Bazy Danych 2

Laboratorium 5

Hibernate & JPA

Mateusz Łopaciński

1. Kod po wprowadzeniu

1.1. Zaimplementowane klasy

1.1.1. Klasa Main

```
package com.matipl01;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

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();
        Product product = new Product("Krzesło", 111);
        try {
            Transaction tx = session.beginTransaction();
            session.save(product);
            tx.commit();
        } finally {
            session.close();
        }
    }
}
```

1.1.2. Klasa Product

```
package com.matipl01;

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

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

    public Product() {}

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

```
}
```

1.2. Plik konfiguracyjny hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "https://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property
name="connection.url">jdbc:derby://127.0.0.1/LopacinskiMateuszJPA</property>
        <property
name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>

        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hbm2ddl.auto">create-drop</property>

        <mapping class="com.matipl01.Product"/>
    </session-factory>
</hibernate-configuration>
```

1.3. Uzyskana tabela

	productID	productName	unitsInStock
1	1	Krzesło	111

2. Wprowadzenie pojęcia Dostawcy

2.1. Zaimplementowane klasy

2.1.1. Metoda main z klasy Main

W implementacji klasy **Main** zmieniła się jedynie metoda `public static void main`, dlatego tylko kod tej metody umieściłem poniżej. Aby móc odczytać poprzednio dodany do bazy danych produkt, zmieniłem wartość właściwości `hbm2ddl.auto` w pliku konfiguracyjnym **hibernate.cfg.xml** na `<property name="hbm2ddl.auto">update</property>`.

```
public static void main(final String[] args) {
        try (Session session = getSession()) {
            Transaction tx = session.beginTransaction();

            // Create the new supplier
            Supplier supplier = new Supplier("Super dostawca", "Malinowa",
"Poznań");

            // Get the previously added product
            Product product = session.get(Product.class, 1);
            product.setSupplier(supplier);
            session.save(supplier);
            tx.commit();

            // Testowanie
            Query query = session.createQuery("from Product");
            query.getResultList().forEach(p -> {
                System.out.println("Produkt '" + p + "' jest dostarczany przez '"
+ ((Product) p).getSupplier() + "'");
```

```
            });
        }
    }
```

2.1.2. Klasa Supplier

Aby móc skorzystać z nowo dodanej klasy dostawcy, konieczne było jej dodanie do pliku **hibernate.cfg.xml**. Umieściłem więc w pliku konfiguracyjnym linijkę: `<mapping class="com.matipl01.Supplier"/>`.

```
package com.matipl01;

import javax.persistence.*;

@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public 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;
    }

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

2.1.3. Klasa Product

```
package com.matipl01;

import javax.persistence.*;

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

    private String productName;
    private int unitsInStock;

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

    public Product() {}

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

    @Override
    public String toString() {
```

```
        return productName + " (" + unitsInStock + " szt.)";
    }

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

    public Supplier getSupplier() {
        return supplier;
    }
}
```

2.2. Logi SQL

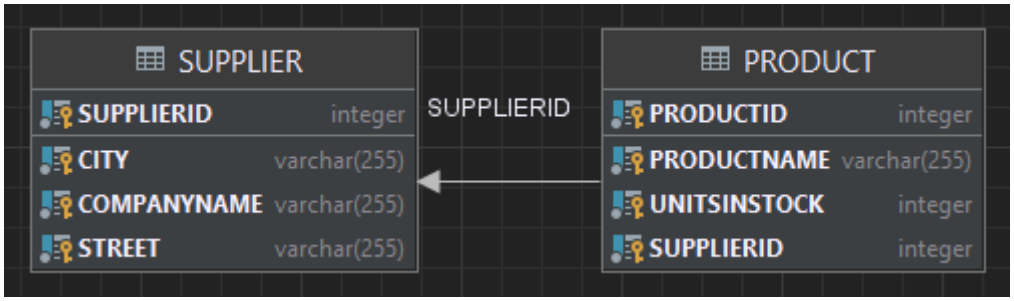
```
alter table Product
   add column supplierID integer

