

README.MD

Hibernate

Author: Uladzislau Tumilovich

Sprawozdanie

Zadanie II. Basics

- a) Został uruchomiony serwer Derby. W konsoli ij zostało wywołano polecenie `connect 'jdbc:derby://127.0.0.1/UTumilovichJPA;create=true'`; oraz `show tables;`

TABLE_SCHEMA	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	SYSEQUENCES	
SYS	SYSSTATEMENTS	
SYS	SYSSTATISTICS	
SYS	SYSTABLEPERMS	
SYS	SYSTABLES	
SYS	SYSTRIGGERS	
SYS	SYSUSERS	
SYS	SYSVIEWS	
SYSIBM	SYSDUMMY1	

23 rows selected
ij>

- b) Został stworzony projekt Javowy o nazwie UTumilovichJPAPractice w IntelliJ

- c) Została stworzona klasa Product z polami `ProductName` oraz `UnitsOnStock`. Dodatkowo zostały dołączone potrzebujące Jar-ki dla uruchomienia projektu.

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int productId;
    public String productName;
    public int unitsOnStock;

    public Product() {
    }

    public Product(String productName, int unitsOnStock) {
        this.productName = productName;
        this.unitsOnStock = unitsOnStock;
    }
}
```

- d) Zostały uzupełnione potrzebne property w konfiguracji hibernate'a

```
<?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/UTumilovichJPA</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>
        <!-- DB schema will be updated if needed -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
    </session-factory>
</hibernate-configuration>
```

Zadanie III

- a) Został stworzony i dodany do bazy przykładowy produkt w metodzie *main* klasy **Main**

```
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();
    }

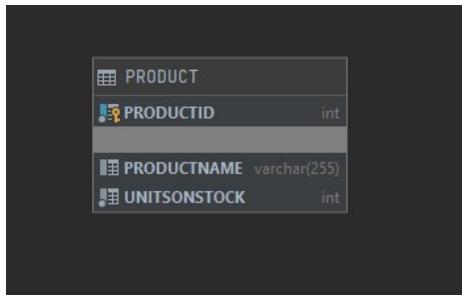
    public static void main(final String[] args) throws Exception {
        final Session session = getSession();

        Transaction transaction = session.beginTransaction();
        session.save(new Product("Laptop", 100));
        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("from " + entityName);
                System.out.println("executing: " + query.getQueryString());
                for (Object o : query.list()) {
                    System.out.println(" " + o);
                }
            }
        } finally {
            session.close();
        }
    }
}
```

- b) Rezultaty działania programu:

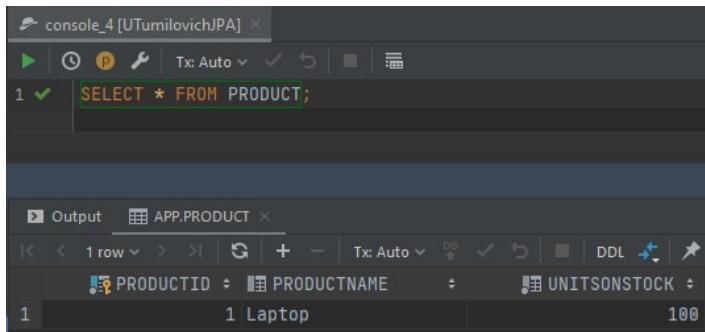
- Schemat bazy danych



- Definicja tabeli *Product* w bazie

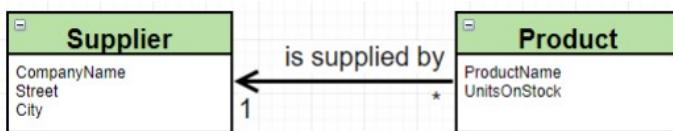
```
-- auto-generated definition
CREATE TABLE PRODUCT
(
    PRODUCTID      INTEGER NOT NULL
        PRIMARY KEY,
    PRODUCTNAME   VARCHAR(255),
    UNITSONSTOCK INTEGER NOT NULL
);
```

- Select z tabeli *Product*



Zadanie IV

Został zmodyfikowany model wprowadzeniem pojęcia Dostawcy ja poniżej:



- a) Najpierw została stworzona klasa **Supplier** z polami *supplierId*, *companyName*, *street*, *city*

```
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierId;
    private String companyName;
    private String street;
    private String city;

    public Supplier() {}

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

b) Dalej została zmodyfikowana klasa **Product** dodaniem nowego pola *supplier* oraz dodatkowymi konstruktorami i metodą *setSupplier*

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsOnStock;

    @ManyToOne
    private Supplier supplier;

    public Product() {
    }

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

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

c) Następnie do pliku konfiguracyjnego został dodany atrybut *mapping* po klasie **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.url">jdbc:derby://127.0.0.1/UTumilovichJPA</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>
        <!-- DB schema will be updated if needed -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
    </session-factory>
</hibernate-configuration>
```

d) Na koniec została zmodyfikowana metoda *main* klasy **Main** na dodanie nowego dostawcy oraz połączenie pola *supplier* istniejącego produktu z dodanym dostawcą

```
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();

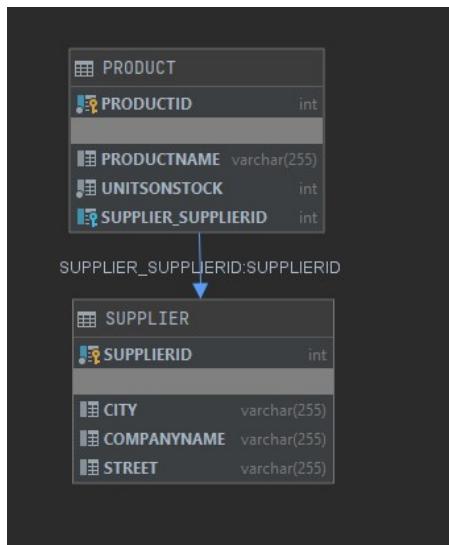
    Transaction transaction = session.beginTransaction();
    Product product = session.get(Product.class, 1);
    Supplier supplier = new Supplier("Supplier", "Somewhere", "Anywhere");
    product.setSupplier(supplier);
    session.save(supplier);
    session.save(product);
    transaction.commit();

    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("from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println(" " + o);
            }
        }
    } finally {
        session.close();
    }
}
}

```

e) Rezultaty działania programu:

- Schemat bazy danych



- Definicja tabeli *Supplier* w bazie

```

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

```

- Select z tabeli *Product*

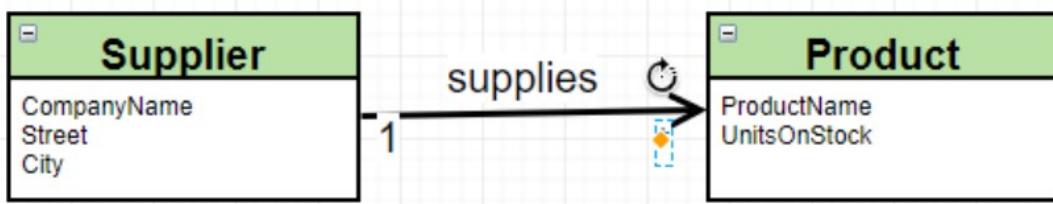
APP.PRODUCT			
PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	1 Laptop	100	2

- Select z tabeli *Supplier*

APP.SUPPLIER			
SUPPLIERID	CITY	COMPANYNAME	STREET
1	2 Anywhere	Supplier	Somewhere

Zadanie V

Relacja została odwrócona zgodnie z poniższym schematem



Z tabelą łącznikową

- a) Pole *supplier* oraz wszystkie konstruktory i metody związane z tym polem zostały usunięte z klasy **Product**

```

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsOnStock;

    public Product() {}

    public Product(String productName, int unitsOnStock) {
        this.productName = productName;
        this.unitsOnStock = unitsOnStock;
    }
}
  
```

- b) Dalej w klasie **Supplier** dodano pole *Set products*, dodatkowy konstruktor oraz metoda *addProduct*

```

@Entity
public class Supplier {
    @Id
    ...
    Set<Product> products;
    ...
    public Supplier() {}

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

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

```

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

@OneToMany
private final Set<Product> products = new HashSet<>();

public Supplier() {
}

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

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

```

c) Na koniec została zmieniona metoda *main* klasy **Main**

```

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) {
        final Session session = getSession();

        Transaction transaction = session.beginTransaction();

        Product product1 = new Product("Filter", 23);
        Product product2 = new Product("Papier", 34);
        Product product3 = new Product("Kubek", 65);

        Supplier supplier = new Supplier("Supplier", "Somewhere", "Anywhere");

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

        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(supplier);

        transaction.commit();

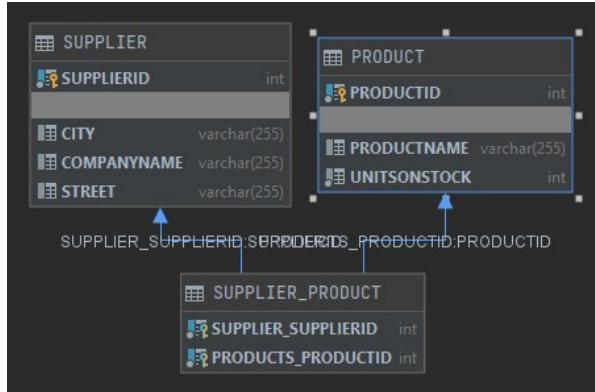
        try {
            System.out.println("querying all the managed entities...");
            final Metamodel metamodel = session.getSessionFactory().getMetamodel();
            for (EntityType<?> entityType : metamodel.getEntities()) {
                final String entityName = entityType.getName();
            }
        }
    }
}

```

```
        final Query query = session.createQuery("from " + entityName);
        System.out.println("executing: " + query.getQueryString());
        for (Object o : query.list()) {
            System.out.println(" " + o);
        }
    }
} finally {
    session.close();
}
}
```

d) Rezultaty działania programu:

- Schemat bazy danych



- Definicja tabeli *Supplier_Product* w bazie

```
-- auto-generated definition
create table SUPPLIER_PRODUCT
(
    SUPPLIER_SUPPLIERID INTEGER not null
        constraint FK24J3KWM0YSJ1J4X3TDHPANE6G
            references SUPPLIER,
    PRODUCTS_PRODUCTID INTEGER not null
        unique
        constraint FKKQ9CRHXKU499AK5YBE26S8WK4
            references PRODUCT,
    primary key (SUPPLIER_SUPPLIERID, PRODUCTS_PRODUCTID)
);
```

