.0.1/AMazurJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
    </session-factory>
</hibernate-configuration>
```

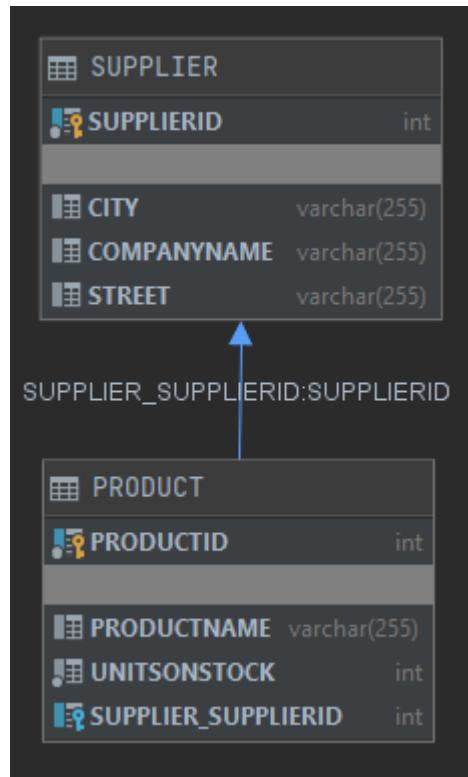
Utworzono nowego dostawcę i przypisano go do wcześniej dodanego produktu.



The screenshot shows a Java code editor with a dark theme. The code is a main method that performs the following steps:

- Creates a session from the factory.
- Begins a transaction.
- Creates a new `Supplier` object with company name "Supplier1", street "Kawiory", and city "Cracow".
- Retrieves a `Product` object from the database.
- Associates the `Supplier` with the `Product`.
- Saves both the `Supplier` and the `Product` back to the database.
- Commits the transaction.
- Attempts to query all managed entities.
- Creates a `Metamodel` and iterates through its entities.
- For each entity, it prints the entity name and executes a query to list all objects of that type.
- Closes the session after the finally block.

Schemat bazy danych wygląda następująco:



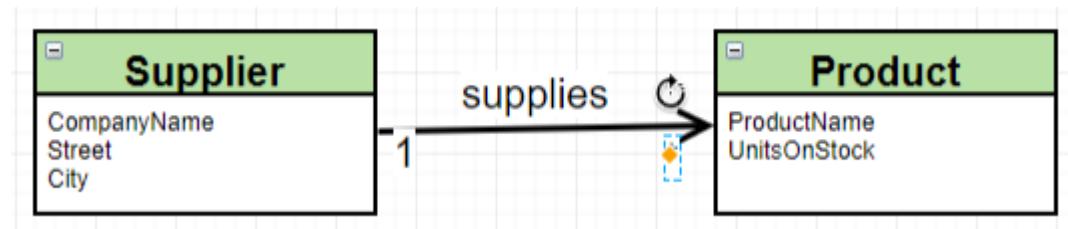
Jak widać dane dodały się poprawnie.

APP.SUPPLIER			
	SUPPLIERID	CITY	COMPANYNAME
1	1	Cracow	Supplier1

APP.PRODUCT			
	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Book	5

Zadanie V. Odwrócona relacja Supplier - Product

Odwrócono relację zgodnie z poniższym schematem.



a) Wariant z tabelą łącznikową

Usunięto z klasy `Product` wcześniej dodane pole `Supplier`.

```
1 import javax.persistence.*;
2
3 @Entity
4 public class Product {
5     @Id
6     @GeneratedValue(strategy = GenerationType.AUTO)
7     public int ProductId;
8     public String ProductName;
9     public int UnitsOnStock;
10
11
12     public Product(String productName, int unitsOnStock) {
13         ProductName = productName;
14         UnitsOnStock = unitsOnStock;
15     }
16
17     public Product() {
18         // for Hibernate
19     }
20
21 }
```

Do klasy `Supplier` dodano zbiór produktów i metodę `addProduct(Product product)` dodającą dany produkt do zbioru produktów dostarczanych przez danego dostawcę.

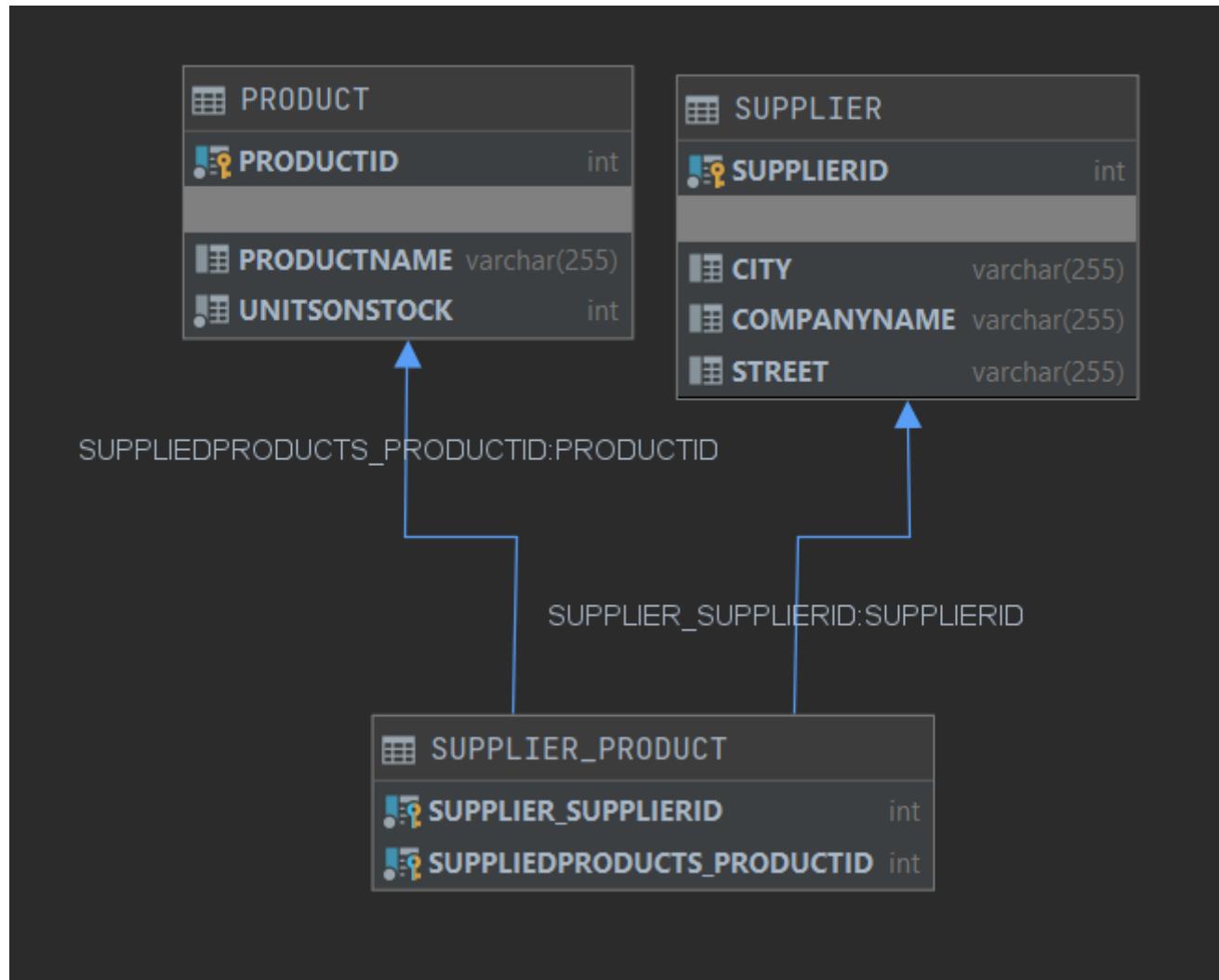
```
1 import javax.persistence.*;
2 import java.util.HashSet;
3 import java.util.Set;
4
5 @Entity
6 public class Supplier {
7     @Id
8     @GeneratedValue(strategy = GenerationType.AUTO)
9     public int SupplierId;
10    public String CompanyName;
11    public String Street;
12    public String City;
13
14    @OneToMany
15    public Set<Product> suppliedProducts = new HashSet<>();
16
17
18    public Supplier(String companyName, String street, String city) {
19        CompanyName = companyName;
20        Street = street;
21        City = city;
22    }
23
24    public Supplier() {
25    }
26
27    public Supplier(String companyName, String street, String city, Product product) {
28        CompanyName = companyName;
29        Street = street;
30        City = city;
31        this.suppliedProducts.add(product);
32    }
33
34    public void addProduct(Product product){
35        this.suppliedProducts.add(product);
36    }
37
38 }
```

Dodano kilka produktów i dostawcę, a następnie przypisano utworzone produkty do zbioru produktów dostarczanych przez dostawcę.

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction transaction = session.beginTransaction();
    Supplier supplier = new Supplier( companyName: "Supplier1", street: "Aleje", city: "Warsaw");
    Product product = new Product( productName: "Window", unitsOnStock: 1);
    Product product2 = new Product( productName: "Book", unitsOnStock: 5);
    supplier.addProduct(product);
    supplier.addProduct(product2);
    session.save(product);
    session.save(product2);
    session.save(supplier);
    transaction.commit();

    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println(" " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

Schemat bazy danych wygląda następująco:



Jak widać powyżej, dodała się tabela łącznikowa **SUPPLIER_PRODUCT**.

Dodane dane:

	SELECT * FROM PRODUCT												
1 ✓	Output APP.PRODUCT X												
	<pre> < < 2 rows < > > G + - Tx: Auto DB ✓ DDL ⚡ ⚡ </pre>												
	<table border="1"> <thead> <tr> <th></th> <th>PRODUCTID</th> <th>PRODUCTNAME</th> <th>UNITSONSTOCK</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1</td> <td>Window</td> <td>1</td> </tr> <tr> <td>2</td> <td>2</td> <td>Book</td> <td>5</td> </tr> </tbody> </table>		PRODUCTID	PRODUCTNAME	UNITSONSTOCK	1	1	Window	1	2	2	Book	5
	PRODUCTID	PRODUCTNAME	UNITSONSTOCK										
1	1	Window	1										
2	2	Book	5										

```
1 ✓ | SELECT * FROM SUPPLIER
```

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	3	Warsaw	Supplier1	Aleje

```
1 ✓ | SELECT * FROM SUPPLIER_PRODUCT
```

	SUPPLIER_SUPPLIERID	SUPPLIEDPRODUCTS_PRODUCTID
1	3	1
2	3	2

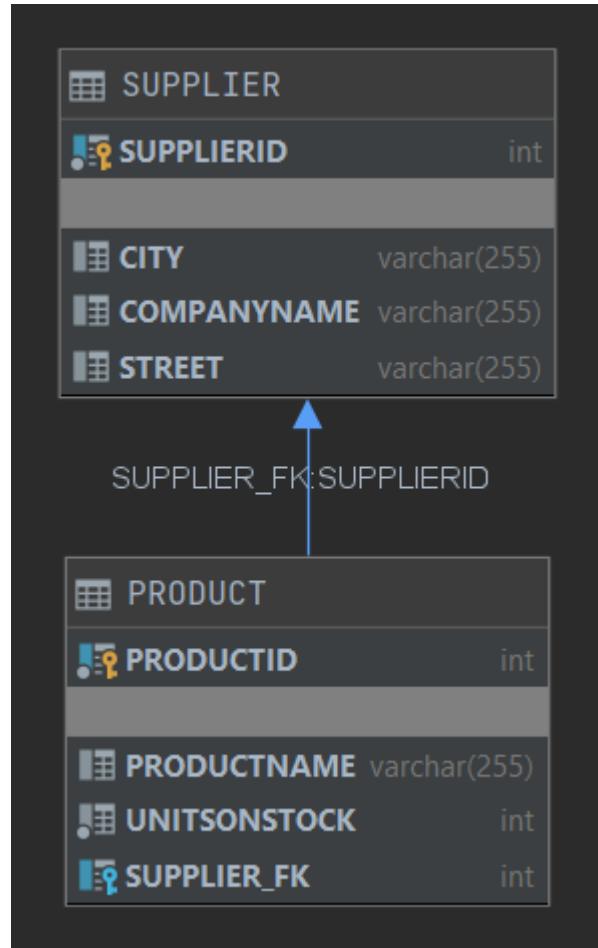
b) Wariant bez tabeli łącznikowej

W klasie `Supplier` nad zbiorem produktów dopisano `@JoinColumn(name="Supplier_FK")`.

```
@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<>();
```

Schemat bazy danych wygląda jak poniżej.



Jak widać nie dodała się tabela łącznikowa. Zamiast niej w tabeli **Product** jest pole **SUPPLIER_FK**.

Skrypt generujący tabele:

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL00000000001-9ac8804c-0171-c07e-3180-fffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL00000000000-ea5bc046-0171-c07e-3180-fffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    SUPPLIER_FK   INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
            references SUPPLIER
);

create index "SQL00000000003-e95a01d1-0171-c07e-3180-fffffc71c8945"
on PRODUCT (SUPPLIER_FK);
```

Dane dodane do bazy:

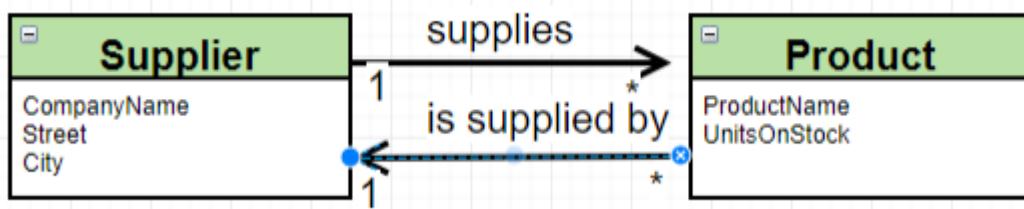
The screenshot shows a database interface with a query editor and a results viewer. The query editor contains the SQL command: `SELECT * FROM PRODUCT`. The results viewer displays the data from the `APP.PRODUCT` table.

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK
1	1	Window	1	3
2	2	Book	5	3

```
1 ✓ | SELECT * FROM SUPPLIER|  
  
Output APP.SUPPLIER X  
|< 1 row >| Tx: Auto | DDL |  
SUPPLIERID CITY COMPANYNAME STREET  
1 3 Warsaw Supplier1 Aleje
```

Zadanie VI. Relacja dwustronna

Zamodelowano relację dwustronną jak poniżej.