create table Supplier (
   supplierID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    primary key (supplierID)
)

alter table Product
   add constraint FKj0x097f8xajoy9j9ryct9pf3o
   foreign key (supplierID)
   references Supplier

select
    product0_.productID as producti1_0_0_,
    product0_.productName as productn2_0_0_,
    product0_.supplierID as supplier4_0_0_,
    product0_.unitsInStock as unitsins3_0_0_,
    supplier1_.supplierID as supplier1_1_1_,
    supplier1_.city as city2_1_1_,
    supplier1_.companyName as companyn3_1_1_,
    supplier1_.street as street4_1_1_
from
    Product product0_
left outer join
    Supplier supplier1_
        on product0_.supplierID=supplier1_.supplierID
where
    product0_.productID=?

 values
     next value for Supplier_SEQ

/* insert com.matipl01.Supplier
    */ insert
    into
        Supplier
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)


/* update
```

```
        com.matipl01.Product */ update
            Product
        set
            productName=?,
            supplierID=?,
            unitsInStock=?
        where
            productID=?

    /*
from
    Product */ select
        product0_.productID as producti1_0_,
        product0_.productName as productn2_0_,
        product0_.supplierID as supplier4_0_,
        product0_.unitsInStock as unitsins3_0_
    from
        Product product0_
```

2.3. Rezultat wykonania kodu

```
Produkt 'Krzesło (111 szt.)' jest dostarczany przez 'Super dostawca'
```

2.4. Diagram bazy danych



2.5. Utworzone tabele

2.5.1. Tabela Product

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIERID
1	1	Krzesło	111	2

2.5.2. Tabela Supplier

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	2	Poznań	Super dostawca	Malinowa

3. Odwrócenie relacji

W pliku **hibernate.cfg.xml** zmieniłem z powrotem wartość `hbm2ddl.auto` na `create-drop`, ponieważ łatwiej jest mi tworzyć wszystkie encje od nowa, a następnie umieszczać je w bazie danych.

3a. Z tabelą łącznikową

3.1. Zaimplementowane klasy

3.1.1. Metoda main z klasy Main

W implementacji tej klasy zmieniła się jedynie metoda `public static void main`, dlatego ponownie załączam tylko tę metodę.

```
public static void main(final String[] args) {
    try (Session session = getSession()) {
        Transaction tx = session.beginTransaction();

        Product product1 = new Product("Krzesło", 111);
        Product product2 = new Product("Stół", 23);
        Product product3 = new Product("Szafa", 44);
        Product product4 = new Product("Komoda", 53);

        Supplier supplier1 = new Supplier("Dostawca 1", "Malinowa", "Poznań");
        Supplier supplier2 = new Supplier("Dostawca 2", "Konwaliowa", "Kraków");
        supplier1.addProducts(product1, product3, product4);
        supplier2.addProducts(product2);

        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(product4);
        session.save(supplier1);
        session.save(supplier2);
        tx.commit();

        // Testowanie
        Query query = session.createQuery("from Supplier");
        query.getResultList().forEach(s -> {
            ((Supplier) s).getProducts().forEach(p -> System.out.println(s + "
dostarcza " + p));
        });
    }
}
```

3.1.2. Klasa Supplier

Tym razem sprzedawca zawiera zbiór produktów jako `private final Collection<Product> products = new ArrayList<>();`.