- Select z tabeli *Product*

The screenshot shows the DBeaver IDE interface. At the top, there's a toolbar with various icons like play, stop, refresh, and a dropdown set to 'Tx: Auto'. Below the toolbar, a status bar displays '1 ✓' and the SQL query 'SELECT * FROM PRODUCT;'. The main area is divided into two sections: 'Output' and 'APP.PRODUCT x'. The 'Output' section has a progress bar at the bottom. The 'APP.PRODUCT x' section contains a table with three rows of data:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	11	Filter	23
2	12	Papier	34
3	13	Kubek	65

- Select z tabeli *Supplier*

```
console_5 [UTumilovichJPA] | SELECT * FROM SUPPLIER;
+-----+-----+-----+-----+
| SUPPLIERID | CITY | COMPANYNAME | STREET |
+-----+-----+-----+-----+
| 14 | Anywhere | Supplier | Somewhere |
+-----+-----+-----+-----+
```

- Select z tabeli *Supplier_Product*

```
console_5 [UTumilovichJPA] | SELECT * FROM SUPPLIER_PRODUCT;
+-----+-----+
| SUPPLIER_SUPPLIERID | PRODUCTS_PRODUCTID |
+-----+-----+
| 14 | 11 |
| 14 | 12 |
| 14 | 13 |
+-----+-----+
```

Bez tabeli łącznikowej

e) Została zmodyfikowana klasa **Supplier** dodaniem `@JoinColumn(name="Supplier_FK")`

```
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierId;
    private String companyName;
    private String street;
    private String city;

    @OneToMany
    @JoinColumn(name="Supplier_FK")
    private final Set<Product> products = new HashSet<>();

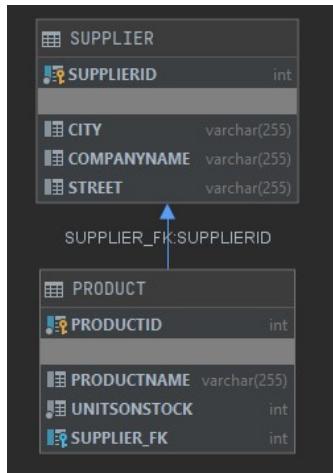
    public Supplier() {
    }

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

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

f) Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```

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 not null,
    SUPPLIER_FK INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
            references SUPPLIER
);
  
```

- Select z tabeli *Product*

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK
1	15	Filter	23	18
2	16	Papier	34	18
3	17	Kubek	65	18

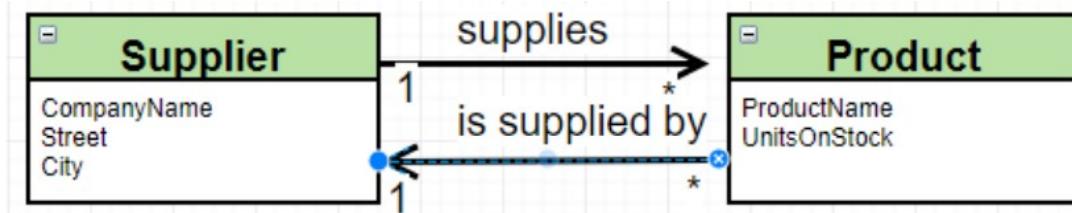
- Select z tabeli *Supplier*

The screenshot shows a database console window titled "console_5 [UTumilovichJPA]". In the top-left pane, there is a green checkmark next to the query "SELECT * FROM SUPPLIER;". The bottom-left pane shows the results of the query:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	18	Anywhere	Supplier	Somewhere

Zadanie VI

Została umodelowana relacja dwustronna jak poniżej:



- a) Została zmodyfikowana klasa **Product** dodaniem metody *setSupplier*, która dodaje dostawcę do tabeli Products oraz produkt do tabeli Supplier, jeśli jeszcze nie był dodany

```

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsOnStock;

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

    public Product() {}

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

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

- b) Dodatkowo została zmodyfikowana klasa **Supplier**, modyfikowaniem metody *addProduct* oraz dodaniem metody *getProducts*, która zwraca listę produktów

```

@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierId;
    private String companyName;
    private String street;
  
```

```

private String city;

@OneToMany
@JoinColumn(name="Supplier_FK")
private final Set<Product> products = new HashSet<>();

public Supplier() {
}

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

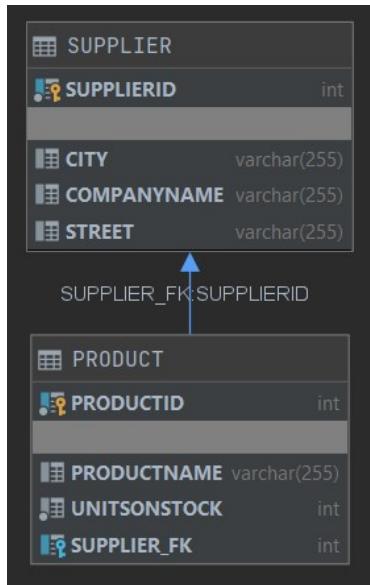
public Set<Product> getProducts() {
    return products;
}

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

```

c) Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```

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 not null,
    SUPPLIER_FK  INTEGER
)

```

```

constraint FKVE96QACVSR1A50RGWL94ENRU
    references SUPPLIER
);

```

- Select z tabeli *Product*

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK
1	27	Filter	23	30
2	28	Papier	34	30
3	29	Kubek	65	30

- Select z tabeli *Supplier*

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	30	Anywhere	Supplier	Somewhere

Zadanie VII

- a) Została dodana klasa **Category** z property *int categoryId, String name* oraz listą produktów *List products*

```

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

    @OneToMany
    @JoinColumn(name="Category_FK")
    private final List<Product> products = new ArrayList<>();

    public Category() {}

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

    public String getName() {
        return name;
    }
}

```

```

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

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

```

b) Został zmodyfikowany plik konfiguracyjny, dodaniem klasy **Category** do mapy

```

<?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/UTumilovichJPA</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>
        <!-- DB schema will be updated if needed -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
        <mapping class="Category"></mapping>
    </session-factory>
</hibernate-configuration>

```

c) Następnie została zmodyfikowana klasa **Product**, dodaniem pola *supplier* oraz wspomagających metod seterów i geterów

```

@Entity
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsOnStock;

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

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

    public Product() {
    }

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

    public String getProductName() {
        return productName;
    }

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

```

```
    }

    public Category getCategory() {
        return category;
    }

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

d) Na koniec została zmodyfikowana metoda `main` oraz dodane metody `printCategoryProducts` i `printProductCategory` klasy **Main**, dla testowania poprawności działania programu

```
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();
    }

    private static void printCategoryProducts(int categoryId) {
        Category category = getSession().get(Category.class, categoryId);
        System.out.println(category.getName() + " product list:");
        for (Product product: category.getProducts()) {
            System.out.println(product.getProductName());
        }
    }

    private static void printProductCategory(int productId) {
        Product product = getSession().get(Product.class, productId);
        System.out.println("Product: " + product.getProductName() + ", Category: " + product.getCategory().getName());
    }

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

        Transaction transaction = session.beginTransaction();

        // Adding category to existing products
        Category category1 = new Category("Other");
        Product product1 = session.get(Product.class, 37);
        Product product2 = session.get(Product.class, 38);
        Product product3 = session.get(Product.class, 39);

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

        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(category1);

        // Creating new products, suppliers and categories
    }
}
```

```
Product newProduct1 = new Product("Banana", 43);
Product newProduct2 = new Product("Orange", 54);
Product newProduct3 = new Product("Lemon", 23);
Supplier newSupplier = new Supplier("FoodCompany", "Budryka", "Kraków");
Category newCategory = new Category("Fruits");

newSupplier.addProduct(newProduct1);
newSupplier.addProduct(newProduct2);
newSupplier.addProduct(newProduct3);

newCategory.addProduct(newProduct1);
newCategory.addProduct(newProduct2);
newCategory.addProduct(newProduct3);

session.save(newProduct1);
session.save(newProduct2);
session.save(newProduct3);
session.save(newSupplier);
session.save(newCategory);

printCategoryProducts(41);
printProductCategory(37);

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("from " + entityName);
        System.out.println("executing: " + query.getQueryString());
        for (Object o : query.list()) {
            System.out.println(" " + o);
        }
    }
} finally {
    session.close();
}
}
```

c) Rezultaty działania programu:

- Rezultat wypisania listy produktów podanej kategorii:

```

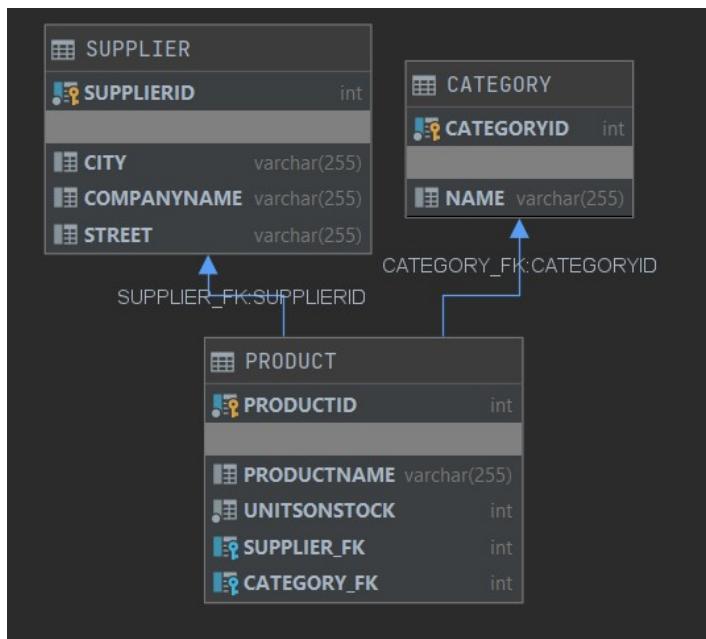
Other product list:
Hibernate:
    select
        products0_.Category_FK as category4_1_0_,
        products0_.productId as producti1_1_0_,
        products0_.productId as producti1_1_1_,
        products0_.Category_FK as category4_1_1_,
        products0_.productName as productn2_1_1_,
        products0_.Supplier_FK as supplier5_1_1_,
        products0_.unitsOnStock as unitsons3_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_.Supplier_FK=supplier1_.supplierId
    where
        products0_.Category_FK=?
Filter
Papier
Kubek

```

- Rezultat wypisania nazwy kategorii dla podanego produktu:

Product: Filter, Category: Other

- Schemat bazy danych



- Definicja bazy:

```

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 not null,
SUPPLIER_FK   INTEGER
    constraint FKVE96QACVSR1A50RGWL94ENRU
        references SUPPLIER,
CATEGORY_FK   INTEGER
    constraint FKKGKXD6GNQYXWWOA0GK95PT3D
        references CATEGORY
);

```

- Select z tabeli *Product*

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK	CATEGORY_FK
1	27	Filter	23	30	31
2	28	Papier	34	30	31
3	29	Kubek	65	30	31
4	32	Banana	43	35	36
5	33	Orange	54	35	36
6	34	Lemon	23	35	36

- Select z tabeli *Supplier*

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	30	Anywhere	Supplier	Somewhere
2	35	Kraków	FoodCompany	Budryka

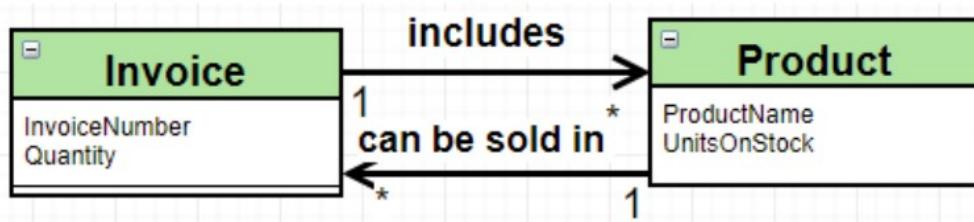
- Select z tabeli *Category*

The screenshot shows a database console window titled "console_5 [UTumilovichJPA]". In the top panel, there is a green checkmark icon followed by the text "1 ✓" and the SQL query "SELECT * FROM CATEGORY;". Below this, the "Output" tab is selected, showing a table titled "APP.CATEGORY" with two rows. The columns are "CATEGORYID" and "NAME". The data is as follows:

	CATEGORYID	NAME
1	31	Other
2	36	Fruits

Zadanie VIII

Została umodelowana relacja wiele-do-wielu jak poniżej:



a) Została utworzona klasa **Invoice** z polami *invoiceNumber* oraz *quantity*

```

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

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

    public Invoice() {
    }

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

    public int getInvoiceNumber() {
        return invoiceNumber;
    }

    public int getQuantity() {
        return quantity;
    }

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

    public void addProduct(Product product) {
        products.add(product);
        product.getInvoices().add(this);
        this.quantity++;
    }
}
  
```

b) Został zmodyfikowany plik konfiguracyjny, dodaniem klasy **Invoice** do mapy

```
<?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/UTumilovichJPA</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>
    <!-- DB schema will be updated if needed -->
    <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>
```

c) Następnie została zmodyfikowana klasa **Product**, dodaniem pola setu atrybutów *invoices* oraz metody *getInvoices*

```
@Entity
public class Product {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int productId;
  private String productName;
  private int unitsOnStock;

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

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

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

  public Product() {
  }

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

  public String getProductName() {
    return productName;
  }

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

  public Category getCategory() {
    return category;
  }

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

```

        if (!category.getProducts().contains(this)) {
            category.addProduct(this);
        }
    }

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

```

d) Na koniec została zmodyfikowana metoda `_main` oraz dodane metody `printInvoiceProducts` i `printProductInvoices` klasy **Main**

```

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();
    }

    private static void printInvoiceProducts(int invoiceNumber) {
        Invoice invoice = getSession().get(Invoice.class, invoiceNumber);
        System.out.println("Invoice number:" + invoiceNumber);
        for (Product product: invoice.getProducts()) {
            System.out.println(product.getProductName());
        }
    }

    private static void printProductInvoices(int productId) {
        Product product = getSession().get(Product.class, productId);
        System.out.println("Product: " + product.getProductName());
        for (Invoice invoice: product.getInvoices()) {
            System.out.println(invoice.getInvoiceNumber());
        }
    }

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

        Transaction transaction = session.beginTransaction();

        Product newProduct1 = new Product("Milk", 45);
        Product newProduct2 = new Product("Yogurt", 76);
        Product newProduct3 = new Product("Crisps", 34);
        Product newProduct4 = new Product("Bread", 56);
        Product newProduct5 = new Product("Jam", 65);
        Product newProduct6 = new Product("Meat", 2);
        Product newProduct7 = new Product("Cookies", 3);

        Supplier newSupplier = new Supplier("Grocery store", "Reymonta", "Kraków");
        Category newCategory = new Category("Food");
        Invoice newInvoice1 = new Invoice(0);
        Invoice newInvoice2 = new Invoice(0);

        newSupplier.addProduct(newProduct1);
        newSupplier.addProduct(newProduct2);
        newSupplier.addProduct(newProduct3);
        newSupplier.addProduct(newProduct4);
        newSupplier.addProduct(newProduct5);
        newSupplier.addProduct(newProduct6);
    }
}

```

```

newSupplier.addProduct(newProduct7);

newCategory.addProduct(newProduct1);
newCategory.addProduct(newProduct2);
newCategory.addProduct(newProduct3);
newCategory.addProduct(newProduct4);
newCategory.addProduct(newProduct5);
newCategory.addProduct(newProduct6);
newCategory.addProduct(newProduct7);

newInvoice1.addProduct(newProduct1);
newInvoice1.addProduct(newProduct2);
newInvoice1.addProduct(newProduct3);
newInvoice1.addProduct(newProduct4);
newInvoice1.addProduct(newProduct5);
newInvoice2.addProduct(newProduct4);
newInvoice2.addProduct(newProduct5);
newInvoice2.addProduct(newProduct6);
newInvoice2.addProduct(newProduct7);

session.save(newProduct1);
session.save(newProduct2);
session.save(newProduct3);
session.save(newProduct4);
session.save(newProduct5);
session.save(newProduct6);
session.save(newProduct7);

session.save(newSupplier);
session.save(newCategory);
session.save(newInvoice1);
session.save(newInvoice2);

printInvoiceProducts(56);
printProductInvoices(50);

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("from " + entityName);
        System.out.println("executing: " + query.getQueryString());
        for (Object o : query.list()) {
            System.out.println(" " + o);
        }
    }
} finally {
    session.close();
}
}

```

e) Rezultaty działania programu:

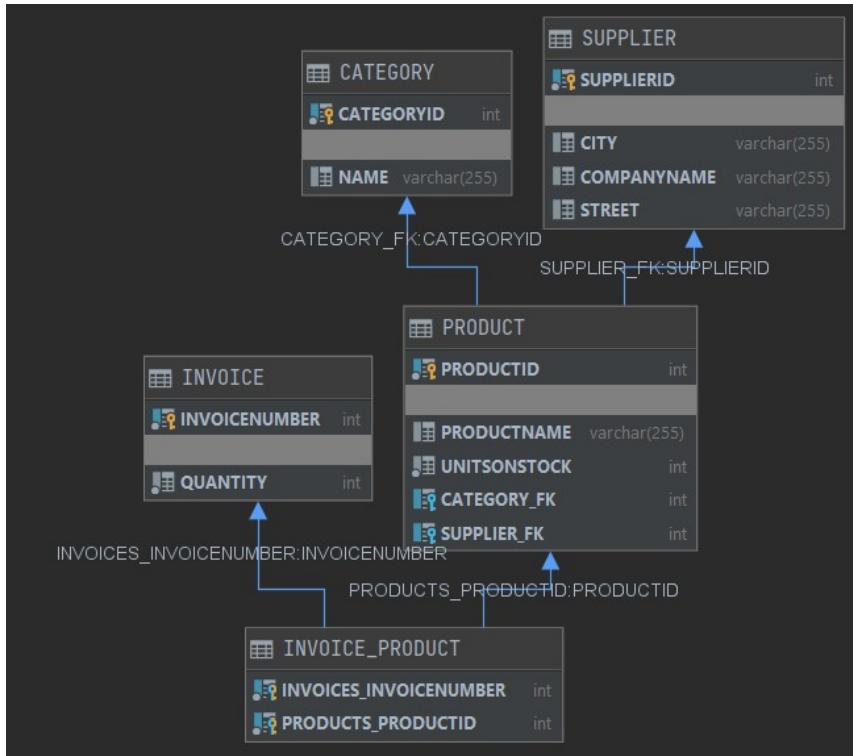
- Rezultat wypisania listy produktów podanej transakcji:

Crisps
Yogurt
Bread
Jam
Milk

- Rezultat wypisania listy transakcji dla podanego produktu:

56

- Schemat bazy danych



- Definicja bazy:

```

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

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

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 not null,
    CATEGORY_FK    INTEGER
        constraint FKRGKXD6GNQYXW0A0GK95PT3D
            references CATEGORY,
    SUPPLIER_FK    INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
            references SUPPLIER
);
    
```

```
create table INVOICE_PRODUCT
(
    INVOICES_INVOICENUMBER INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
            references INVOICE,
    PRODUCTS_PRODUCTID      INTEGER not null
        constraint FK30WFENK1TV2NDEPWLJ01N1G5
            references PRODUCT,
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)
);
```

- Select z tabeli *Product*

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY_FK	SUPPLIER_FK
1	47	Milk	45	55	54
2	48	Yogurt	76	55	54
3	49	Crisps	34	55	54
4	50	Bread	56	55	54
5	51	Jam	65	55	54
6	52	Meat	2	55	54
7	53	Cookies	3	55	54

- Select z tabeli *Supplier*

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	54	Kraków	Grocery store	Reymonta

- Select z tabeli *Category*

The screenshot shows a database console window titled "console_5 [UTumilovichJPA]". In the top panel, there is a green checkmark icon followed by the SQL query: "SELECT * FROM CATEGORY;". Below the query, the results are displayed in a table:

	CATEGORYID	NAME
1	55	Food

- Select z tabeli *Invoice*

The screenshot shows a database console window titled "console_5 [UTumilovichJPA]". In the top panel, there is a green checkmark icon followed by the SQL query: "SELECT * FROM INVOICE;". Below the query, the results are displayed in a table:

	INVOICENUMBER	QUANTITY
1	56	5
2	57	4

- Select z tabeli *Invoice_Product*

The screenshot shows a database console window titled "console_5 [UTumilovichJPA]". In the top panel, there is a green checkmark icon followed by the SQL query: "SELECT * FROM INVOICE_PRODUCT;". Below the query, the results are displayed in a table:

	INVOICES_INVOICENUMBER	PRODUCTS_PRODUCTID
1	56	47
2	56	48
3	56	49
4	56	50
5	56	51
6	57	50
7	57	51
8	57	52
9	57	53

Zadanie X

a) W folderze META-INF został stworzony plik **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="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/UTumilovichJPA"/>
            <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>
```

b) Została stworzona nowa klasa **MainJPA** z przykładowymi obiektami do uzupełnienia bazy

```
public class MainJPA {
    private static EntityManagerFactory entityManagerFactory;

    private static EntityManager getEntityManager() {
        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("derby");
        }
        return entityManagerFactory.createEntityManager();
    }

    public static void main(String[] args) {
        EntityManager entityManager = getEntityManager();
        EntityTransaction entityTransaction = entityManager.getTransaction();
        entityTransaction.begin();

        Product product1 = new Product("Filter", 23);
        Product product2 = new Product("Papier", 34);
        Product product3 = new Product("Kubek", 65);

        Supplier supplier = new Supplier("Supplier", "Somewhere", "Anywhere");

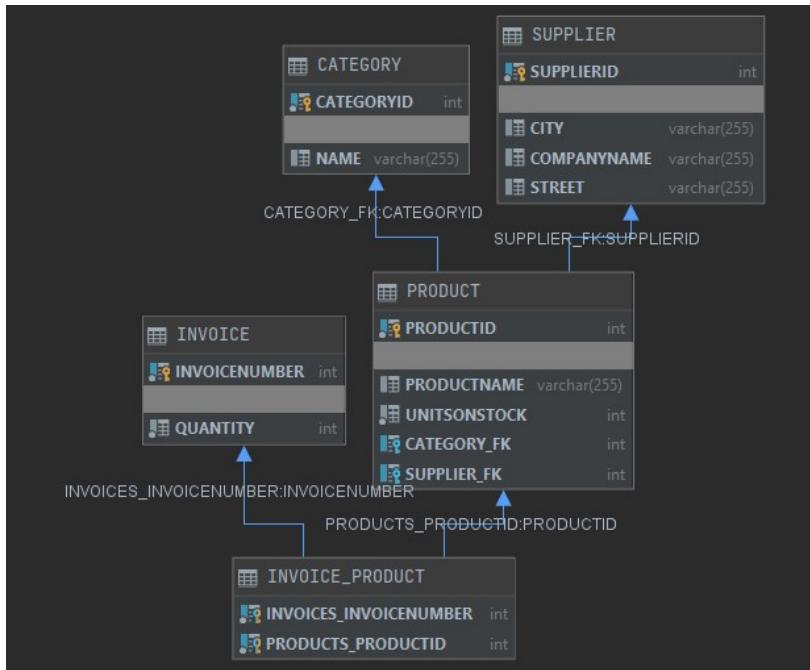
        supplier.addProduct(product1);
        supplier.addProduct(product2);
        supplier.addProduct(product3);

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

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

c) Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```

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

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

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 not null,
    CATEGORY_FK    INTEGER
        constraint FKKRGKXD6GNQYXW0A0GK95PT3D
        references CATEGORY,
    SUPPLIER_FK    INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
        references SUPPLIER
);

create table INVOICE_PRODUCT
(
    INVOICES_INVOICENUMBER INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
        references INVOICE,
    PRODUCTS_PRODUCTID     INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
        references PRODUCT
);
    
```

```

PRODUCTS_PRODUCTID      INTEGER not null
    constraint FK30WFENK1TV2NDEPWLJ01N1G5
        references PRODUCT,
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)
);

```

- Select z tabeli *Product*

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Filter	23	<null>	4
2	2	Papier	34	<null>	4
3	3	Kubek	65	<null>	4

- Select z tabeli *Supplier*

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Anywhere	Supplier	Somewhere

Zadanie XI

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

- a) Została zmodyfikowana klasa **Product** dodaniem dodatkowych parametrów polu *invoices*

```

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsOnStock;

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

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

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

```

```

public Product() {
}

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

public String getProductName() {
    return productName;
}

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

public Category getCategory() {
    return category;
}

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

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

```

b) Została zmodyfikowana klasa **Invoices** dodaniem dodatkowych parametrów polu *products*

```

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

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

    public Invoice() {
    }

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

    public int getInvoiceNumber() {
        return invoiceNumber;
    }

    public int getQuantity() {
        return quantity;
    }

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

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

```

```
        this.quantity++;
    }
}
```

c) Została zmiodyfikowana metoda *main* klasy **MainJPA** w taki sposób, że po utworzeniu produktów i dodaniu ich do transakcji zostały one nie dodane do bazy

```
public class MainJPA {
    private static EntityManagerFactory entityManagerFactory;

    private static EntityManager getEntityManager() {
        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("derby");
        }
        return entityManagerFactory.createEntityManager();
    }

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

        Product newProduct1 = new Product("Milk", 45);
        Product newProduct2 = new Product("Yogurt", 76);
        Product newProduct3 = new Product("Crisps", 34);
        Product newProduct4 = new Product("Bread", 56);
        Product newProduct5 = new Product("Jam", 65);
        Product newProduct6 = new Product("Meat", 2);
        Product newProduct7 = new Product("Cookies", 3);

        Supplier newSupplier = new Supplier("Grocery store", "Reymonta", "Kraków");
        Category newCategory = new Category("Food");
        Invoice newInvoice1 = new Invoice(0);
        Invoice newInvoice2 = new Invoice(0);

        newSupplier.addProduct(newProduct1);
        newSupplier.addProduct(newProduct2);
        newSupplier.addProduct(newProduct3);
        newSupplier.addProduct(newProduct4);
        newSupplier.addProduct(newProduct5);
        newSupplier.addProduct(newProduct6);
        newSupplier.addProduct(newProduct7);

        newCategory.addProduct(newProduct1);
        newCategory.addProduct(newProduct2);
        newCategory.addProduct(newProduct3);
        newCategory.addProduct(newProduct4);
        newCategory.addProduct(newProduct5);
        newCategory.addProduct(newProduct6);
        newCategory.addProduct(newProduct7);

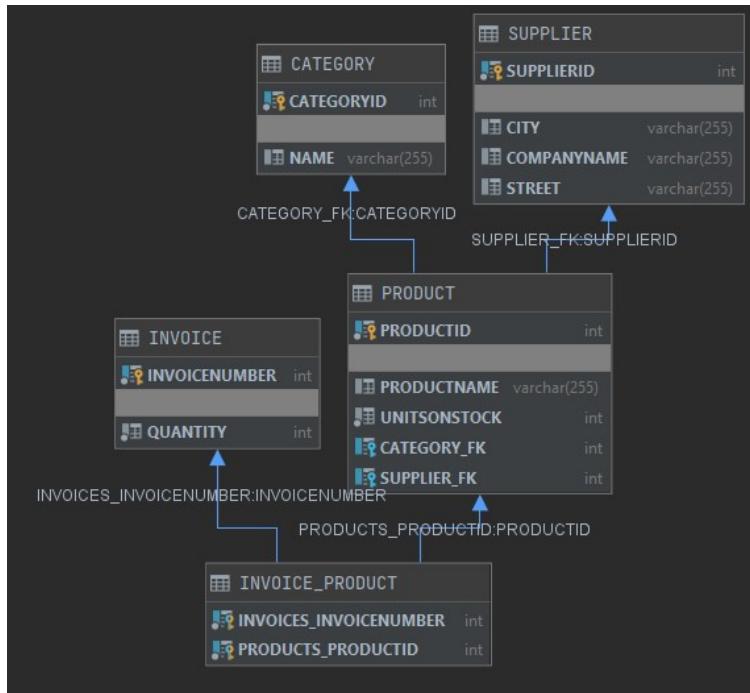
        newInvoice1.addProduct(newProduct1);
        newInvoice1.addProduct(newProduct2);
        newInvoice1.addProduct(newProduct3);
        newInvoice1.addProduct(newProduct4);
        newInvoice1.addProduct(newProduct5);
        newInvoice2.addProduct(newProduct4);
        newInvoice2.addProduct(newProduct5);
        newInvoice2.addProduct(newProduct6);
        newInvoice2.addProduct(newProduct7);

        entityManager.persist(newSupplier);
        entityManager.persist(newCategory);
        entityManager.persist(newInvoice1);
        entityManager.persist(newInvoice2);

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

f) Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```

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

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

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 not null,
    CATEGORY_FK  INTEGER
        constraint FKKRGKXD6GNQYXW0A0GK95PT3D
        references CATEGORY,
    SUPPLIER_FK   INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
        references SUPPLIER
);
  
```

```
create table INVOICE_PRODUCT
(
    INVOICES_INVOICENUMBER INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
            references INVOICE,
    PRODUCTS_PRODUCTID      INTEGER not null
        constraint FK30WFENK1TV2NDEPWLJ01N1G5
            references PRODUCT,
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)
);
```

- Select z tabeli *Product*

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY_FK	SUPPLIER_FK
1	4	Crisps	34	2	1
2	5	Milk	45	2	1
3	6	Yogurt	76	2	1
4	7	Bread	56	2	1
5	9	Meat	2	2	1
6	10	Cookies	3	2	1
7	11	Jam	65	2	1

- Select z tabeli *Supplier*

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	Grocery store	Reymonta

- Select z tabeli *Category*

The screenshot shows the MySQL Workbench interface with a query editor window titled "console_5 [UTumilovichJPA]". The query "SELECT * FROM CATEGORY;" has been run and returned one row of data. The result set is displayed in a table with columns "CATEGORYID" and "NAME". The data shows a single entry with Category ID 1 and Name "Food".

CATEGORYID	NAME
1	Food

- Select z tabeli *Invoice*

The screenshot shows the MySQL Workbench interface with a query editor window titled "console_5 [UTumilovichJPA]". The query "SELECT * FROM INVOICE;" has been run and returned two rows of data. The result set is displayed in a table with columns "INVOICENUMBER" and "QUANTITY". The data shows two entries: one with Invoice Number 3 and Quantity 5, and another with Invoice Number 8 and Quantity 4.

INVOICENUMBER	QUANTITY
3	5
8	4

- Select z tabeli *Invoice_Product*

The screenshot shows the MySQL Workbench interface with a query editor window titled "console_5 [UTumilovichJPA]". The query "SELECT * FROM INVOICE_PRODUCT;" has been run and returned nine rows of data. The result set is displayed in a table with columns "INVOICES_INVOICENUMBER" and "PRODUCTS_PRODUCTID". The data shows various product IDs associated with different invoice numbers, with some products appearing in multiple invoices.

INVOICES_INVOICENUMBER	PRODUCTS_PRODUCTID
3	4
3	5
3	6
3	7
3	11
8	7
8	9
8	10
8	11

Zadanie XII

- Została stworzona klasa **Address** z polami *country*, *city*, *street*, *zipCode*