Metody dodane/zmodyfikowane w klasie **Supplier**:

- `suppliesProduct(Product product)` zwracająca prawdę, jeżeli dostawca ma dany produkt w zbiorze dostarczanych produktów i fałsz w przeciwnym przypadku.
- `addProduct(Product product)` dodająca produkt do zbioru dostarczanych produktów i przypisującą dostawcę produktowi

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

    public Supplier() {
    }

    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);
    }
}
```

Klasa Product:

Dodano do klasy pole `Supplier` i metodę `setSupplier(Supplier supplier)` przypisującą dostawcę do produktu oraz dodającą produkt do zbioru produktów dostarczanych przez dostawcę, jeżeli nie należy on jeszcze do tego zbioru.

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int ProductId;
    public String ProductName;
    public int UnitsOnStock;

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

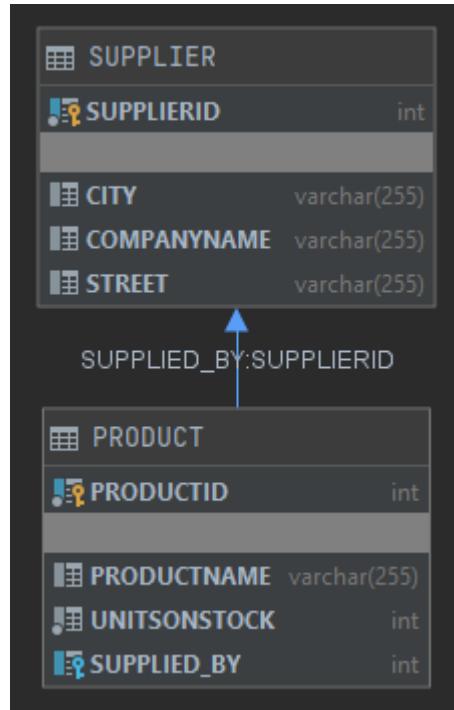
    public Product(String productName, int unitsOnStock) {
        ProductName = productName;
        UnitsOnStock = unitsOnStock;
    }

    public Product() {
        // for Hibernate
    }

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

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

Schemat bazy danych wygląda następująco.



Skrypt tworzący tabelę:

```

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000005-55410238-0171-c07e-3180-fffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID   INTEGER not null
        constraint "SQL0000000004-a1f24232-0171-c07e-3180-fffffc71c8945"
            primary key,
    PRODUCTNAME VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);
create index "SQL0000000006-0128423f-0171-c07e-3180-fffffc71c8945"
    on PRODUCT (SUPPLIED_BY);

```

Dodane dane:

```
1 ✓ | SELECT * FROM SUPPLIER
```

Output APP.SUPPLIER

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	3	Warsaw	Supplier1	Aleje

```
1 ✓ | SELECT * FROM PRODUCT
```

Output APP.PRODUCT

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIED_BY
1	1	Window	1	3
2	2	Book	5	3

Zadanie VII. Klasa Category

a) Modyfikacja modelu

Dodano klasę **Category** z polami:

- private int CategoryId
- private String Name
- private List <Product> Products

Klasa ta posiada metodę **addProduct(Product product)**, która dodaje dany produkt do listy produktów bieżącej kategorii oraz przypisuje kategorię produktowi.

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

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

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

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

    public Category() {}

    public String getName() { return Name; }

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

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

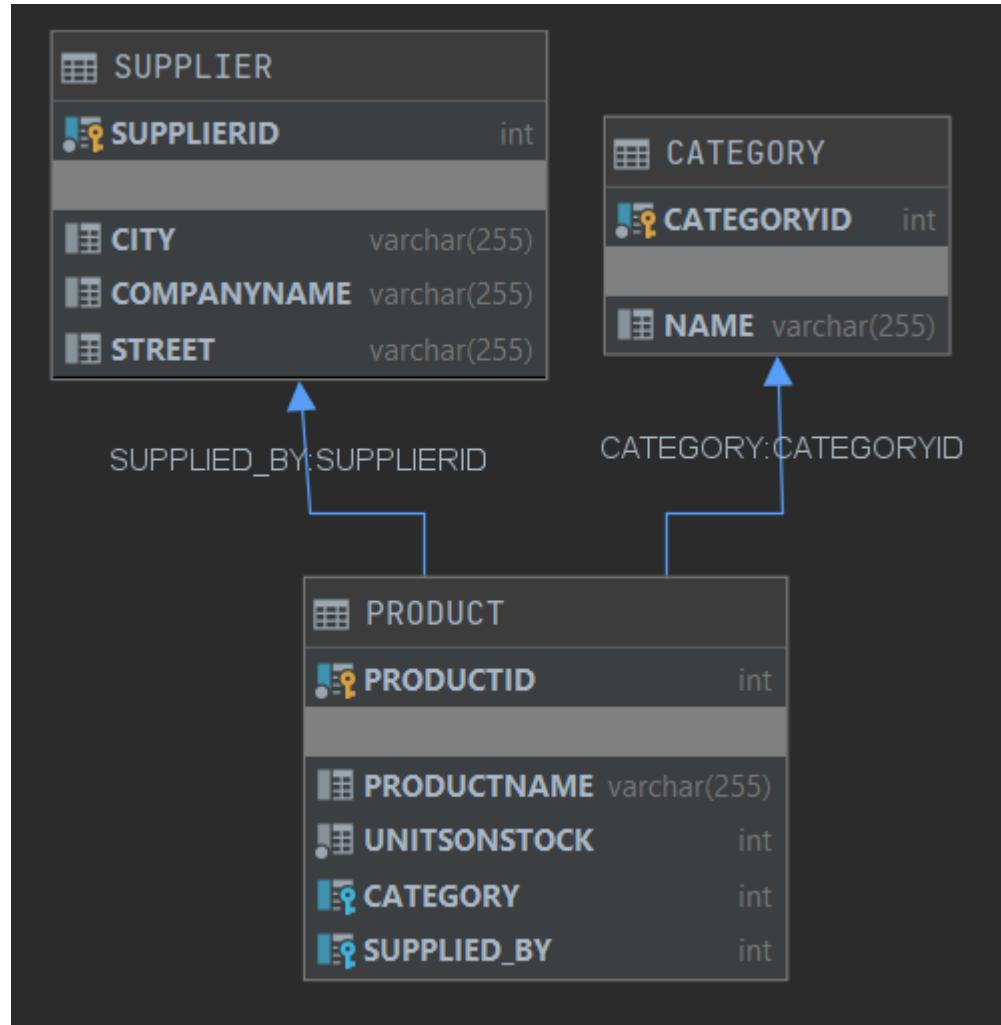
Do pliku `hibernate.cfg.xml` dodano Category.

```
<?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.url">jdbc:derby://127.0.0.1/AMazurJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">create</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
        <mapping class="Category"></mapping>
    </session-factory>
</hibernate-configuration>
```

Zmodyfikowano klasę `Product`, dodając wskazanie na kategorię, do której należy oraz metodę `setCategory(Category category)`, przypisującą daną kategorię produktowi i dodającą produkt do listy produktów kategorii, jeśli jeszcze nie został tam dodany. Zmieniono również dostępność atrybutów we wszystkich klasach na prywatne.

```
public class Product {  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private int ProductId;  
    private String ProductName;  
    private int UnitsOnStock;  
  
    @ManyToOne  
    @JoinColumn(name="SUPPLIED_BY")  
    private Supplier supplier;  
  
    @ManyToOne  
    @JoinColumn(name="CATEGORY")  
    private Category category;  
  
    public Product(String productName, int unitsOnStock) {...}  
  
    public Product() {}  
  
    public Product(String productName, int unitsOnStock, Supplier supplier) {...}  
  
    public void setSupplier(Supplier supplier){...}  
  
    public Category getCategory(){ return this.category; }  
  
    public void setCategory(Category category){  
        this.category = category;  
        if (!category.getProducts().contains(this))  
            category.addProduct(this);  
    }  
}
```

Schemat bazy wygląda następująco:



Skrypt generujący tabele:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000007-c6970388-0171-c07e-3180-fffffc71c8945"
            primary key,
    NAME      VARCHAR(255)
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000009-da248395-0171-c07e-3180-fffffc71c8945"
            primary key,
    CITY       VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET     VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID      INTEGER not null
        constraint "SQL0000000008-f4ca438f-0171-c07e-3180-fffffc71c8945"
            primary key,
    PRODUCTNAME   VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY      INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY,
    SUPPLIED_BY   INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create index "SQL0000000010-b86e839c-0171-c07e-3180-fffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000011-1c0c03a0-0171-c07e-3180-fffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

b), c) Dodanie danych

W **Mainie** dodano dostawcę, kilka produktów i kategorii jak poniżej.

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

    try {

        Transaction transaction = session.beginTransaction();
        Supplier supplier = new Supplier( companyName: "FoodSupplier", street: "Downing Street", city: "London");
        Product product1 = new Product( productName: "Banana", unitsOnStock: 10);
        Product product2 = new Product( productName: "Orange", unitsOnStock: 5);
        Product product3 = new Product( productName: "Carrot", unitsOnStock: 1);
        Category category1 = new Category( name: "Fruits");
        Category category2 = new Category( name: "Vegetables");

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

        category1.addProduct(product1);
        category1.addProduct(product2);
        category2.addProduct(product3);

        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(supplier);
        session.save(category1);
        session.save(category2);
        transaction.commit();

        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println(" " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

Po uruchomieniu powyższego kodu baza danych wygląda następująco:

The screenshot shows a database interface with a toolbar at the top. A query window displays the command `SELECT * FROM SUPPLIER`. Below it, the results are shown in a table titled `APP.SUPPLIER`. The table has four columns: `SUPPLIERID`, `CITY`, `COMPANYNAME`, and `STREET`. One row is present, with values 1, London, FoodSupplier, and Downing Street respectively.

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	London	FoodSupplier	Downing Street

The screenshot shows a database interface with a toolbar at the top. A query window displays the command `SELECT * FROM CATEGORY`. Below it, the results are shown in a table titled `APP.CATEGORY`. The table has two columns: `CATEGORYID` and `NAME`. Two rows are present, with values 1, Fruits and 2, Vegetables respectively.

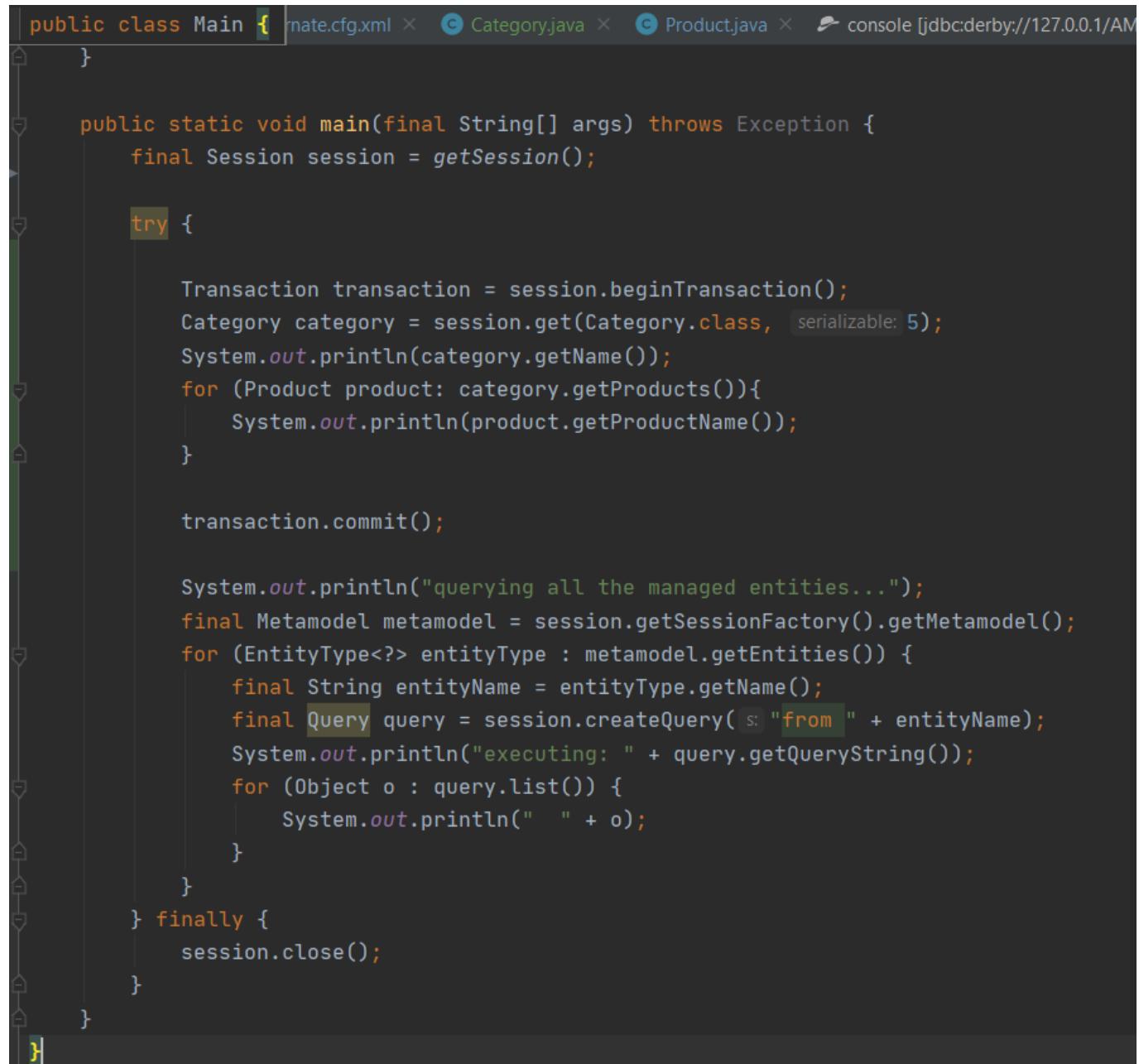
	CATEGORYID	NAME
1	5	Fruits
2	6	Vegetables

The screenshot shows a database interface with a toolbar at the top. A query window displays the command `SELECT * FROM PRODUCT`. Below it, the results are shown in a table titled `APP.PRODUCT`. The table has five columns: `PRODUCTID`, `PRODUCTNAME`, `UNITSONSTOCK`, `CATEGORY`, and `SUPPLIED_BY`. Three rows are present, with values corresponding to Banana, Orange, and Carrot respectively.

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	1	Banana	10	5	4
2	2	Orange	5	5	4
3	3	Carrot	1	6	4

d) Wyciąganie danych z bazy

- Produkty z wybranej kategorii:



The screenshot shows a Java code editor with the following code:

```
public class Main { nate.cfg.xml × Category.java × Product.java × console [jdbc:derby://127.0.0.1/AM]
}

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

    try {
        Transaction transaction = session.beginTransaction();
        Category category = session.get(Category.class, serializable: 5);
        System.out.println(category.getName());
        for (Product product: category.getProducts()){
            System.out.println(product.getProductName());
        }

        transaction.commit();

        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println(" " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

The code demonstrates how to use JPA to query a database. It starts by getting a session from the factory. Inside a try block, it begins a transaction, retrieves a category, and prints its name. It then iterates over the products in that category and prints their names. After committing the transaction, it queries all managed entities using their names as the starting point for the query string. Finally, it closes the session.