```
package com.matipl01;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;

@Table(name = "Suppliers")
@Entity
@SequenceGenerator(name = "Supplier_SEQ")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"Supplier_SEQ")
```

```
    public int supplierID;

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

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

    public Supplier() {}

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

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

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

    public void addProducts(Product ...products) {
        this.products.addAll(Arrays.asList(products));
    }
}
```

3.1.3. Klasa Product

```
package com.matipl01;

import javax.persistence.*;

@Table(name = "Products")
@Entity
@SequenceGenerator(name = "Product_SEQ")
public class Product {
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator = "Product_SEQ")
    private int productID;

    private String productName;
    private int unitsInStock;

    public Product() {}

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

    @Override
    public String toString() {
        return productName + " (" + unitsInStock + " szt.)";
    }
}
```

3.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji (używam ich po to, aby dla każdej tabeli id było generowane niezależnie, startując od 1).

```
create table Products (
   productID integer not null,
    productName varchar(255),
    unitsInStock integer not null,
    primary key (productID)
)

create table SupplierProducts (
   supplierID integer not null,
    productID integer not null
)

create table Suppliers (
   supplierID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    primary key (supplierID)
)

alter table SupplierProducts
   add constraint UK_9nc9hk63pkcj735l1lw563bh5 unique (productID)

alter table SupplierProducts
   add constraint FKnvospn0k2a1ldi72ui92wv0wg
   foreign key (productID)
   references Products

alter table SupplierProducts
   add constraint FKrglebkocbp0c6faljji6kkind
   foreign key (supplierID)
   references Suppliers

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, unitsInStock, productID)
    values
        (?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, unitsInStock, productID)
    values
        (?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, unitsInStock, productID)
    values
        (?, ?, ?)
```

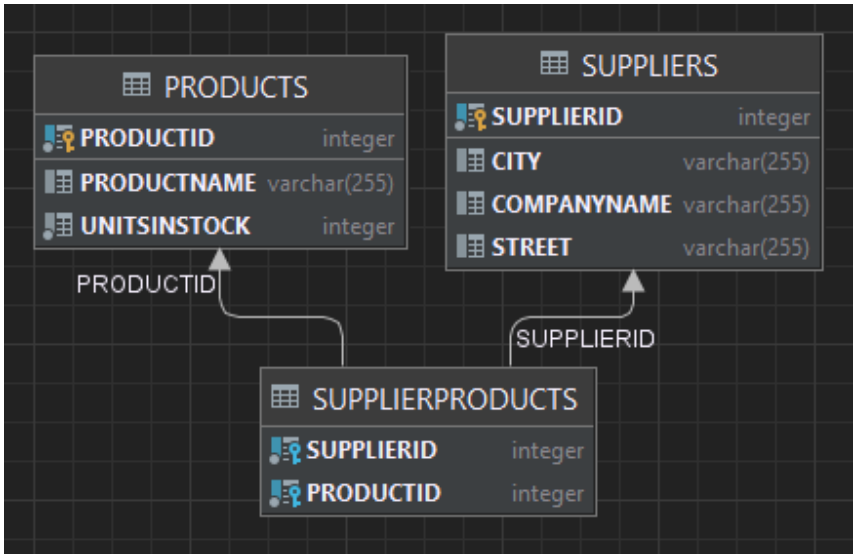
```
    /* insert com.matipl01.Product
        */ insert
        into
            Products
            (productName, unitsInStock, productID)
        values
            (?, ?, ?)

    /* insert com.matipl01.Supplier
        */ insert
        into
            Suppliers
            (city, companyName, street, supplierID)
        values
            (?, ?, ?, ?)

    /* insert com.matipl01.Supplier
        */ insert
        into
            Suppliers
            (city, companyName, street, supplierID)
        values
            (?, ?, ?, ?)

    /* insert collection
        row com.matipl01.Supplier.products */ insert
        into
            SupplierProducts
            (supplierID, productID)
        values
            (?, ?)

    /* insert collection
        row com.matipl01.Supplier.products */ insert
        into
            SupplierProducts
            (supplierID, productID)
        values
            (?, ?)

    /* insert collection
        row com.matipl01.Supplier.products */ insert
        into
            SupplierProducts
            (supplierID, productID)
        values
            (?, ?)