```

@Embeddable
public class Address {
    private String country;
    private String city;
    private String street;
    private String zipCode;

    public Address() {
    }

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

```

b) Klasa **Address** została wprowadzona do klasy **Supplier**

```

@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierId;
    private String companyName;

    @Embedded
    private Address address;

    @OneToMany
    @JoinColumn(name="Supplier_FK")
    private final Set<Product> products = new HashSet<>();

    public Supplier() {
    }

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

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

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

```

c) Zostały utworzone przykładowe obiekty do zapisywania w bazie w metodzie main klasy **MainJPA**

```

public class MainJPA {
    private static EntityManagerFactory entityManagerFactory;

    private static EntityManager getEntityManager() {
        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("derby");
        }
        return entityManagerFactory.createEntityManager();
    }

    public static void main(String[] args) {
        EntityManager entityManager = getEntityManager();
        EntityTransaction entityTransaction = entityManager.getTransaction();
        entityTransaction.begin();
    }
}

```

```

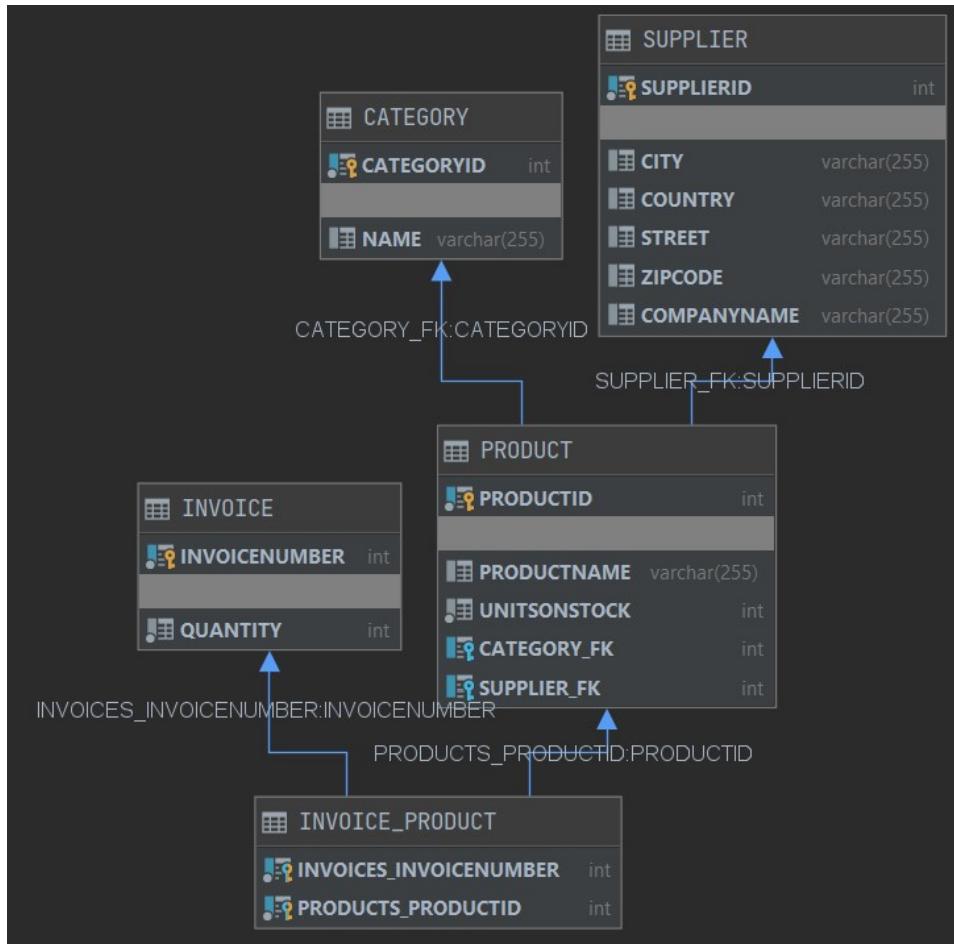
entityManager.persist(new Supplier("Grocery store",
    new Address("Poland", "Krakow", "Budryka", "30-072")));
entityManager.persist(new Supplier("Food Supplier",
    new Address("Belarus", "Minsk", "Kammenogorskaya", "220017")));

entityTransaction.commit();
entityManager.close();
}
}

```

d) Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```

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

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

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 not null,
    CATEGORY_FK  INTEGER
        constraint FKKGKXD6GNQYXW0A0GK95PT3D
        references CATEGORY,
    SUPPLIER_FK  INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
        references SUPPLIER
);

create table INVOICE_PRODUCT
(
    INVOICES_INVOICENUMBER INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
        references INVOICE,
    PRODUCTS_PRODUCTID    INTEGER not null
        constraint FK30WFENK1TV2NDEPWLTJ01N1G5
        references PRODUCT,
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)
);

```

- Select z tabeli *Supplier*

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

Query Editor: The query `SELECT * FROM SUPPLIER;` is entered in the editor.

Results Grid: The results show two rows of data from the `SUPPLIER` table:

	SUPPLIERID	CITY	COUNTRY	STREET	ZIPCODE	COMPANYNAME
1	1	Krakow	Poland	Budryka	30-072	Grocery store
2	2	Minsk	Belarus	Kamennogorskaya	220017	Food Supplier

e) Model został zmodyfikowany w taki sposób, że dane adresowe znajdują się w klasie dostawców. Zmapowano 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="Supplier_FK")
private final Set<Product> products = new HashSet<>();

public Supplier() {
}

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

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

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

```

f) Została zmieniona metoda *main* klasy **MainJPA**

```

public class MainJPA {
    private static EntityManagerFactory entityManagerFactory;

    private static EntityManager getEntityManager() {
        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("derby");
        }
        return entityManagerFactory.createEntityManager();
    }

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

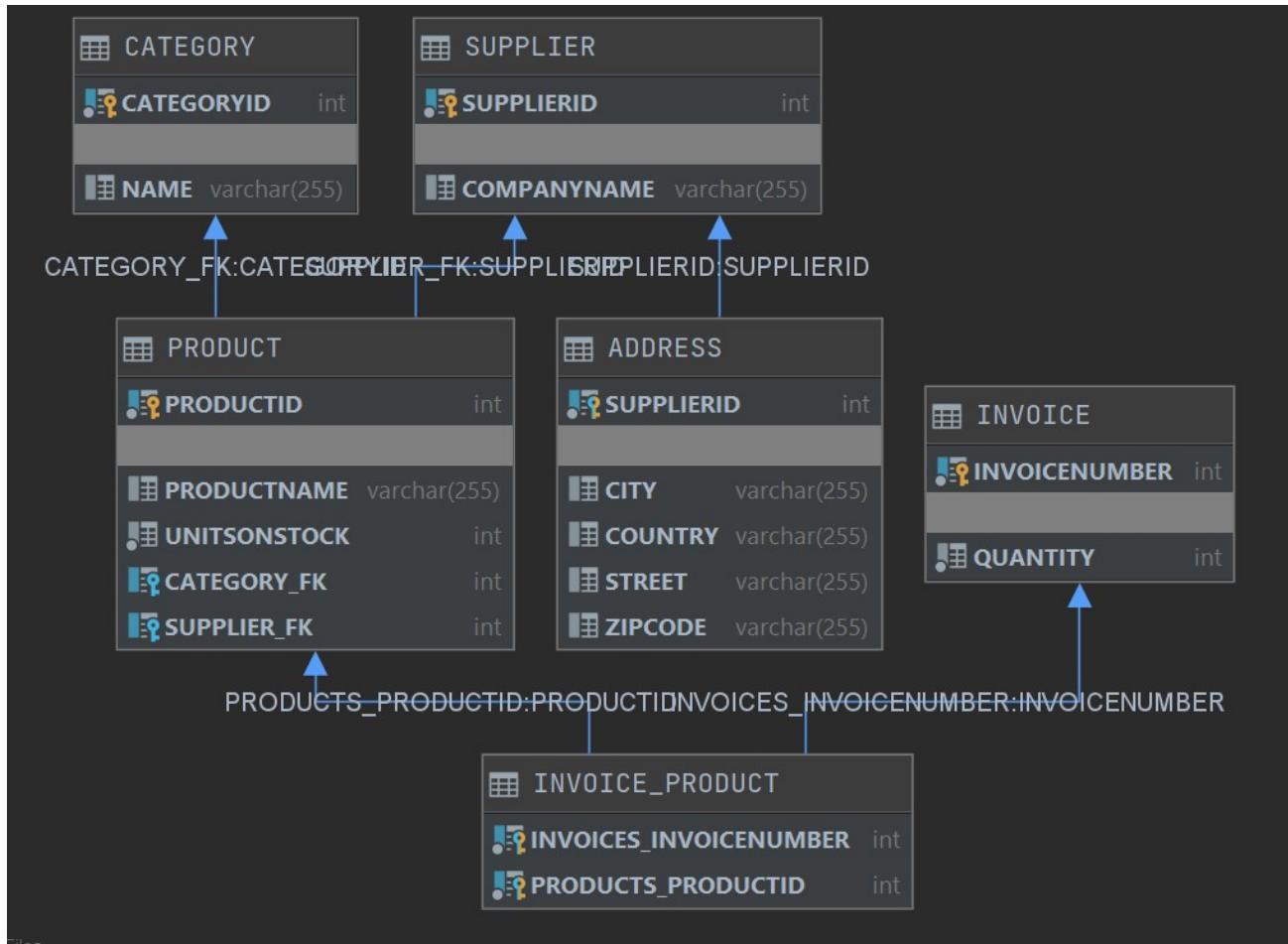
        entityManager.persist(new Supplier("Grocery store", "Poland", "Krakow", "Budryka", "30-072"));
        entityManager.persist(new Supplier("Food Supplier", "Belarus", "Minsk", "Kamennogorskaya", "220017"));

        entityTransaction.commit();
        entityManager.close();
    }
}

```

g) Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```

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

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

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 FKJ5UAJ5TD7CUTXC78VUKFD4A7K
        references SUPPLIER
);
    
```

```

);
create table PRODUCT
(
    PRODUCTID      INTEGER not null
        primary key,
    PRODUCTNAME   VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY_FK   INTEGER
        constraint FKKGKXD6GNQYXWWOA0GK95PT3D
        references CATEGORY,
    SUPPLIER_FK   INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
        references SUPPLIER
);
create table INVOICE_PRODUCT
(
    INVOICES_INVOICENUMBER INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
        references INVOICE,
    PRODUCTS_PRODUCTID     INTEGER not null
        constraint FK30WFENK1TV2NDEPWLJ01N1G5
        references PRODUCT,
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)
);