Fruits

Hibernate:

```
select
    products0_.CATEGORY as category4_1_0_,
    products0_.ProductId as producti1_1_0_,
    products0_.ProductId as producti1_1_1_,
    products0_.ProductName as productn2_1_1_,
    products0_.UnitsOnStock as unitsons3_1_1_,
    products0_.CATEGORY as category4_1_1_,
    products0_.SUPPLIED_BY as supplied5_1_1_,
    supplier1_.SupplierId as supplier1_2_2_,
    supplier1_.City as city2_2_2_,
    supplier1_.CompanyName as companyn3_2_2_,
    supplier1_.Street as street4_2_2_
from
    Product products0_
left outer join
    Supplier supplier1_
        on products0_.SUPPLIED_BY=supplier1_.SupplierId
where
    products0_.CATEGORY=?
```

Banana

Orange

- Kategoria, do której należy wybrany produkt:

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

    try {

        Transaction transaction = session.beginTransaction();
        Product product = session.get(Product.class, serializable: 3);
        System.out.println(product.getProductName());
        System.out.println(product.getCategory().getName());

        transaction.commit();

        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery(s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println(" " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

```
Hibernate:
```

```
select
    product0_.ProductId as producti1_1_0_,
    product0_.ProductName as productn2_1_0_,
    product0_.UnitsOnStock as unitsons3_1_0_,
    product0_.CATEGORY as category4_1_0_,
    product0_.SUPPLIED_BY as supplied5_1_0_,
    category1_.CategoryId as category1_0_1_,
    category1_.Name as name2_0_1_,
    supplier2_.SupplierId as supplier1_2_2_,
    supplier2_.City as city2_2_2_,
    supplier2_.CompanyName as companyn3_2_2_,
    supplier2_.Street as street4_2_2_
from
    Product product0_
left outer join
    Category category1_
        on product0_.CATEGORY=category1_.CategoryId
left outer join
    Supplier supplier2_
        on product0_.SUPPLIED_BY=supplier2_.SupplierId
where
    product0_.ProductId=?
```

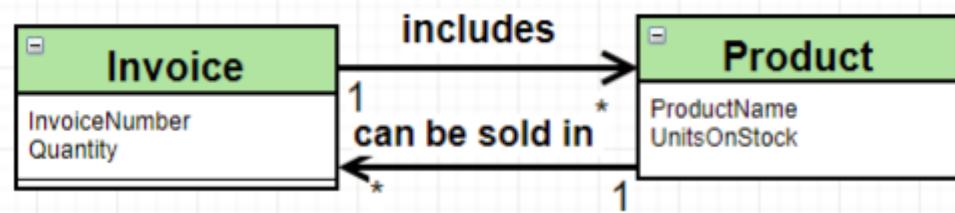
Carrot

Vegetables

Zadanie VIII. Klasa Invoice (relacje wiele-do wielu)

a) Modyfikacja modelu

Zamodelowano relację wiele-do-wielu jak poniżej.



Stworzono klasę **Invoice** o atrybutach:

- `private int InvoiceNumber`
- `private int Quantity`
- `private Set <Product> includesProducts`

Dodano do niej między innymi metodę `addProduct(Product product)`, dodającą produkt do zbioru produktów faktury, fakturę do zbioru faktur danego produktu i zwiększającą ilość produktów na fakturze o 1.

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

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

    public Invoice(int quantity) { Quantity = quantity; }

    public Invoice() {}

    public void addProduct(Product product){
        includesProducts.add(product);
        product.getCanBeSoldIn().add(this);
        this.Quantity += 1;
    }

    public int getQuantity() { return Quantity; }

    public Set<Product> getIncludesProducts() { return includesProducts; }

    public int getInvoiceNumber() { return InvoiceNumber; }
}
```

Do klasy `Product` dodano zbiór faktur, które zawierają bieżący produkt:

- private Set<Invoice> canBeSoldIn

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductId;
    private String ProductName;
    private int UnitsOnStock;

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

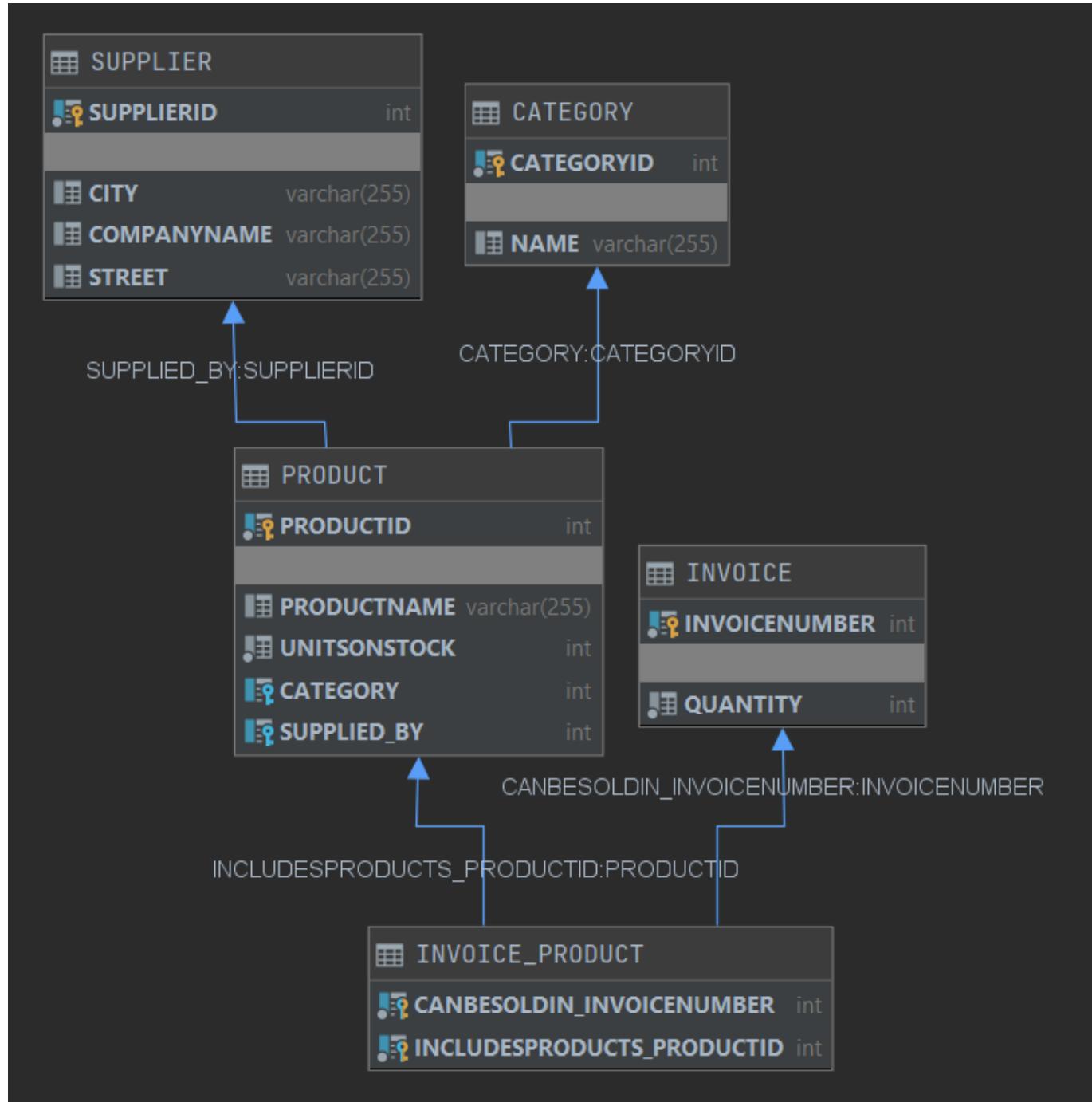
    @ManyToOne
    @JoinColumn(name="CATEGORY")
    private Category category;

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

Do pliku `hibernate.cfg.xml` dodano `Invoice`.

```
<?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.url">jdbc:derby://127.0.0.1/AMazurJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
        <mapping class="Category"></mapping>
        <mapping class="Invoice"></mapping>
    </session-factory>
</hibernate-configuration>
```

Schemat bazy danych wygląda następująco:



Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000018-cccb420a-0171-c670-619d-fffffc71c8945"
            primary key,
    NAME      VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000019-7fde0210-0171-c670-619d-fffffc71c8945"
            primary key,
    QUANTITY    INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000022-194c4222-0171-c670-619d-fffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000021-a61e821c-0171-c670-619d-fffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY   INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000020-f2f9c216-0171-c670-619d-fffffc71c8945"
);
```

```
    primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000023-cc830228-0171-c670-619d-fffffc71c8945"
on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000024-2eac822c-0171-c670-619d-fffffc71c8945"
on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000025-90da0230-0171-c670-619d-fffffc71c8945"
on PRODUCT (CATEGORY);

create index "SQL0000000026-f30b8234-0171-c670-619d-fffffc71c8945"
on PRODUCT (SUPPLIED_BY);
```

Dodanie obiektów do bazy

Dodano kilka produktów, faktur, dostawcę i kategorie.

```
Transaction transaction = session.beginTransaction();
Product product1 = new Product( productName: "TV", unitsOnStock: 2);
Product product2 = new Product( productName: "Computer", unitsOnStock: 1);
Product product3 = new Product( productName: "Telephone", unitsOnStock: 4);
Supplier supplier = new Supplier( companyName: "Media Supplier", street: "Street", city: "New York");
Category category = new Category( name: "Electronics");
Invoice invoice1 = new Invoice( quantity: 0);
Invoice invoice2 = new Invoice( quantity: 0);

supplier.addProduct(product1);
supplier.addProduct(product2);
supplier.addProduct(product3);
category.addProduct(product1);
category.addProduct(product2);
category.addProduct(product3);
invoice1.addProduct(product1);
invoice1.addProduct(product2);
invoice2.addProduct(product1);
invoice2.addProduct(product2);
invoice2.addProduct(product3);

session.save(product1);
session.save(product2);
session.save(product3);
session.save(supplier);
session.save(category);
session.save(invoice1);
session.save(invoice2);
transaction.commit();
```

Zawartość bazy danych po uruchomieniu powyższego kodu:

1 ✓ | SELECT * FROM SUPPLIER

Output APP.SUPPLIER ×

| SUPPLIERID | CITY | COMPANYNAME | STREET |

1 | 4 | London | FoodSupplier | Downing Street

2 | 10 | New York | Media Supplier | Street

1 ✓ | SELECT * FROM CATEGORY

Output APP.CATEGORY ×

| CATEGORYID | NAME |

1 | 5 | Fruits

2 | 6 | Vegetables

3 | 11 | Electronics

1 ✓ | SELECT * FROM PRODUCT

Output APP.PRODUCT ×

| PRODUCTID | PRODUCTNAME | UNITSONSTOCK | CATEGORY | SUPPLIED_BY |

1 | 1 | Banana | 10 | 5 | 4

2 | 2 | Orange | 5 | 5 | 4

3 | 3 | Carrot | 1 | 6 | 4

4 | 7 | TV | 2 | 11 | 10

5 | 8 | Computer | 1 | 11 | 10

6 | 9 | Telephone | 4 | 11 | 10

```
1 ✓ | SELECT * FROM INVOICE|
```

Output APP.INVOICE X

|< < 2 rows <|> >| Tx: Auto | DB |

	INVOICENUMBER	QUANTITY
1	12	2
2	13	3

```
1 ✓ | SELECT * FROM INVOICE_PRODUCT|
```

Output APP.INVOICE_PRODUCT X

|< < 5 rows <|> >| Tx: Auto | DB | ✓ | DDL | |

	CANBESOLDIN_INVOICENUMBER	INCLUDESPRODUCTS_PRODUCTID
1	12	7
2	12	8
3	13	7
4	13	8
5	13	9

b) Wypisanie produktów sprzedanych w ramach wybranej faktury

```
Transaction transaction = session.beginTransaction();
Invoice invoice = session.get(Invoice.class, serializable: 12);
for(Product product: invoice.getIncludesProducts())
    System.out.println(product.getProductName());

transaction.commit();
```

```
Hibernate:
select
    canbesoldi0_.includesProducts_ProductId as includes2_2_0_,
    canbesoldi0_.canBeSoldIn_InvoiceNumber as canbesol1_2_0_,
    invoice1_.InvoiceNumber as invoicen1_1_1_,
    invoice1_.Quantity as quantity2_1_1_
from
    Invoice_Product canbesoldi0_
inner join
    Invoice invoice1_
        on canbesoldi0_.canBeSoldIn_InvoiceNumber=invoice1_.InvoiceNumber
where
    canbesoldi0_.includesProducts_ProductId=?
```

TV
Computer

c) Wypisanie faktur w ramach, których był sprzedany wybrany produkt

```

Transaction transaction = session.beginTransaction();
Product product = session.get(Product.class, serializable: 8);
for (Invoice invoice: product.getCanBeSoldIn())
    System.out.println("Invoice number: " + invoice.getInvoiceNumber());

transaction.commit();