    /* insert collection
        row com.matipl01.Supplier.products */ insert
        into
            SupplierProducts
            (supplierID, productID)
        values
            (?, ?)

    /*
from
    Supplier */ select
        supplier0_.supplierID as supplier1_2_,
        supplier0_.city as city2_2_,
        supplier0_.companyName as companyn3_2_,
```

```
        supplier0_.street as street4_2_
    from
        Suppliers supplier0_
```

3.3. Rezultat wykonania kodu

```
Dostawca 1 dostarcza Krzesło (111 szt.)
Dostawca 1 dostarcza Szafa (44 szt.)
Dostawca 1 dostarcza Komoda (53 szt.)
Dostawca 2 dostarcza Stół (23 szt.)
```

3.4. Diagram bazy danych



3.5. Utworzone tabele

3.5.1. Tabela Products

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK
1	1	Krzesło	111
2	2	Stół	23
3	3	Szafa	44
4	4	Komoda	53

3.5.2. Tabela Suppliers

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Poznań	Dostawca 1	Malinowa
2	2	Kraków	Dostawca 2	Konwaliowa

3.5.3. Tabela SupplierProducts

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

3b. Bez tabeli łącznikowej

3.1. Zaimplementowane klasy

3.1.1. Metoda main z klasy Main

Bez zmian

3.1.2. Klasa Supplier

Zmiana dotyczy jedynie dekoratorów dekorujących kolekcję `private final Collection<Product> products = new ArrayList<>();`. Zmodyfikowany został jedynie dekorator `@JoinTable`. Otrzymujemy więc finalnie:

```
@OneToMany
@JoinColumn(name = "supplierID")
private final Collection<Product> products = new ArrayList<>();
```

3.1.3. Klasa Product

Bez zmian

3.2. Logi SQL

Pomijam **DROP TABLE** (usuwanie tabel z poprzedniego zadania) oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji (używam ich po to, aby dla każdej tabeli id było generowane niezależnie, startując od 1).

```
create table Products (
   productID integer not null,
    productName varchar(255),
    unitsInStock integer not null,
    supplierID integer,
    primary key (productID)
)

create table Suppliers (
   supplierID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    primary key (supplierID)
)

alter table Products
   add constraint FKbjx75exi25f1c48i92gu8rvlx
   foreign key (supplierID)
   references Suppliers

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, unitsInStock, productID)
    values
        (?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
```

```
        Products
        (productName, unitsInStock, productID)
    values
        (?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, unitsInStock, productID)
    values
        (?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, unitsInStock, productID)
    values
        (?, ?, ?)

/* insert com.matipl01.Supplier
    */ insert
    into
        Suppliers
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)

/* insert com.matipl01.Supplier
    */ insert
    into
        Suppliers
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)

/* create one-to-many row com.matipl01.Supplier.products */ update
    Products
set
    supplierID=?
where
    productID=?

/* create one-to-many row com.matipl01.Supplier.products */ update
    Products
set
    supplierID=?
where
    productID=?

/* create one-to-many row com.matipl01.Supplier.products */ update
    Products
set
    supplierID=?
where
    productID=?
```

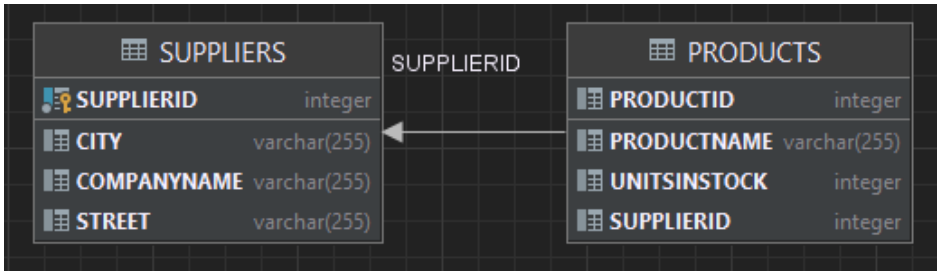
```
    /* create one-to-many row com.matipl01.Supplier.products */ update
        Products
    set
        supplierID=?
    where
        productID=?