```

- Select z tabeli *Supplier*

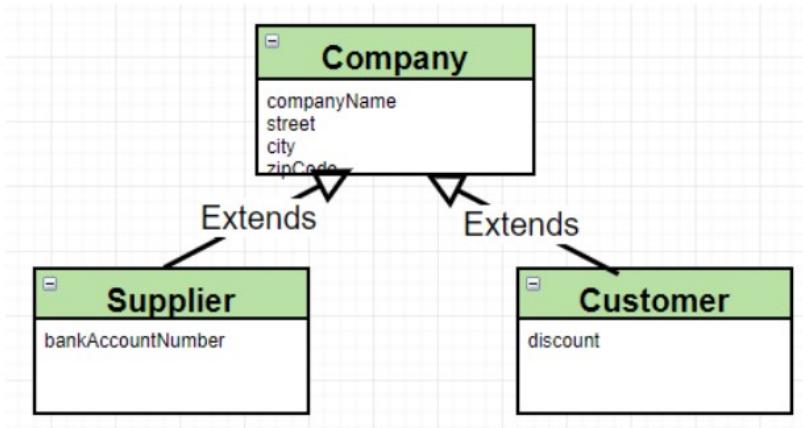
	SUPPLIERID	COMPANYNAME
1	1	Grocery store
2	2	Food Supplier

- Select z tabeli *Address*

CITY	COUNTRY	STREET	ZIPCODE	SUPPLIERID
Krakow	Poland	Budryka	30-072	1
Minsk	Belarus	Kammennogorskaya	220017	2

Zadanie XIII

Do modelu została wprowadzona następująca hierarchia:



a) Single table strategy

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) {
        this.companyName = companyName;
        this.country = country;
        this.city = city;
        this.street = street;
        this.zipCode = zipCode;
    }
}
  
```

Klasa Customer

```

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

    public Customer() {}

    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;
    }
}

```

Klasa Supplier

```

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

    @OneToMany
    @JoinColumn(name="Supplier_FK")
    private final Set<Product> products = new HashSet<>();

    public Supplier() {
    }

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

    public String getBankAccountNumber() {
        return bankAccountNumber;
    }

    public void setBankAccountNumber(String bankAccountNumber) {
        this.bankAccountNumber = bankAccountNumber;
    }

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

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

```

Klasa MainJPA

```

public class MainJPA {
    private static EntityManagerFactory entityManagerFactory;

    private static EntityManager getEntityManager() {
        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("derby");
        }
        return entityManagerFactory.createEntityManager();
    }

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

        Supplier newSupplier1 = new Supplier("Food Store", "Poland", "Kraków",
                                             "Budryka", "30-072", "12345678");
        Supplier newSupplier2 = new Supplier("Elektronics Store", "Poland", "Warsaw",
                                             "Centralna", "30-342", "87654321");
        Customer newCustomer1 = new Customer("Techno", "Belarus", "Minsk",
                                              "Kamennogorskaya", "220017", 0.15);
        Customer newCustomer2 = new Customer("FoodMania", "Belarus", "Brest",
                                              "Mogilowskaya", "343234", 0.03);
    }
}

```

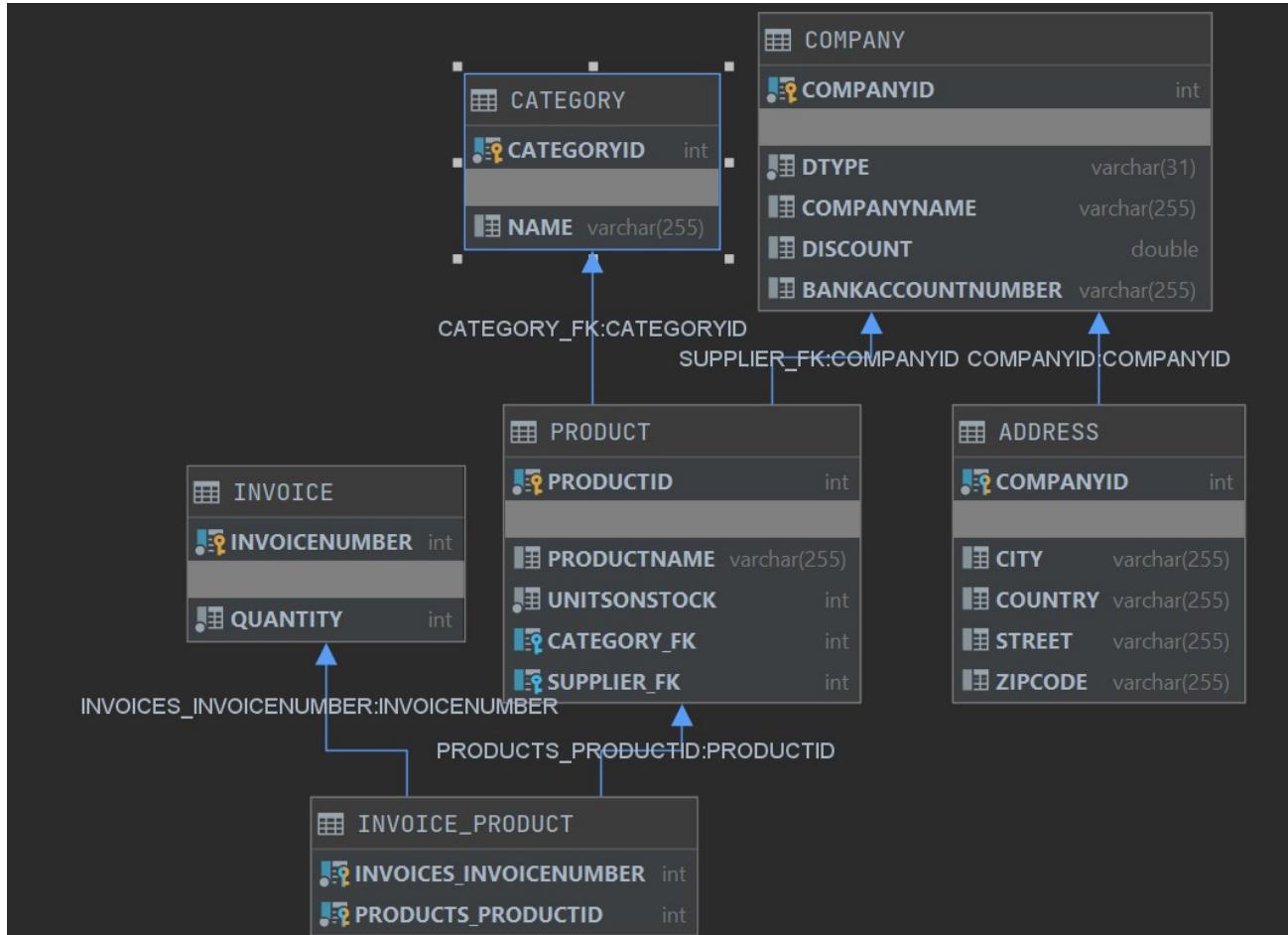
```
        entityManager.persist(newSupplier1);
        entityManager.persist(newSupplier2);
        entityManager.persist(newCustomer1);
        entityManager.persist(newCustomer2);

        entityTransaction.commit();
        entityManager.close();
    }

}
```

Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

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

create table COMPANY
(
    DTTYPE           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 FKMDMN12VHMKR08PII3XAK07XG9
        references COMPANY
);

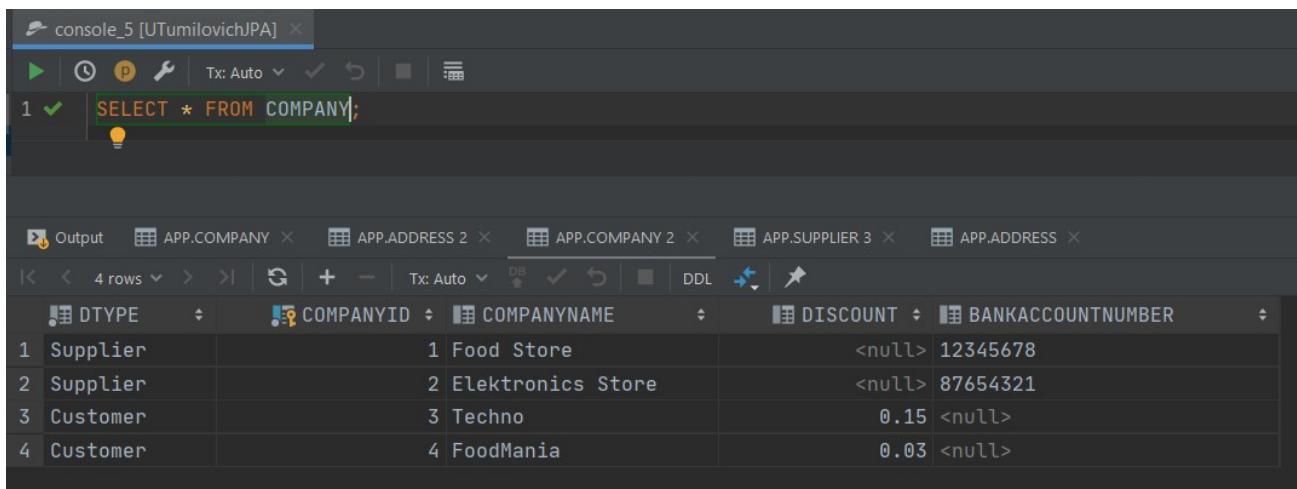
create table INVOICE
(
    INVOICENUMBER INTEGER not null
        primary key,
    QUANTITY      INTEGER not null
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY_FK  INTEGER
        constraint FKKRGKXD6GNQYXWWOA0GK95PT3D
        references CATEGORY,
    SUPPLIER_FK  INTEGER
        constraint FK514DWEWBVHD00SS9KHJQKJDJM
        references COMPANY
);

create table INVOICE_PRODUCT
(
    INVOICES_INVOICENUMBER INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
        references INVOICE,
    PRODUCTS_PRODUCTID     INTEGER not null
        constraint FK30WFENK1TV2NDEPWLTJ01N1G5
        references PRODUCT,
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)
);