```

```

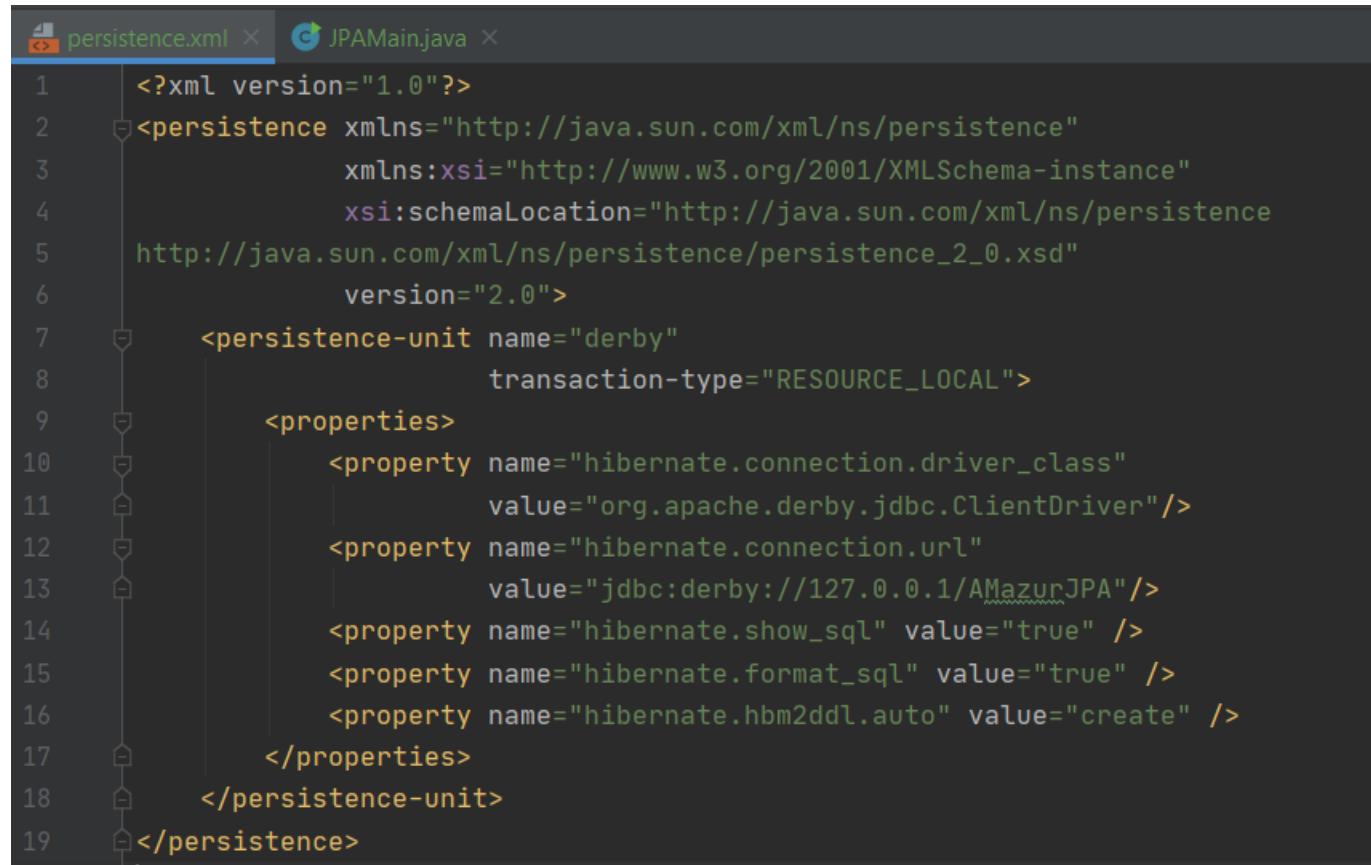
Hibernate:
    select
        product0_.ProductId as producti1_3_0_,
        product0_.ProductName as productn2_3_0_,
        product0_.UnitsOnStock as unitsons3_3_0_,
        product0_.CATEGORY as category4_3_0_,
        product0_.SUPPLIED_BY as supplied5_3_0_,
        canbesoldi1_.includesProducts_ProductId as includes2_2_1_,
        invoice2_.InvoiceNumber as canbesol1_2_1_,
        invoice2_.InvoiceNumber as invicen1_1_2_,
        invoice2_.Quantity as quantity2_1_2_,
        category3_.CategoryId as category1_0_3_,
        category3_.Name as name2_0_3_,
        supplier4_.SupplierId as supplier1_4_4_,
        supplier4_.City as city2_4_4_,
        supplier4_.CompanyName as companyn3_4_4_,
        supplier4_.Street as street4_4_4_
    from
        Product product0_
    left outer join
        Invoice_Product canbesoldi1_
            on product0_.ProductId=canbesoldi1_.includesProducts_ProductId
    left outer join
        Invoice invoice2_
            on canbesoldi1_.canBeSoldIn_InvoiceNumber=invoice2_.InvoiceNumber
    left outer join
        Category category3_
            on product0_.CATEGORY=category3_.CategoryId
    left outer join
        Supplier supplier4_
            on product0_.SUPPLIED_BY=supplier4_.SupplierId
    where
        product0_.ProductId=?
```

Invoice number: 13

Invoice number: 12

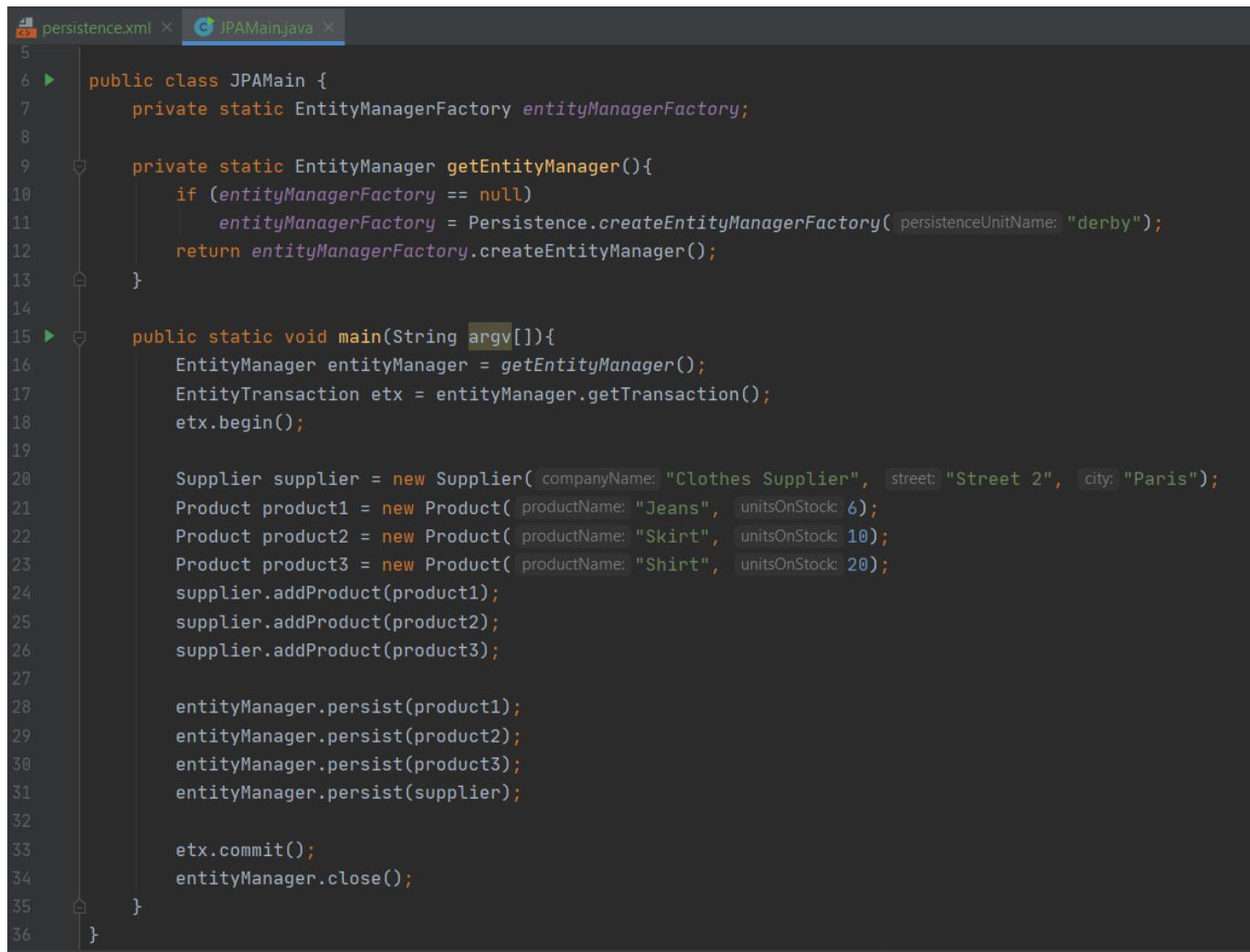
Zadanie IX-X. JPA

Stworzono plik `persistence.xml` w folderze `META-INF` oraz nowego maina `JPAMain.java` jak poniżej:



The screenshot shows a code editor with two tabs: "persistence.xml" and "JPAMain.java". The "persistence.xml" tab is active, displaying the XML configuration for a persistence unit named "derby". The configuration includes properties for the connection driver and URL, and settings for Hibernate like show SQL and auto-create DDL. The "JPAMain.java" tab is visible in the background.

```
<?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="derby"
                      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/AMazurJPA"/>
            <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>
```

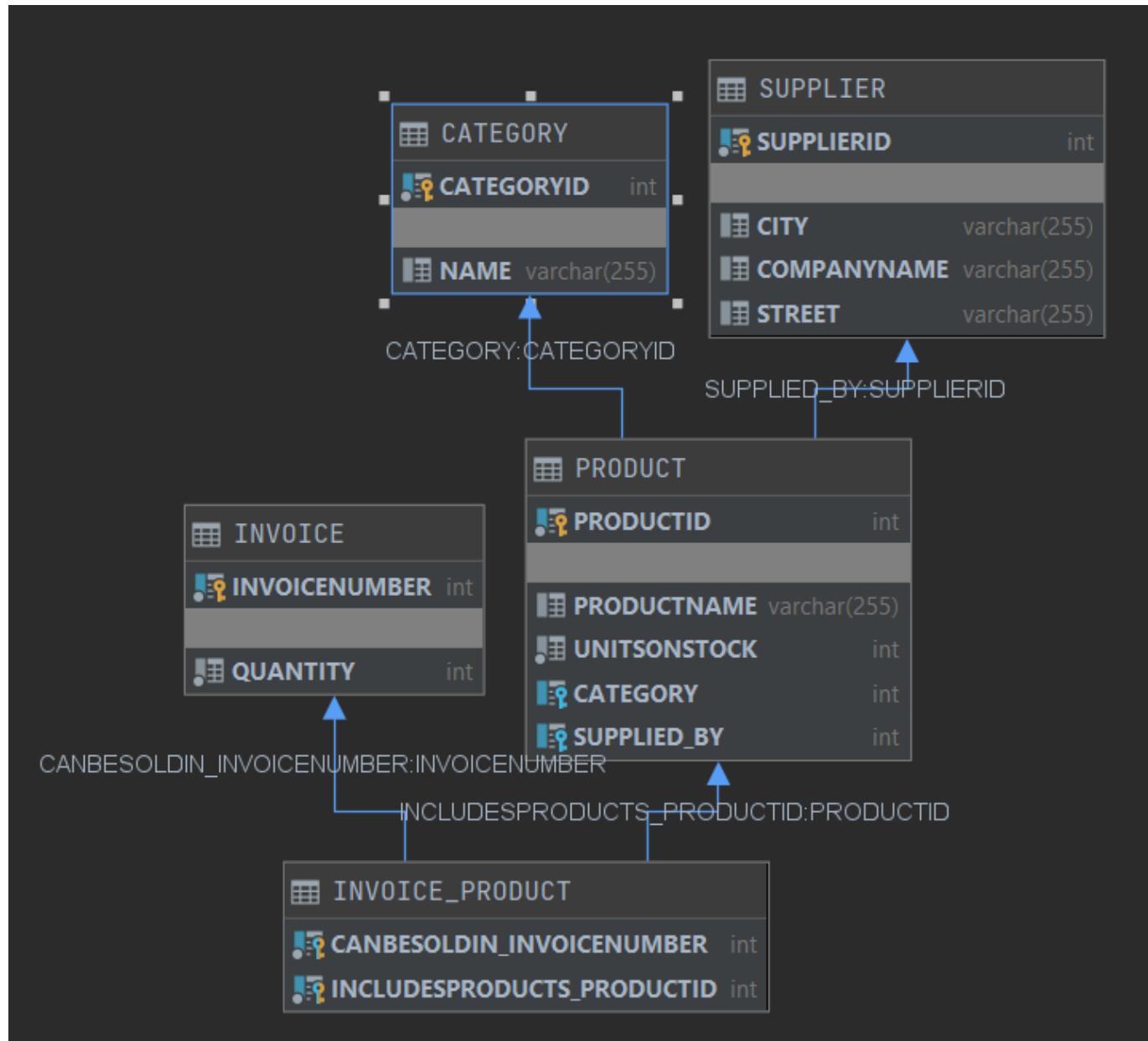


The screenshot shows a Java code editor with two tabs: "persistence.xml" and "JPAMain.java". The "JPAMain.java" tab is active, displaying the following code:

```
5
6 > public class JPAMain {
7     private static EntityManagerFactory entityManagerFactory;
8
9     private static EntityManager getEntityManager(){
10        if (entityManagerFactory == null)
11            entityManagerFactory = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
12        return entityManagerFactory.createEntityManager();
13    }
14
15 > public static void main(String argv[]){
16     EntityManager entityManager = getEntityManager();
17     EntityTransaction etx = entityManager.getTransaction();
18     etx.begin();
19
20     Supplier supplier = new Supplier( companyName: "Clothes Supplier", street: "Street 2", city: "Paris");
21     Product product1 = new Product( productName: "Jeans", unitsOnStock: 6);
22     Product product2 = new Product( productName: "Skirt", unitsOnStock: 10);
23     Product product3 = new Product( productName: "Shirt", unitsOnStock: 20);
24     supplier.addProduct(product1);
25     supplier.addProduct(product2);
26     supplier.addProduct(product3);
27
28     entityManager.persist(product1);
29     entityManager.persist(product2);
30     entityManager.persist(product3);
31     entityManager.persist(supplier);
32
33     etx.commit();
34     entityManager.close();
35 }
36 }
```

Dodano do bazy dostawcę i kilka produktów.