    /*
from
    Supplier */ select
        supplier0_.supplierID as supplier1_1_,
        supplier0_.city as city2_1_,
        supplier0_.companyName as companyn3_1_,
        supplier0_.street as street4_1_
    from
        Suppliers supplier0_
```

3.3. Rezultat wykonania kodu

Taki sam jak poprzednio.

```
Dostawca 1 dostarcza Krzesło (111 szt.)
Dostawca 1 dostarcza Szafa (44 szt.)
Dostawca 1 dostarcza Komoda (53 szt.)
Dostawca 2 dostarcza Stół (23 szt.)
```

3.4. Diagram bazy danych



3.5. Utworzone tabele

3.5.1. Tabela Products

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIERID
1	1	Krzesło	111	1
2	2	Stół	23	2
3	3	Szafa	44	1
4	4	Komoda	53	1

3.5.2. Tabela Suppliers

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Poznań	Dostawca 1	Malinowa
2	2	Kraków	Dostawca 2	Konwaliowa

4. Relacja dwustronna

Łączymy poprzednie rozwiązania

4.1. Zaimplementowane klasy

4.1.1. Matoda main z klasy Main

```
public static void main(final String[] args) {
    try (Session session = getSession()) {
        Transaction tx = session.beginTransaction();

        Product product1 = new Product("Krzesło", 111);
        Product product2 = new Product("Stół", 23);
        Product product3 = new Product("Szafa", 44);
        Product product4 = new Product("Komoda", 53);

        Supplier supplier1 = new Supplier("Dostawca 1", "Malinowa", "Poznań");
        Supplier supplier2 = new Supplier("Dostawca 2", "Konwaliowa", "Kraków");

        supplier1.addProducts(product1, product3, product4);
        product1.setSupplier(supplier1);
        product3.setSupplier(supplier1);
        product4.setSupplier(supplier1);

        supplier2.addProducts(product2);
        product2.setSupplier(supplier2);

        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(product4);
        session.save(supplier1);
        session.save(supplier2);
        tx.commit();

        // Testowanie
        Query query = session.createQuery("from Supplier");
        query.getResultList().forEach(s -> {
            ((Supplier) s).getProducts().forEach(p -> System.out.println(s + "
dostarcza " + p));
        });

        query = session.createQuery("from Product");
        query.getResultList().forEach(p -> {
            System.out.println(p + " jest dostarczany/e/a przez " + ((Product)
p).getSupplier());
        });
    }
}
```

4.1.2. Klasa Supplier

```
package com.matipl01;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;

@Entity
@Table(name = "Suppliers")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"Supplier_GEN")
    @SequenceGenerator(name = "Supplier_GEN", sequenceName = "Supplier_SEQ")
```

```java
    public int supplierID;

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

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

    public Supplier() {}

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

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

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

    public void addProducts(Product ...products) {
        this.products.addAll(Arrays.asList(products));
    }
}
```

4.1.3. Klasa Product

```java
package com.matipl01;

import javax.persistence.*;

@Entity
@Table(name = "Products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Product_GEN")
    @SequenceGenerator(name = "Product_GEN", sequenceName = "Product_SEQ")
    private int productID;

    private String productName;
    private int unitsInStock;

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

    public Product() {}

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

    @Override
    public String toString() {
        return productName + " (" + unitsInStock + " szt.)";
    }

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

```
    public Supplier getSupplier() {
        return supplier;
    }
}
```

4.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
create table Products (
   productID integer not null,
    productName varchar(255),
    unitsInStock integer not null,
    supplierID integer,
    primary key (productID)
)

create table Suppliers (
   supplierID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    primary key (supplierID)
)

alter table Products
   add constraint FKbjx75exi25f1c48i92gu8rvlx
   foreign key (supplierID)
   references Suppliers

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (productName, supplierID, unitsInStock, productID)
    values
```

```
        (?, ?, ?, ?)