```

- Select z tabeli Company



The screenshot shows a database console window titled "console_5 [UTumilovichJPA]". The query "SELECT * FROM COMPANY;" has been run, indicated by a green checkmark in the status bar. The results are displayed in a table below:

	DTYPE	COMPANYID	COMPANYNAME	DISCOUNT	BANKACCOUNTNUMBER
1	Supplier	1	Food Store	<null>	12345678
2	Supplier	2	Elektronics Store	<null>	87654321
3	Customer	3	Techno	0.15	<null>
4	Customer	4	FoodMania	0.03	<null>

- Select z tabeli Address

The screenshot shows a database management interface. At the top, there's a toolbar with various icons like play, stop, and refresh. Below it is a query editor window titled 'console_5 [UTumilovichJPA]'. Inside, a green checkmark indicates a successful execution of the query: 'SELECT * FROM ADDRESS;'. The main area displays a table with four rows of data from the 'ADDRESS' table. The columns are labeled: CITY, COUNTRY, STREET, ZIPCODE, and COMPANYID. The data is as follows:

	CITY	COUNTRY	STREET	ZIPCODE	COMPANYID
1	Kraków	Poland	Budryka	30-072	1
2	Warsaw	Poland	Centralna	30-342	2
3	Minsk	Belarus	Kamennogorskaya	220017	3
4	Brest	Belarus	Mogilowskaya	343234	4

b) Joined strategy

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;

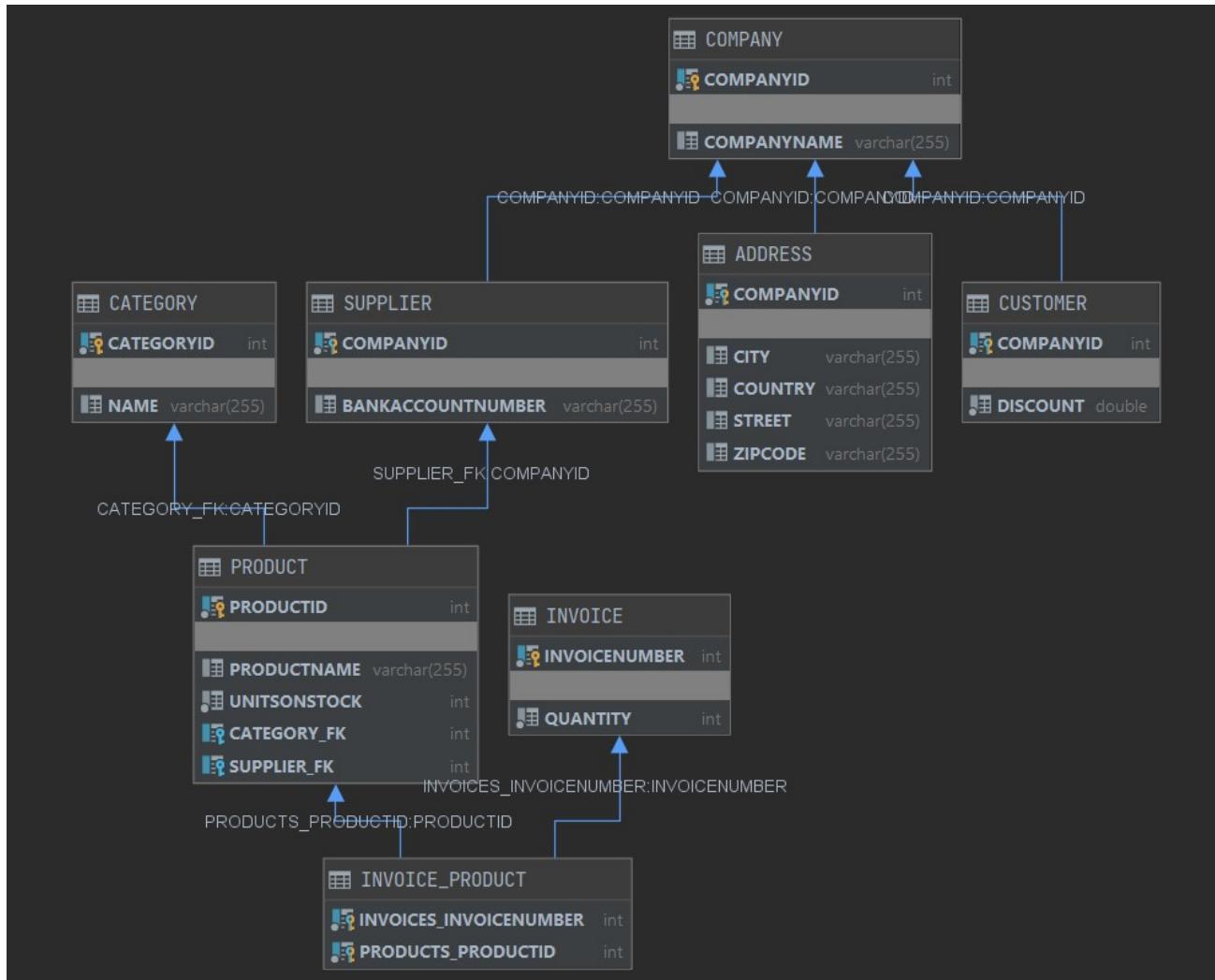
    public Company() {
    }

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

```

Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```

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 FKMDMN12VHMKR08PII3XAK07XG9
        references COMPANY
);

create table CUSTOMER
(

```

```

DISCOUNT  DOUBLE  not null,
COMPANYID  INTEGER not null
    primary key
    constraint FKQ5B9XCMQP3UIH4U11R37L6229
        references COMPANY
);

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

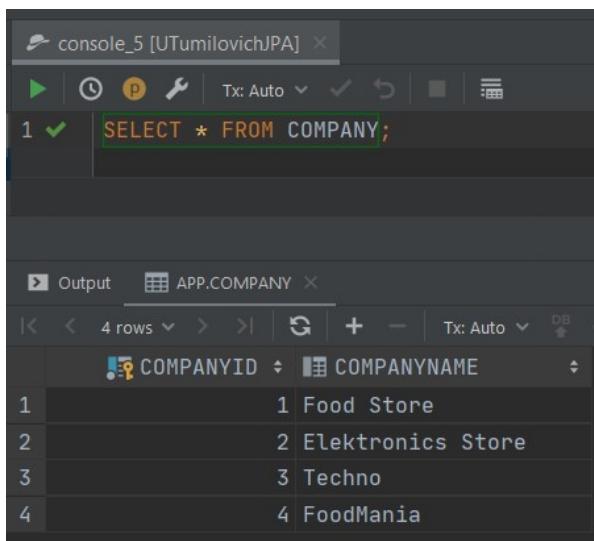
create table SUPPLIER
(
    BANKACCOUNTNUMBER  VARCHAR(255),
    COMPANYID          INTEGER not null
        primary key
    constraint FK43EKYFATN1CY2AW2U5FGM8ETW
        references COMPANY
);

create table PRODUCT
(
    PRODUCTID      INTEGER not null
        primary key,
    PRODUCTNAME   VARCHAR(255),
    UNITSONSTOCK  INTEGER not null,
    CATEGORY_FK   INTEGER
        constraint FKKGKXD6GNQYXWW0A0GK95PT3D
            references CATEGORY,
    SUPPLIER_FK   INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
            references SUPPLIER
);

create table INVOICE_PRODUCT
(
    INVOICES_INVOICENUMBER  INTEGER not null
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT
            references INVOICE,
    PRODUCTS_PRODUCTID      INTEGER not null
        constraint FK30WFENK1TV2NDEPWLTLJ01N1G5
            references PRODUCT,
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)
);

```

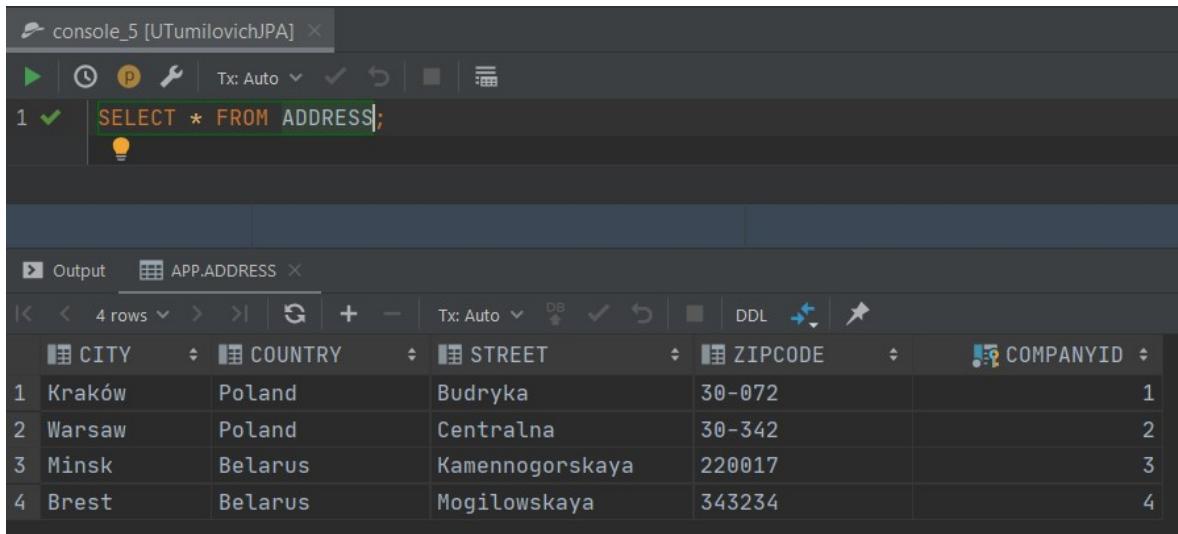
- Select z tabeli Company



The screenshot shows the IntelliJ IDEA Database tool interface. At the top, there's a toolbar with various icons. Below it is a query editor window titled "console_5 [UTumilovichJPA]". The query "SELECT * FROM COMPANY;" is entered in the editor. To the right of the editor is an "Output" window titled "APP.COMPANY". This window displays a table with four rows of data:

	COMPANYID	COMPANYNAME
1	1	Food Store
2	2	Elektronics Store
3	3	Techno
4	4	FoodMania

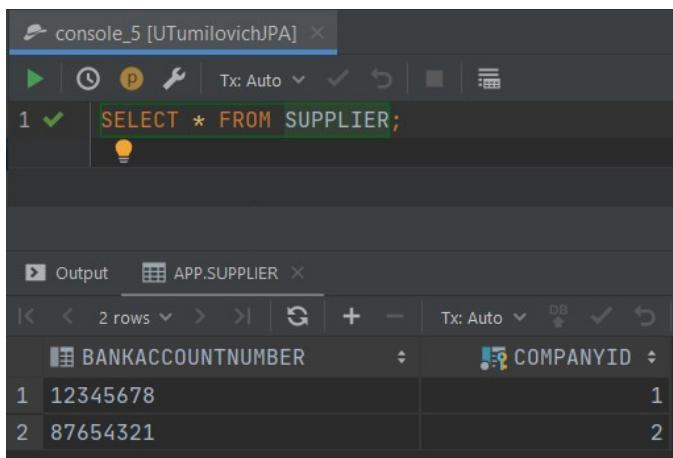
- Select z tabeli *Address*



The screenshot shows the DBeaver interface with a query window containing the SQL command `SELECT * FROM ADDRESS;`. Below the query window is the output pane titled "APP.ADDRESS" which displays the following data:

	CITY	COUNTRY	STREET	ZIPCODE	COMPANYID
1	Kraków	Poland	Budryka	30-072	1
2	Warsaw	Poland	Centralna	30-342	2
3	Minsk	Belarus	Kamennogorskaya	220017	3
4	Brest	Belarus	Mogilowskaya	343234	4

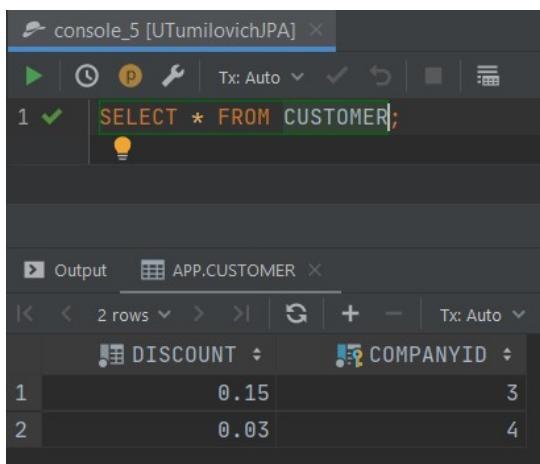
- Select z tabeli *Supplier*



The screenshot shows the DBeaver interface with a query window containing the SQL command `SELECT * FROM SUPPLIER;`. Below the query window is the output pane titled "APP.SUPPLIER" which displays the following data:

	BANKACCOUNTNUMBER	COMPANYID
1	12345678	1
2	87654321	2

- Select z tabeli *Customer*



The screenshot shows the DBeaver interface with a query window containing the SQL command `SELECT * FROM CUSTOMER;`. Below the query window is the output pane titled "APP.CUSTOMER" which displays the following data:

	DISCOUNT	COMPANYID
1	0.15	3
2	0.03	4

c) Table per class strategy

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;

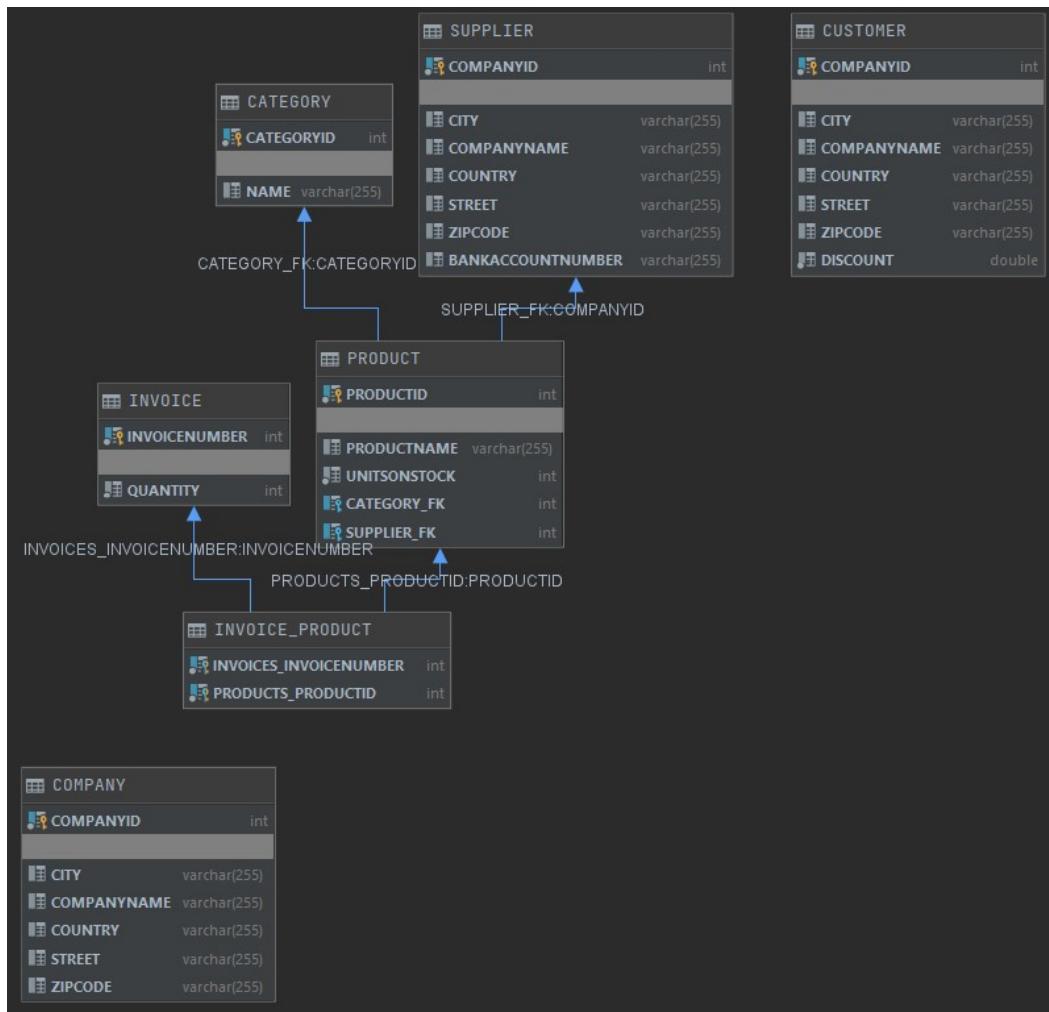
    public Company() {
    }

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

```

Rezultaty działania programu:

- Schemat bazy danych



- Definicja bazy:

```
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  
(  
    INVOICENUMBER INTEGER not null  
        primary key,  
    QUANTITY      INTEGER not null  
);  
  
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 not null,  
    CATEGORY_FK   INTEGER  
        constraint FKKGKXD6GNQYXWW0A0GK95PT3D  
            references CATEGORY,  
    SUPPLIER_FK   INTEGER  
        constraint FKVE96QACVSR1A50RGWL94ENRU  
            references SUPPLIER  
);  
  
create table INVOICE_PRODUCT  
(  
    INVOICES_INVOICENUMBER INTEGER not null  
        constraint FKCBQYL9U4EH1TWS13U6PK5J2NT  
            references INVOICE,  
    PRODUCTS_PRODUCTID     INTEGER not null  
        constraint FK30WFENK1TV2NDEPWLTJ01N1G5  
            references PRODUCT,  
    primary key (INVOICES_INVOICENUMBER, PRODUCTS_PRODUCTID)  
);
```

- Select z tabeli *Company*

The screenshot shows a JPA console interface with two panes. The top pane contains the SQL query: `SELECT * FROM COMPANY;`. The bottom pane displays the results for the APP.COMPANY table, which has columns: COMPANYID, CITY, COMPANYNAME, COUNTRY, STREET, and ZIPCODE. Two rows are shown:

	COMPANYID	CITY	COMPANYNAME	COUNTRY	STREET	ZIPCODE
1	1	Kraków	Food Store	Poland	Budryka	30-072
2	2	Warsaw	Elektronics Store	Poland	Centralna	30-342

- Select z tabeli *Supplier*

The screenshot shows a JPA console interface with two panes. The top pane contains the SQL query: `SELECT * FROM SUPPLIER;`. The bottom pane displays the results for the APP.SUPPLIER table, which has columns: COMPANYID, CITY, COMPANYNAME, COUNTRY, STREET, ZIPCODE, and BANKACCOUNTNUMBER. Two rows are shown:

	COMPANYID	CITY	COMPANYNAME	COUNTRY	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	1	Kraków	Food Store	Poland	Budryka	30-072	12345678
2	2	Warsaw	Elektronics Store	Poland	Centralna	30-342	87654321

- Select z tabeli *Customer*

The screenshot shows a JPA console interface with two panes. The top pane contains the SQL query: `SELECT * FROM CUSTOMER;`. The bottom pane displays the results for the APP.CUSTOMER table, which has columns: COMPANYID, CITY, COMPANYNAME, COUNTRY, STREET, ZIPCODE, and DISCOUNT. Two rows are shown:

	COMPANYID	CITY	COMPANYNAME	COUNTRY	STREET	ZIPCODE	DISCOUNT
1	3	Minsk	Techno	Belarus	Kamennogorskaya	220017	0.15
2	4	Brest	FoodMania	Belarus	Mogilowskaya	343234	0.03