Schemat bazy wygląda następująco:



Skrypt generujący bazę:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000000-d7bd80dd-0171-c670-619d-fffffc71c8945"
            primary key,
    NAME      VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000001-f14580e4-0171-c670-619d-fffffc71c8945"
            primary key,
    QUANTITY     INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID   INTEGER not null
        constraint "SQL0000000004-1e2700f9-0171-c670-619d-fffffc71c8945"
            primary key,
    CITY         VARCHAR(255),
    COMPANYNAME  VARCHAR(255),
    STREET       VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID     INTEGER not null
        constraint "SQL0000000003-247a40f2-0171-c670-619d-fffffc71c8945"
            primary key,
    PRODUCTNAME   VARCHAR(255),
    UNITSONSTOCK  INTEGER not null,
    CATEGORY      INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY   INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000002-3ad9c0eb-0171-c670-619d-fffffc71c8945"
);
```

```
primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000005-97e00100-0171-c670-619d-fffffc71c8945"
on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000006-61228105-0171-c670-619d-fffffc71c8945"
on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000007-6a6b410a-0171-c670-619d-fffffc71c8945"
on PRODUCT (CATEGORY);

create index "SQL0000000008-83ba410f-0171-c670-619d-fffffc71c8945"
on PRODUCT (SUPPLIED_BY);
```

Obiekty dodane do bazy:

The screenshot shows a database management interface with two panes. The top pane contains a SQL query: `SELECT * FROM SUPPLIER`. The bottom pane displays the results in a grid format, labeled `APP.SUPPLIER`. The data shows one row with Supplier ID 1, City 4 (Paris), Company Name 'Clothes Supplier', and Street 'Street 2'.

SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Paris	Clothes Supplier

The screenshot shows a database management interface with two panes. The top pane contains a SQL query: `SELECT * FROM PRODUCT`. The bottom pane displays the results in a grid format, labeled `APP.PRODUCT`. The data shows three rows with Product IDs 1, 2, and 3, names 'Jeans', 'Skirt', and 'Shirt' respectively, unit stock counts of 6, 10, and 20, and a supplied by value of 4.

PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	1 Jeans	6	<null>	4
2	2 Skirt	10	<null>	4
3	3 Shirt	20	<null>	4

Zadanie XI. Kaskady

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

W klasie `Product` zbiór faktur wygląda następująco:

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

W klasie `Invoice` zbiór produktów:

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

W klasie `JPAMain` stworzono kilka produktów, kategorię, dostawcę i faktury i dodano wszystkie rzeczy oprócz produktów do bazy.

The screenshot shows a database interface with a query editor and a results table.

Query Editor:

```
1 ✓ | SELECT * FROM INVOICE
```

Results Table:

	INVOICENUMBER	QUANTITY
1	1	1
2	3	3

```
public static void main(String argv[]){
    EntityManager entityManager = getEntityManager();
    EntityTransaction etx = entityManager.getTransaction();
    etx.begin();

    Supplier supplier = new Supplier( companyName: "Clothes Supplier", street: "Street 2", city: "Paris");
    Category category = new Category( name: "Clothes");
    Product product1 = new Product( productName: "Jeans", unitsOnStock: 6);
    Product product2 = new Product( productName: "Skirt", unitsOnStock: 10);
    Product product3 = new Product( productName: "Shirt", unitsOnStock: 20);
    supplier.addProduct(product1);
    supplier.addProduct(product2);
    supplier.addProduct(product3);
    category.addProduct(product1);
    category.addProduct(product2);
    category.addProduct(product3);

    Invoice invoice1 = new Invoice( quantity: 0);
    Invoice invoice2 = new Invoice( quantity: 0);

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

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

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

The screenshot shows a database interface with a toolbar at the top containing various icons for operations like running, saving, and closing. Below the toolbar, a query window displays the SQL command: `SELECT * FROM INVOICE_PRODUCT`. The result set is shown in a table titled `APP.INVOICE_PRODUCT`. The table has two columns: `CANBESOLDIN_INVOICENUMBER` and `INCLUDESPRODUCTS_PRODUCTID`. The data is as follows:

	CANBESOLDIN_INVOICENUMBER	INCLUDESPRODUCTS_PRODUCTID
1	1	2
2	3	2
3	3	4
4	3	5

The screenshot shows a database interface with a query window containing the SQL command `SELECT * FROM PRODUCT`. Below the query window is a table titled `APP.PRODUCT` with four columns: `PRODUCTID`, `PRODUCTNAME`, `UNITSONSTOCK`, and `CATEGORY`. The table contains three rows of data:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY
1	2	Jeans	6	6
2	4	Skirt	10	6
3	5	Shirt	20	6

The screenshot shows a database interface with a query window containing the SQL command `SELECT * FROM SUPPLIER`. Below the query window is a table titled `APP.SUPPLIER` with four columns: `SUPPLIERID`, `CITY`, `COMPANYNAME`, and `STREET`. The table contains one row of data:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	7	Paris	Clothes Supplier	Street 2

The screenshot shows a database interface with a query window containing the SQL command `SELECT * FROM CATEGORY`. Below the query window is a table titled `APP.CATEGORY` with two columns: `CATEGORYID` and `NAME`. The table contains one row of data:

	CATEGORYID	NAME
1	6	Clothes

Wszystkie obiekty (łącznie z produktami) dodaly się prawidłowo do bazy.

Spróbowano również dodać wszystkie obiekty oprócz faktur jak poniżej.

```
public static void main(String argv[]){
    EntityManager entityManager = getEntityManager();
    EntityTransaction etx = entityManager.getTransaction();
    etx.begin();

    Supplier supplier = new Supplier( companyName: "Furniture Supplier", street: "Street 3", city: "Rome");
    Category category = new Category( name: "Furniture");
    Product product1 = new Product( productName: "Bed", unitsOnStock: 3);
    Product product2 = new Product( productName: "Chair", unitsOnStock: 15);
    Product product3 = new Product( productName: "Table", unitsOnStock: 2);
    supplier.addProduct(product1);
    supplier.addProduct(product2);
    supplier.addProduct(product3);
    category.addProduct(product1);
    category.addProduct(product2);
    category.addProduct(product3);

    Invoice invoice1 = new Invoice( quantity: 0);
    Invoice invoice2 = new Invoice( quantity: 0);

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

    entityManager.persist(product1);
    entityManager.persist(product2);
    entityManager.persist(product3);
    entityManager.persist(category);
    entityManager.persist(supplier);

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

The screenshot shows a database interface with a query editor and a results table.

Query Editor:

```
1 ✓ SELECT PRODUCTID, PRODUCTNAME, UNITSONSTOCK, CATEGORY, SUPPLIED_BY, INVOICENUMBER, QUANTITY
2   FROM PRODUCT
3   JOIN INVOICE_PRODUCT IP ON PRODUCT.PRODUCTID = IP.INCLUDESPRODUCTS_PRODUCTID
4   JOIN INVOICE I ON IP.CANBESOLDIN_INVOICENUMBER = I.INVOICENUMBER
```

Results Table:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY	INVOICENUMBER	QUANTITY
1	1	Bed	3	6	7	2	1
2	1	Bed	3	6	7	3	3
3	4	Chair	15	6	7	3	3
4	5	Table	2	6	7	3	3

Obiekty (łącznie z fakturami) również dodały się prawidłowo. Kaskadowość zatem działa.

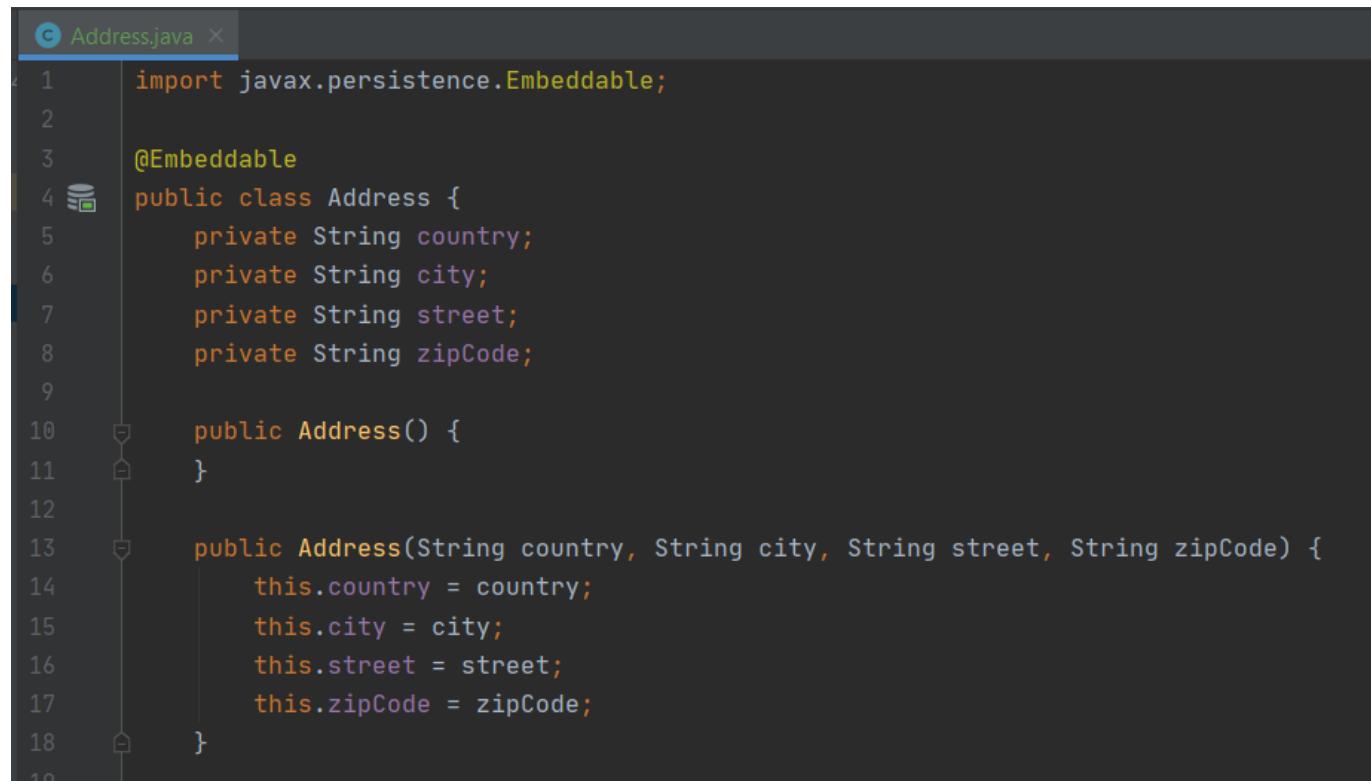
Zadanie XII. Embedded class

a), b) Modyfikacja modelu

Dodano do modelu nową klasę **Address** i "wbudowano" ją do tabeli dostawców.

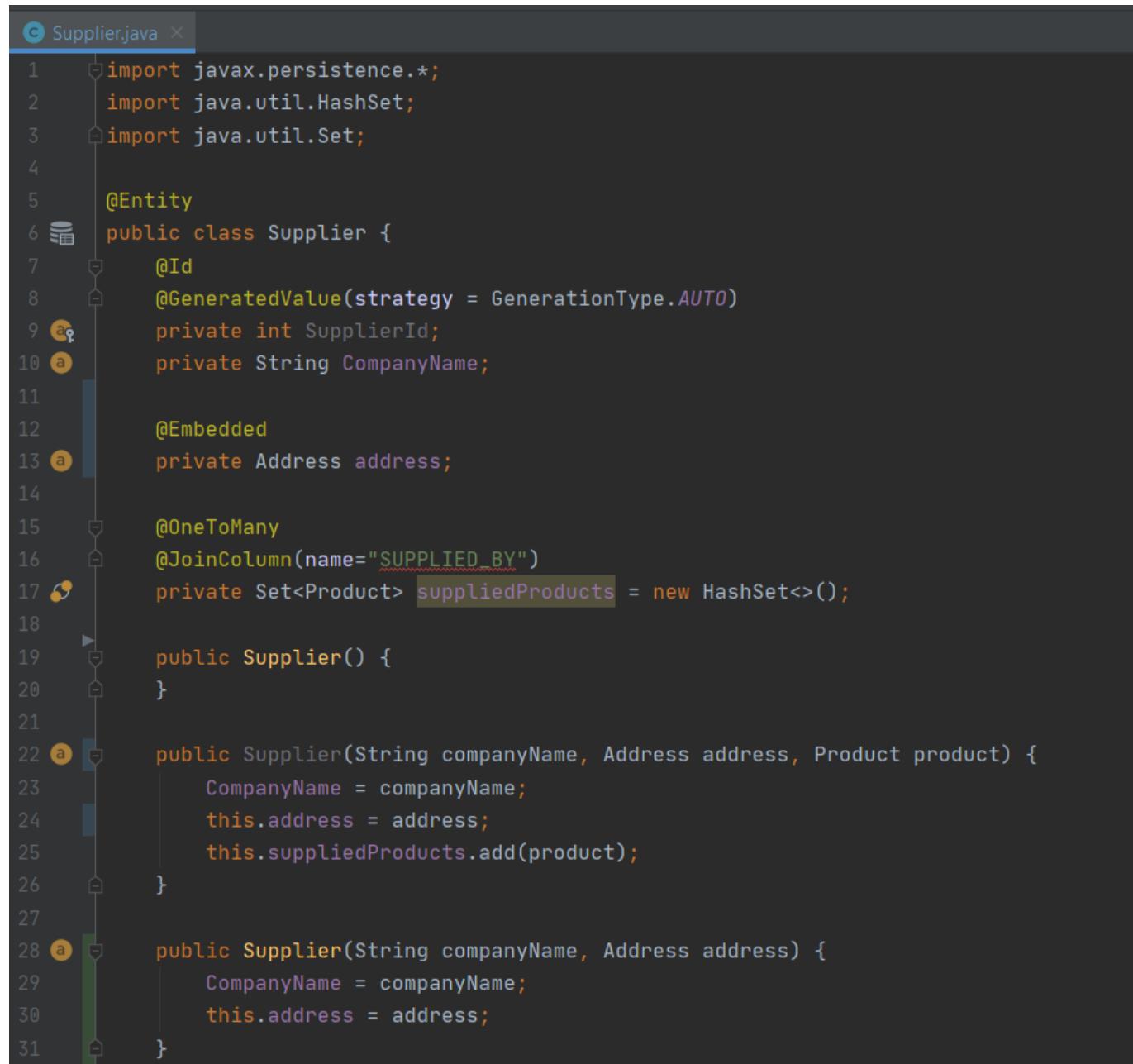
Klasa ta zawiera następujące pola:

- private String country
- private String city
- private String street
- private String zipCode