/* insert com.matipl01.Supplier
    */ insert
    into
        Suppliers
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)

/* insert com.matipl01.Supplier
    */ insert
    into
        Suppliers
        (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)

/* update
    com.matipl01.Product */ update
        Products
    set
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* create one-to-many row com.matipl01.Supplier.products */ update
```

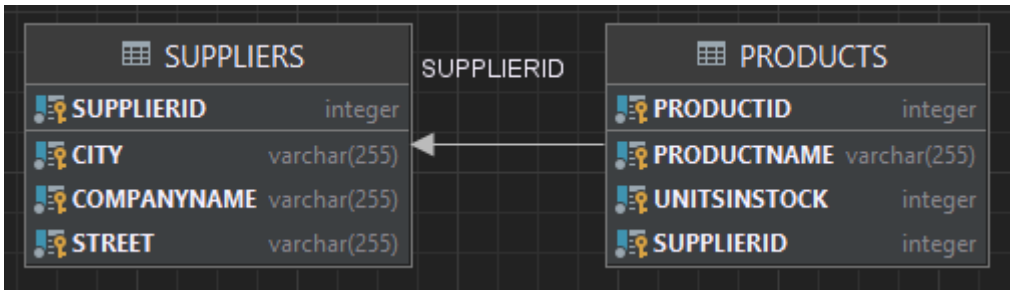
```
    Products
set
    supplierID=?
where
    productID=?

/* create one-to-many row com.matipl01.Supplier.products */ update
    Products
set
    supplierID=?
where
    productID=?

/* create one-to-many row com.matipl01.Supplier.products */ update
    Products
set
    supplierID=?
where
    productID=?

/* create one-to-many row com.matipl01.Supplier.products */ update
    Products
set
    supplierID=?
where
    productID=?

    /*
from
    Supplier */ select
        supplier0_.supplierID as supplier1_1_,
        supplier0_.city as city2_1_,
        supplier0_.companyName as companyn3_1_,
        supplier0_.street as street4_1_
    from
        Suppliers supplier0_

    /*
from
    Product */ select
        product0_.productID as producti1_0_,
        product0_.productName as productn2_0_,
        product0_.supplierID as supplier4_0_,
        product0_.unitsInStock as unitsins3_0_
    from
        Products product0_
```

4.3. Rezultat wykonania kodu

```
Dostawca 1 dostarcza Krzesło (111 szt.)
Dostawca 1 dostarcza Szafa (44 szt.)
Dostawca 1 dostarcza Komoda (53 szt.)
Dostawca 2 dostarcza Stół (23 szt.)
```

```
Krzesło (111 szt.) jest dostarczany/e/a przez Dostawca 1
Stół (23 szt.) jest dostarczany/e/a przez Dostawca 2
Szafa (44 szt.) jest dostarczany/e/a przez Dostawca 1
Komoda (53 szt.) jest dostarczany/e/a przez Dostawca 1
```

4.4. Diagram bazy danych



4.5. Utworzone tabele

4.5.1. Tabela Products

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIERID
1	1	Krzesło	111	1
2	2	Stół	23	2
3	3	Szafa	44	1
4	4	Komoda	53	1

4.5.2. Tabela Suppliers

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Poznań	Dostawca 1	Malinowa
2	2	Kraków	Dostawca 2	Konwaliowa

5. Dodanie klasy Category

Ponieważ znów konieczne jest zmodyfikowanie istniejących już produktów, ponownie zmieniłem wartość `hbm2ddl.auto` w pliku **hibarnate.cfg.xml** na `update`.

5.1. Zaimplementowane klasy

Poniżej umieściłem jedynie kod klas, które używałem podczas realizacji tego podpunktu.

5.1.1. Matoda main z klasy Main