```
1 import javax.persistence.Embeddable;
2
3 @Embeddable
4 public class Address {
5     private String country;
6     private String city;
7     private String street;
8     private String zipCode;
9
10    public Address() {
11    }
12
13    public Address(String country, String city, String street, String zipCode) {
14        this.country = country;
15        this.city = city;
16        this.street = street;
17        this.zipCode = zipCode;
18    }
19}
```

W klasie `Supplier` pola `City` i `Street` zastąpiono polem `Address`:



```

1 import javax.persistence.*;
2 import java.util.HashSet;
3 import java.util.Set;
4
5 @Entity
6 public class Supplier {
7     @Id
8     @GeneratedValue(strategy = GenerationType.AUTO)
9     private int SupplierId;
10    private String CompanyName;
11
12    @Embedded
13    private Address address;
14
15    @OneToMany
16    @JoinColumn(name = "SUPPLIED_BY")
17    private Set<Product> suppliedProducts = new HashSet<>();
18
19    public Supplier() {
20    }
21
22    public Supplier(String companyName, Address address, Product product) {
23        CompanyName = companyName;
24        this.address = address;
25        this.suppliedProducts.add(product);
26    }
27
28    public Supplier(String companyName, Address address) {
29        CompanyName = companyName;
30        this.address = address;
31    }

```

Dodano do bazy kilku dostawców:

```

etx.begin();

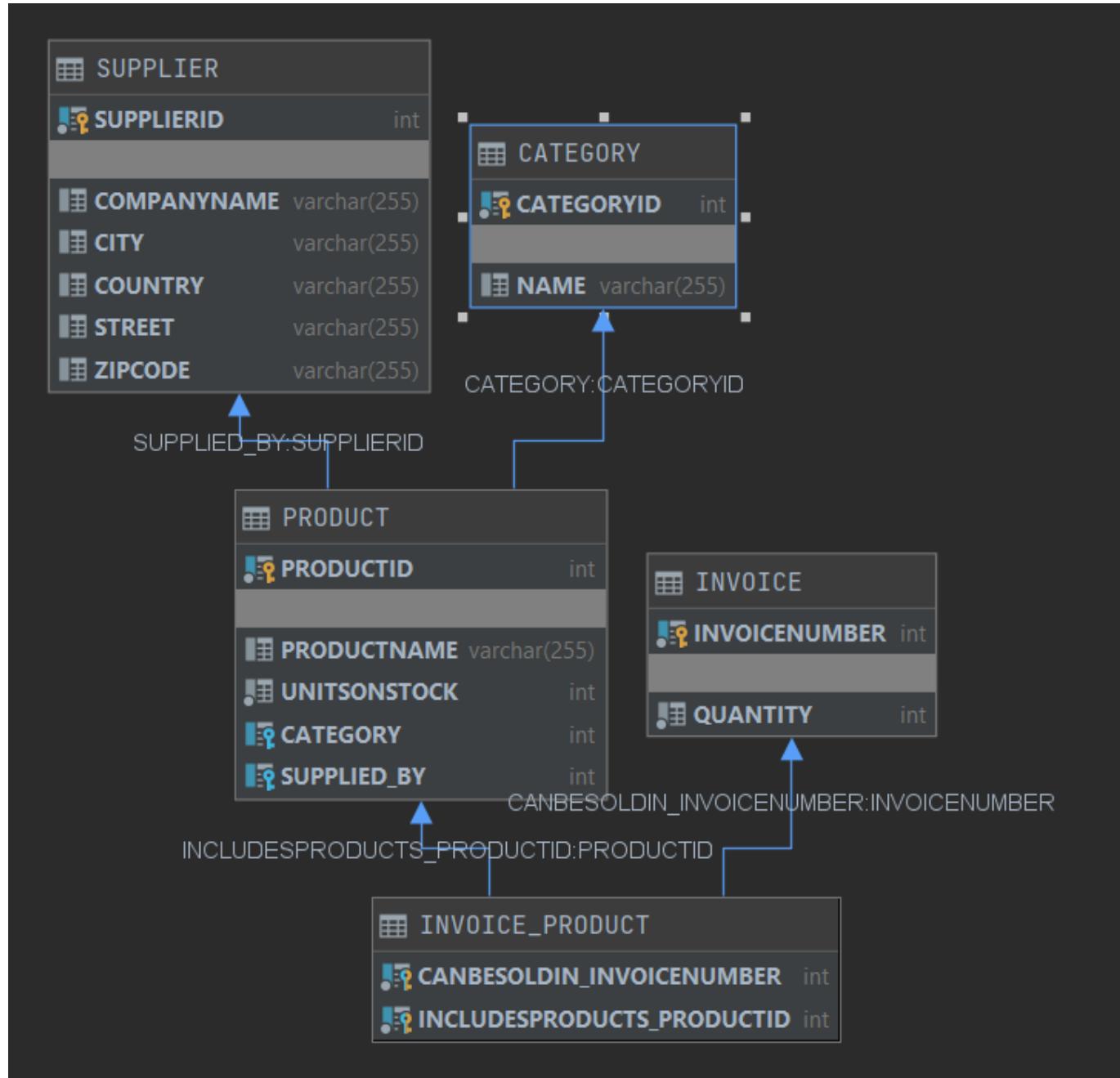
Address address1 = new Address( country: "England", city: "London", street: "Downing Street", zipCode: "33-330");
Address address2 = new Address( country: "Italy", city: "Rome", street: "Street1", zipCode: "33-395");
Supplier supplier1 = new Supplier( companyName: "Supplier1", address1);
Supplier supplier2 = new Supplier( companyName: "Supplier2", address2);
Supplier supplier3 = new Supplier( companyName: "Supplier3", address1);

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

etx.commit();
entityManager.close();

```

Schemat bazy danych wygląda następująco:



Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000054-5abe851c-0171-c670-619d-fffffc71c8945"
            primary key,
    NAME      VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000055-d26c4522-0171-c670-619d-fffffc71c8945"
            primary key,
    QUANTITY    INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000058-a2f88535-0171-c670-619d-fffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255),
    CITY        VARCHAR(255),
    COUNTRY     VARCHAR(255),
    STREET       VARCHAR(255),
    ZIPCODE     VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000057-01e2c52e-0171-c670-619d-fffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY   INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
```

```
    references PRODUCT (PRODUCTID),
constraint "SQL0000000056-8a230528-0171-c670-619d-fffffc71c8945"
    primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000059-8acbc53b-0171-c670-619d-fffffc71c8945"
on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000060-d008453f-0171-c670-619d-fffffc71c8945"
on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000061-1548c543-0171-c670-619d-fffffc71c8945"
on PRODUCT (CATEGORY);

create index "SQL0000000062-5a8d4547-0171-c670-619d-fffffc71c8945"
on PRODUCT (SUPPLIED_BY);
```

Dane dodały się poprawnie:

The screenshot shows a database interface with two main sections. The top section is a query editor with the following content:

```
1 ✓ | SELECT * FROM SUPPLIER
```

The bottom section is a results grid titled "APP.SUPPLIER" showing the following data:

	SUPPLIERID	COMPANYNAME	CITY	COUNTRY	STREET	ZIPCODE
1	1	Supplier1	London	England	Downing Street	33-330
2	2	Supplier2	Rome	Italy	Street1	33-395
3	3	Supplier3	London	England	Downing Street	33-330

c), d) Modyfikacja modelu

Zmodyfikowano model w taki sposób, aby dane adresowe znajdowały się w klasie dostawców. Zmapowano to do dwóch osobnych tabel.

Klasa `Supplier`:

```
@Entity
@SecondaryTable(name="ADDRESS")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierId;
    private 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")
    private Set<Product> suppliedProducts = new HashSet<>();
}
```

Dodano kilku dostawców:

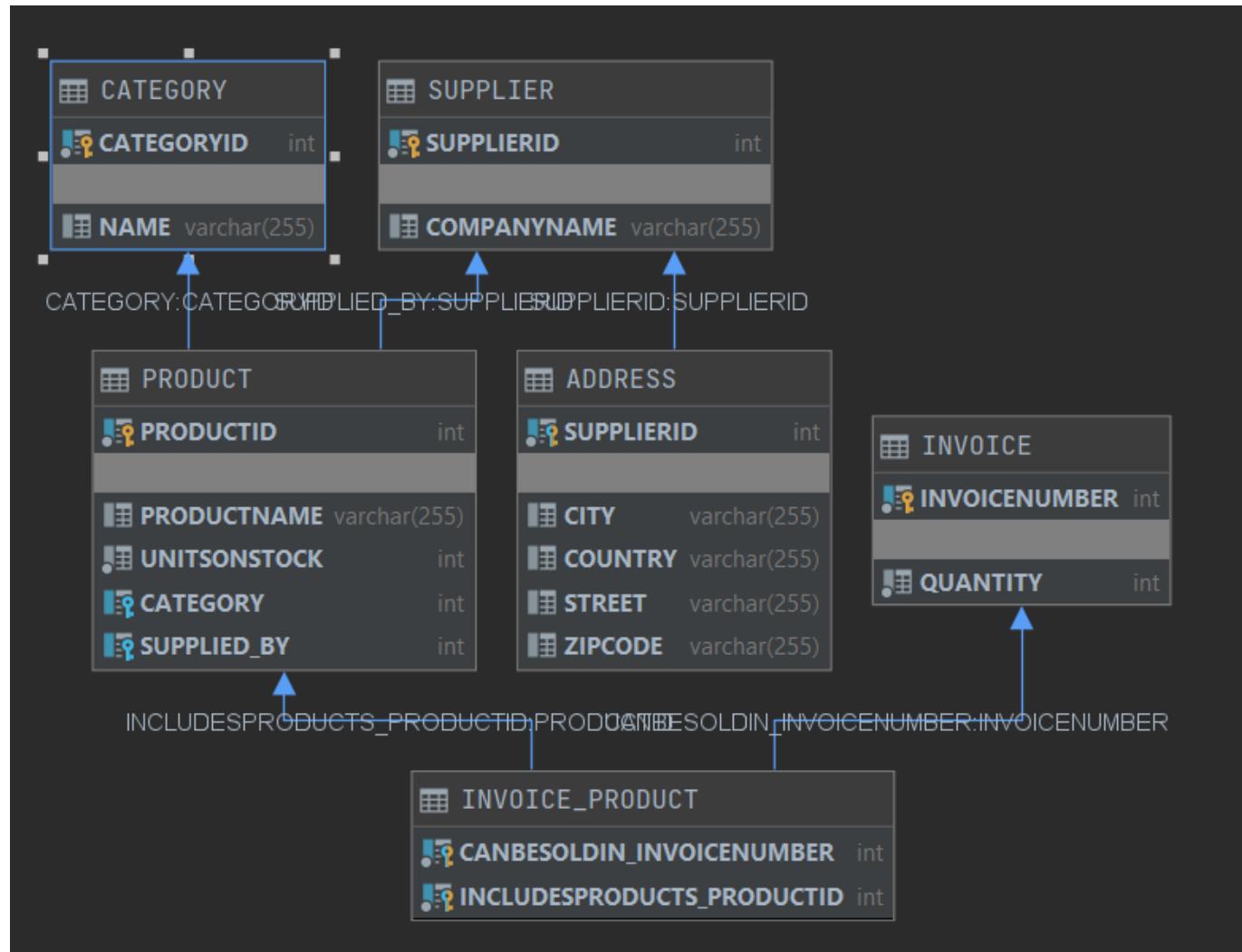
```
etx.begin();

Supplier supplier1 = new Supplier( companyName: "Supplier1", country: "England", city: "London", street: "Downing Street", zipCode: "33-330");
Supplier supplier2 = new Supplier( companyName: "Supplier2", country: "Italy", city: "Rome", street: "Street1", zipCode: "33-395");
Supplier supplier3 = new Supplier( companyName: "Supplier3", country: "Poland", city: "Cracow", street: "Kawiory", zipCode: "33-300");

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

etx.commit();
entityManager.close();
```

Schemat bazy danych:



Skrypt generujący bazę:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000075-791dc68b-0171-c670-619d-fffffc71c8945"
            primary key,
    NAME      VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000076-e2f20691-0171-c670-619d-fffffc71c8945"
            primary key,
    QUANTITY    INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000079-20a4c6a3-0171-c670-619d-fffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255)
);

create table ADDRESS
(
    CITY      VARCHAR(255),
    COUNTRY   VARCHAR(255),
    STREET    VARCHAR(255),
    ZIPCODE   VARCHAR(255),
    SUPPLIERID INTEGER not null
        constraint "SQL0000000074-8f528685-0171-c670-619d-fffffc71c8945"
            primary key
        constraint FKFIXSMY956R97J539UD6866JN5
            references SUPPLIER (SUPPLIERID)
);

create index "SQL0000000080-8a9d06a9-0171-c670-619d-fffffc71c8945"
    on ADDRESS (SUPPLIERID);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000078-36b5869d-0171-c670-619d-fffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
```

```

SUPPLIED_BY  INTEGER
    constraint FKRKVCU2QJUUMNU5IHG9CWRBTN
        references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID  INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000077-4ccf4697-0171-c670-619d-fffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000081-3af2c6ae-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000082-a1a246b2-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000083-0855c6b6-0171-c670-619d-fffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000084-6f0d46ba-0171-c670-619d-fffffc71c8945"
    on PRODUCT (SUPPLIED_BY);

```

Obiekty dodane do bazy:

The screenshot shows the Oracle SQL Developer interface. At the top, there's a toolbar with various icons. Below it, the SQL editor window displays the following command:

```
1 ✓ | SELECT * FROM SUPPLIER
```

The output pane below shows the results of the query:

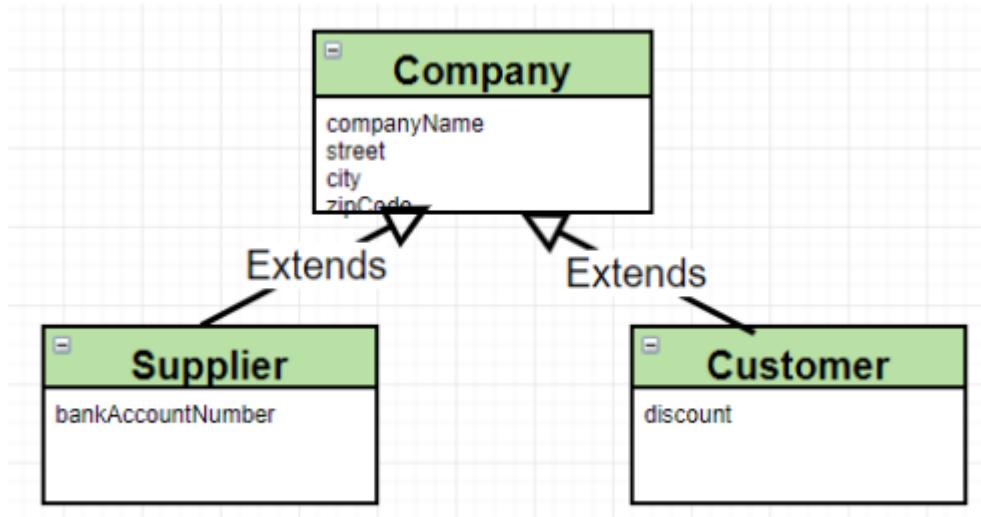
	SUPPLIERID	COMPANYNAME
1	1	Supplier1
2	2	Supplier2
3	3	Supplier3

The screenshot shows a database interface with a toolbar at the top containing various icons for navigation and operations. Below the toolbar, a query window displays the SQL command: `SELECT * FROM ADDRESS`. The result set is shown in a table titled "APP.ADDRESS". The table has five columns: CITY, COUNTRY, STREET, ZIPCODE, and SUPPLIERID. There are three rows of data:

	CITY	COUNTRY	STREET	ZIPCODE	SUPPLIERID
1	London	England	Downing Street	33-330	1
2	Rome	Italy	Street1	33-395	2
3	Cracow	Poland	Kawiory	33-300	3

Zadanie XIII. Dziedziczenie

Wprowadzono do modelu następującą hierarchię:



a) SINGLE TABLE

Klasa **Company**:

```

@Entity
@SecondaryTable(name="ADDRESS")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyId;
    private 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;
        this.country = country;
        this.city = city;
        this.street = street;
        this.zipCode = zipCode;
    }
}
  
```

Klasa Supplier:

```
@Entity
public class Supplier extends Company{
    private String bankAccountNumber;

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

    public Supplier() {
    }

    public Supplier(String bankAccountNumber) {
        this.bankAccountNumber = bankAccountNumber;
    }

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

Klasa Customer:

```
@Entity
public class Customer extends Company{
    private double discount;

    public Customer(){}
    public Customer(double discount) { this.discount = discount; }

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

    public double getDiscount() { return discount; }

    public void setDiscount(double discount) { this.discount = discount; }
}
```

Dodanie obiektów do bazy:

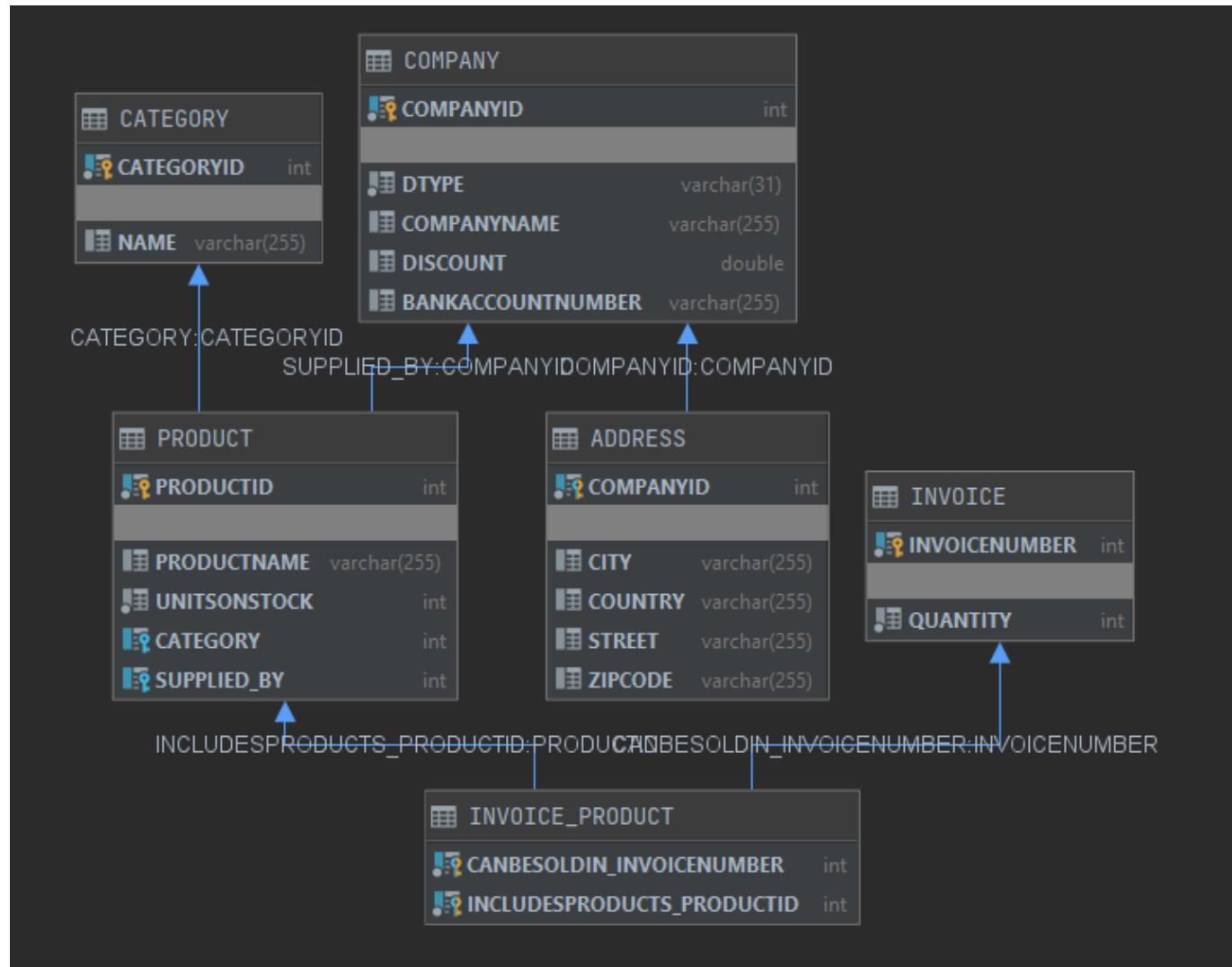
```
emtx.begin();

Supplier supplier1 = new Supplier( companyName: "Supplier1", country: "England", city: "London", street: "Downing Street", zipCode: "33-330", bankAccountNumber: "123456789");
Supplier supplier2 = new Supplier( companyName: "Supplier2", country: "Italy", city: "Rome", street: "Street1", zipCode: "33-395", bankAccountNumber: "987654321");
Customer customer1 = new Customer( companyName: "Customer1", country: "Poland", city: "Cracow", street: "Kawiory", zipCode: "33-300", discount: 0.5);
Customer customer2 = new Customer( companyName: "Customer2", country: "Poland", city: "Warsaw", street: "Street2", zipCode: "33-360", discount: 0.2);

entityManager.persist(supplier1);
entityManager.persist(supplier2);
entityManager.persist(customer1);
entityManager.persist(customer2);

emtx.commit();
entityManager.close();
```

Schemat bazy danych:



Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000108-0f9fc95b-0171-c670-619d-fffffc71c8945"
            primary key,
    NAME      VARCHAR(255)
);

create table COMPANY
(
    DTTYPE          VARCHAR(31) not null,
    COMPANYID       INTEGER      not null
        constraint "SQL0000000109-7dac0961-0171-c670-619d-fffffc71c8945"
            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
        constraint "SQL0000000107-219c8955-0171-c670-619d-fffffc71c8945"
            primary key
        constraint FKG1JYTBE0JJH2N1XV4JDOHNOQ2
            references COMPANY (COMPANYID)
);

create index "SQL0000000113-36370979-0171-c670-619d-fffffc71c8945"
on ADDRESS (COMPANYID);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000110-ebc14967-0171-c670-619d-fffffc71c8945"
            primary key,
    QUANTITY       INTEGER not null
);

create table PRODUCT
(
    PRODUCTID      INTEGER not null
        constraint "SQL0000000112-c806c973-0171-c670-619d-fffffc71c8945"
            primary key,
    PRODUCTNAME    VARCHAR(255),
    UNITSONSTOCK  INTEGER not null,
);
```

```

CATEGORY      INTEGER
    constraint FKEDNTEOGHHMGHKELRH3HBU75LP
        references CATEGORY (CATEGORYID),
SUPPLIED_BY   INTEGER
    constraint FKORWD7TI86M40HR0CRO5GFGLHG
        references COMPANY (COMPANYID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
constraint "SQL0000000111-d9df896d-0171-c670-619d-fffffc71c8945"
    primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000114-6a10c97e-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000115-d3904982-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000116-3d13c986-0171-c670-619d-fffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000117-a69b498a-0171-c670-619d-fffffc71c8945"
    on PRODUCT (SUPPLIED_BY);

```

Obiekty w bazie danych:

The screenshot shows a database interface with a toolbar at the top and a results grid below. The toolbar includes icons for execution, transaction status, and connection information. The results grid displays the following data:

	DTYPE	COMPANYID	COMPANYNAME	DISCOUNT	BANKACCOUNTNUMBER
1	Supplier	1	Supplier1	<null>	123456789
2	Supplier	2	Supplier2	<null>	987654321
3	Customer	3	Customer1	0.5	<null>
4	Customer	4	Customer2	0.2	<null>

The screenshot shows a database interface with a toolbar at the top and a main window below. In the main window, there is a query editor with the SQL command `SELECT * FROM ADDRESS`. Below the query editor is a table titled "APP.ADDRESS". The table has columns: CITY, COUNTRY, STREET, ZIPCODE, and COMPANYID. There are four rows of data:

	CITY	COUNTRY	STREET	ZIPCODE	COMPANYID
1	London	England	Downing Street	33-330	1
2	Rome	Italy	Street1	33-395	2
3	Cracow	Poland	Kawiory	33-300	3
4	Warsaw	Poland	Street2	33-360	4

b) JOINED

Klasa **Company**:

```

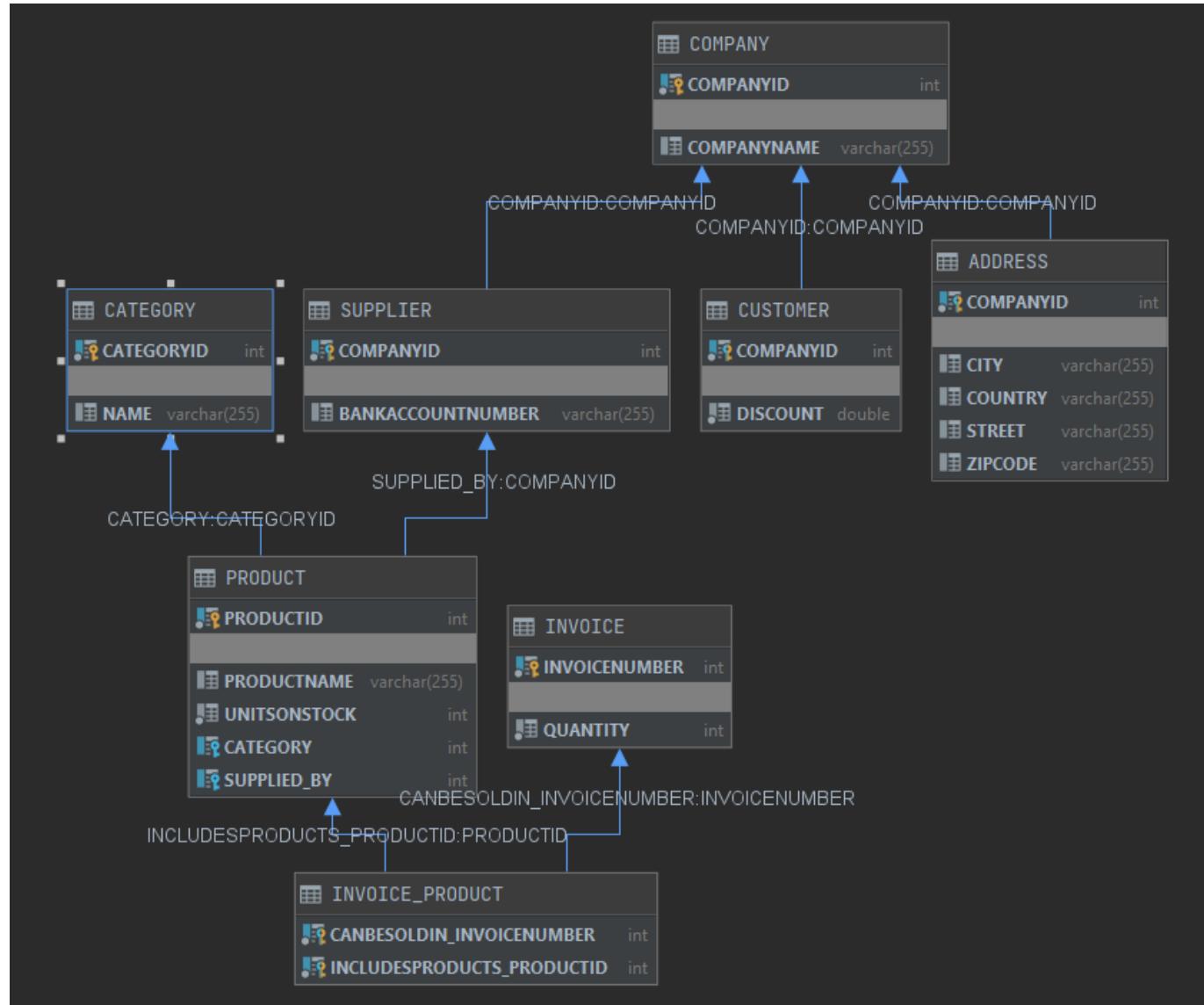
@Entity
@SecondaryTable(name="ADDRESS")
@Inheritance(strategy = InheritanceType.JOINED)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyId;
    private String CompanyName;

    @Column(table="ADDRESS")
    private String country;
    @Column(table="ADDRESS")
    private String city;
    @Column(table="ADDRESS")
    private String street;
    @Column(table="ADDRESS")
    private String zipCode;
}

```

Klasy **Supplier** i **Customer** pozostały bez zmian. Do bazy dodano te same obiekty, co poprzednio.

Schemat bazy danych:



Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000134-2bff0bb9-0171-c670-619d-fffffc71c8945"
            primary key,
    NAME      VARCHAR(255)
);

create table COMPANY
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000120-f3748a15-0171-c670-619d-fffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255)
);

create table ADDRESS
(
    CITY      VARCHAR(255),
    COUNTRY   VARCHAR(255),
    STREET    VARCHAR(255),
    ZIPCODE   VARCHAR(255),
    COMPANYID INTEGER not null
        constraint "SQL0000000133-ba6ecbb3-0171-c670-619d-fffffc71c8945"
            primary key
        constraint FKG1JYTBE0JJH2N1XV4JDOHNOQ2
            references COMPANY (COMPANYID)
);

create index "SQL0000000140-21fc0be1-0171-c670-619d-fffffc71c8945"
on ADDRESS (COMPANYID);

create table CUSTOMER
(
    DISCOUNT  DOUBLE not null,
    COMPANYID INTEGER not null
        constraint "SQL0000000135-e958cbc3-0171-c670-619d-fffffc71c8945"
            primary key
        constraint FKRR2PXJ29E8FMTV5S31R8FKX7K
            references COMPANY (COMPANYID)
);

create index "SQL0000000141-98d7cbe6-0171-c670-619d-fffffc71c8945"
on CUSTOMER (COMPANYID);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000136-5b010bc9-0171-c670-619d-fffffc71c8945"
            primary key,
```

```
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    BANKACCOUNTNUMBER VARCHAR(255),
    COMPANYID        INTEGER not null
        constraint "SQL0000000139-b02fcbdb-0171-c670-619d-fffffc71c8945"
        primary key
    constraint FKNR3T1U2JVA8PL6UPKPAXXABV
        references COMPANY (COMPANYID)
);

create table PRODUCT
(
    PRODUCTID      INTEGER not null
        constraint "SQL0000000138-be6c8bd5-0171-c670-619d-fffffc71c8945"
        primary key,
    PRODUCTNAME    VARCHAR(255),
    UNITSONSTOCK  INTEGER not null,
    CATEGORY       INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
        references CATEGORY (CATEGORYID),
    SUPPLIED_BY    INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
        references SUPPLIER (COMPANYID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
        references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
        references PRODUCT (PRODUCTID),
    constraint "SQL0000000137-ccb24bcf-0171-c670-619d-fffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000142-dfb9cbeb-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);
create index "SQL0000000143-2ba64bef-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);
create index "SQL0000000144-7796cbf3-0171-c670-619d-fffffc71c8945"
    on PRODUCT (CATEGORY);
create index "SQL0000000145-c38b4bf7-0171-c670-619d-fffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
create index "SQL0000000146-0f83cbfb-0171-c670-619d-fffffc71c8945"
    on SUPPLIER (COMPANYID);
```

Obiekty w bazie danych:

The screenshot shows a database interface with a toolbar at the top. A green checkmark icon and the text "SELECT * FROM COMPANY" are visible in the query editor. Below the editor is a table titled "APP.COMPANY". The table has two columns: "COMPANYID" and "COMPANYNAME". The data consists of four rows:

	COMPANYID	COMPANYNAME
1	1	Supplier1
2	2	Supplier2
3	3	Customer1
4	4	Customer2

The screenshot shows a database interface with a toolbar at the top. A green checkmark icon and the text "SELECT * FROM ADDRESS" are visible in the query editor. Below the editor is a table titled "APP.ADDRESS". The table has six columns: CITY, COUNTRY, STREET, ZIPCODE, COMPANYID, and COMPANYNAME. The data consists of four rows:

	CITY	COUNTRY	STREET	ZIPCODE	COMPANYID
1	London	England	Downing Street	33-330	1
2	Rome	Italy	Street1	33-395	2
3	Cracow	Poland	Kawiory	33-300	3
4	Warsaw	Poland	Street2	33-360	4

The screenshot shows a database interface with a toolbar at the top containing icons for running, saving, and preferences, along with dropdowns for transaction mode (Tx: Auto) and a checkmark button. Below the toolbar, a status bar indicates '1 ✓' and the query 'SELECT * FROM SUPPLIER'. The main area displays the results of the query in a table titled 'APP.SUPPLIER'. The table has two columns: 'BANKACCOUNTNUMBER' and 'COMPANYID'. The data shows two rows: row 1 with BANKACCOUNTNUMBER '123456789' and COMPANYID '1'; and row 2 with BANKACCOUNTNUMBER '987654321' and COMPANYID '2'. Navigation buttons for rows and columns are visible above the table.

	BANKACCOUNTNUMBER	COMPANYID
1	123456789	1
2	987654321	2

The screenshot shows a database interface with a toolbar at the top containing icons for running, saving, and preferences, along with dropdowns for transaction mode (Tx: Auto) and a checkmark button. Below the toolbar, a status bar indicates '1 ✓' and the query 'SELECT * FROM CUSTOMER'. The main area displays the results of the query in a table titled 'APP.CUSTOMER'. The table has two columns: 'DISCOUNT' and 'COMPANYID'. The data shows two rows: row 1 with DISCOUNT '0.5' and COMPANYID '3'; and row 2 with DISCOUNT '0.2' and COMPANYID '4'. Navigation buttons for rows and columns are visible above the table.

	DISCOUNT	COMPANYID
1	0.5	3
2	0.2	4

c) TABLE PER CLASS

Klasa **Company**:

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

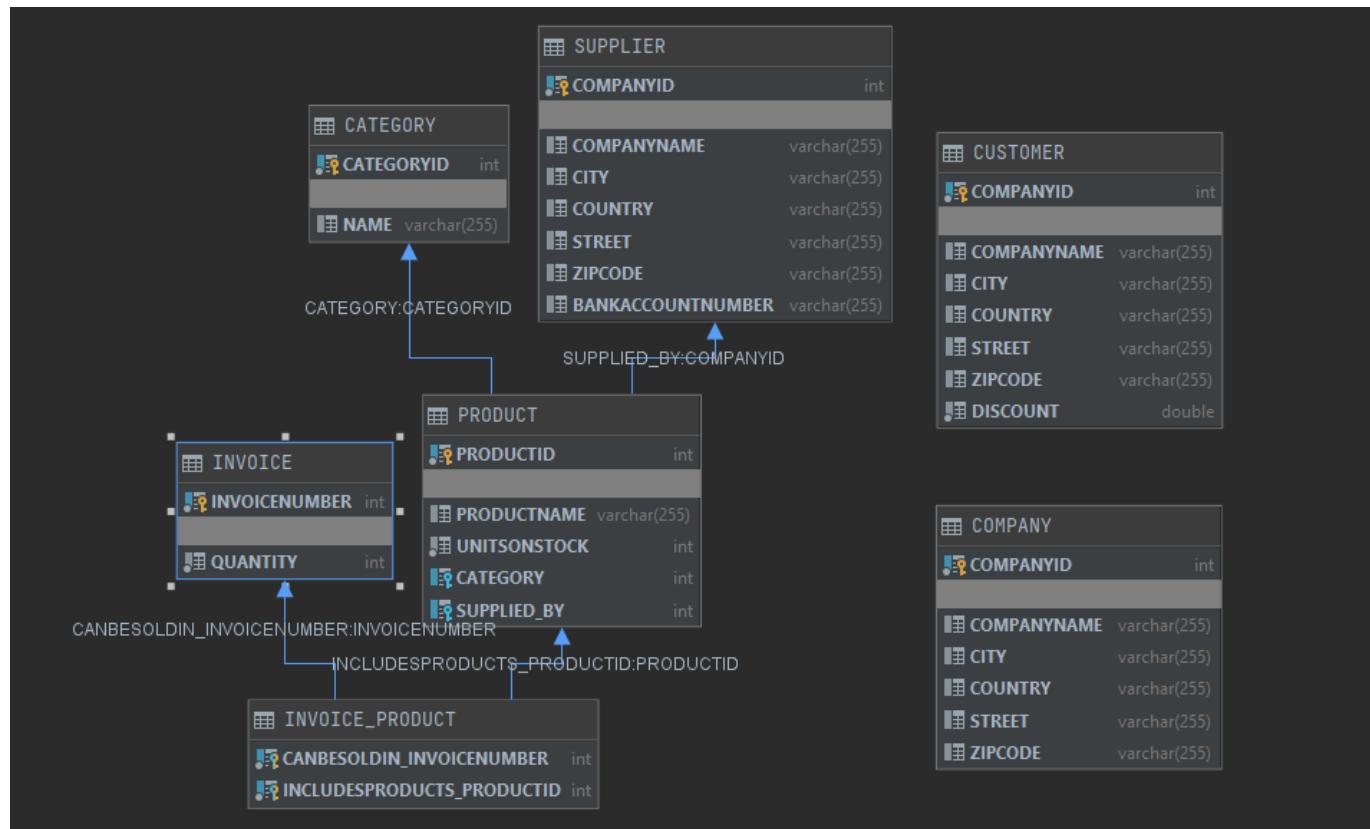
    private String country;
    private String city;
    private String street;
    private String zipCode;

    public Company() {}

    public Company(String companyName, String country, String city, String street, String zipCode) {
        CompanyName = companyName;
        this.country = country;
        this.city = city;
        this.street = street;
        this.zipCode = zipCode;
    }
}
```

Klasy **Supplier** i **Customer** pozostały bez zmian. Do bazy dodano te same obiekty, co poprzednio.

Schemat bazy danych:



Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID  INTEGER not null
        constraint "SQL0000000193-d4429205-0171-c670-619d-fffffc71c8945"
            primary key,
    NAME        VARCHAR(255)
);

create table COMPANY
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000194-cf4dd20b-0171-c670-619d-fffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255),
    CITY         VARCHAR(255),
    COUNTRY      VARCHAR(255),
    STREET        VARCHAR(255),
    ZIPCODE      VARCHAR(255)
);

create table CUSTOMER
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000195-4a621211-0171-c670-619d-fffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255),
    CITY         VARCHAR(255),
    COUNTRY      VARCHAR(255),
    STREET        VARCHAR(255),
    ZIPCODE      VARCHAR(255),
    DISCOUNT     DOUBLE  not null
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000196-c57f5217-0171-c670-619d-fffffc71c8945"
            primary key,
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    COMPANYID      INTEGER not null
        constraint "SQL0000000199-370d1229-0171-c670-619d-fffffc71c8945"
            primary key,
    COMPANYNAME    VARCHAR(255),
    CITY           VARCHAR(255),
    COUNTRY         VARCHAR(255),
    STREET          VARCHAR(255),
    ZIPCODE        VARCHAR(255),
```

```

        BANKACCOUNTNUMBER VARCHAR(255)
    );

create table PRODUCT
(
    PRODUCTID      INTEGER not null
        constraint "SQL0000000198-bbd4d223-0171-c670-619d-fffffc71c8945"
            primary key,
    PRODUCTNAME    VARCHAR(255),
    UNITSONSTOCK  INTEGER not null,
    CATEGORY       INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY    INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (COMPANYID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000197-c0a5921d-0171-c670-619d-fffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL000000200-b24e522f-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

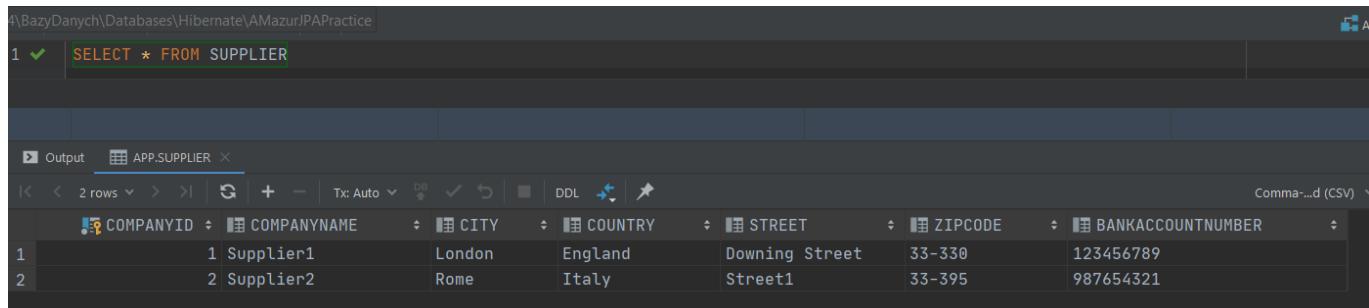
create index "SQL000000201-047ed233-0171-c670-619d-fffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL000000202-56b35237-0171-c670-619d-fffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL000000203-a8ebd23b-0171-c670-619d-fffffc71c8945"
    on PRODUCT (SUPPLIED_BY);

```

Obiekty w bazie danych:



The screenshot shows a database interface with a query window containing the following SQL statement:

```
1 ✓ | SELECT * FROM SUPPLIER
```

The results are displayed in a table titled "APP.SUPPLIER". The table has the following columns: COMPANYID, COMPANYNAME, CITY, COUNTRY, STREET, ZIPCODE, and BANKACCOUNTNUMBER. There are two rows of data:

	COMPANYID	COMPANYNAME	CITY	COUNTRY	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	1	Supplier1	London	England	Downing Street	33-330	123456789
2	2	Supplier2	Rome	Italy	Street1	33-395	987654321

The screenshot shows a database interface with a toolbar at the top. In the main area, a query window displays the SQL command `SELECT * FROM CUSTOMER`. Below it, an output window titled `APP.CUSTOMER` shows the results of the query:

	COMPANYID	COMPANYNAME	CITY	COUNTRY	STREET	ZIPCODE	DISCOUNT
1	3	Customer1	Cracow	Poland	Kawiory	33-300	0.5
2	4	Customer2	Warsaw	Poland	Street2	33-360	0.2

The screenshot shows a database interface with a toolbar at the top. In the main area, a query window displays the SQL command `SELECT * FROM COMPANY`. Below it, an output window titled `APP.COMPANY` shows the results of the query:

	COMPANYID	COMPANYNAME	CITY	COUNTRY	STREET	ZIPCODE