```
public static void main(final String[] args) {
    Query query;

    try (Session session = getSession()) {
        Transaction tx = session.beginTransaction();
        Category furniture = new Category("meble");
        Category food = new Category("food");  ← wiem, że powinno być po polsku

        // Assign category to the existing products
        query = session.createQuery("from Product");
        List<Product> products = query.getResultList();
        products.forEach(furniture::addProduct);
        products.forEach(p -> p.setCategory(furniture));
```

```
            // Create new products and assign a category to them
            Product apple = new Product("Jabłko", food, 59);
            Product bread = new Product("Chleb", food, 232);
            food.addProduct(apple);
            food.addProduct(bread);

            session.save(furniture);
            session.save(food);
            session.save(apple);
            session.save(bread);

            // Tests
                // Produkty należące do kategorii
            query = session.createQuery("from Category");
            query.getResultList().forEach(c -> {
                System.out.println("Kategoria: " + c + ": ");
                ((Category) c).getProducts().forEach(p -> System.out.println("\t- " + p
+ ","));
            });
            tx.commit();

                // Kategoria, do której należy produkt
            System.out.println(apple + " należy do kategorii: " + apple.getCategory());

            if (products.size() >= 2) {
                Product product = products.get(1);
                System.out.println(product + " należy do kategorii: " +
product.getCategory());
            }
        }
```

5.1.2. Klasa Product

```
package com.matipl01;

import javax.persistence.*;

@Entity
@Table(name = "Products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Product_GEN")
    @SequenceGenerator(name = "Product_GEN", sequenceName = "Product_SEQ")
    private int productID;

    private String productName;
    private int unitsInStock;

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

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

    public Product() {}

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

    @Override
    public String toString() {
        return productName + " (" + unitsInStock + " szt.)";
    }
```

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

    public Supplier getSupplier() {
        return supplier;
    }

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

    public Category getCategory() {
        return category;
    }
}
```

5.1.3. Klasa Category

```
package com.matipl01;

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

@Entity
@Table(name = "Categories")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"Category_GEN")
    @SequenceGenerator(name = "Category_GEN", sequenceName = "Category_SEQ")
    private int categoryID;
    private String name;

    @OneToMany
    private final Collection<Product> products = new ArrayList<>();

    public Category() {}

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

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

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

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

5.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
create table Categories (
   categoryID integer not null,
    name varchar(255),
```

```
        primary key (categoryID)
    )

    alter table Products
       add column categoryID integer

    alter table Products
       add constraint FKn4dvny5ajgqgw20l5nb7imd5t
       foreign key (categoryID)
       references Categories

    /*
from
    Product */ select
        product0_.productID as producti1_1_,
        product0_.categoryID as category4_1_,
        product0_.productName as productn2_1_,
        product0_.supplierID as supplier5_1_,
        product0_.unitsInStock as unitsins3_1_
    from
        Products product0_

    select
        supplier0_.supplierID as supplier1_2_0_,
        supplier0_.city as city2_2_0_,
        supplier0_.companyName as companyn3_2_0_,
        supplier0_.street as street4_2_0_
    from
        Suppliers supplier0_
    where
        supplier0_.supplierID=?

    select
        supplier0_.supplierID as supplier1_2_0_,
        supplier0_.city as city2_2_0_,
        supplier0_.companyName as companyn3_2_0_,
        supplier0_.street as street4_2_0_
    from
        Suppliers supplier0_
    where
        supplier0_.supplierID=?

    /* insert com.matipl01.Category
        */ insert
        into
            Categories
            (name, categoryID)
        values
            (?, ?)

    /* insert com.matipl01.Category
        */ insert
        into
            Categories
            (name, categoryID)
        values
            (?, ?)

    /* insert com.matipl01.Product
```

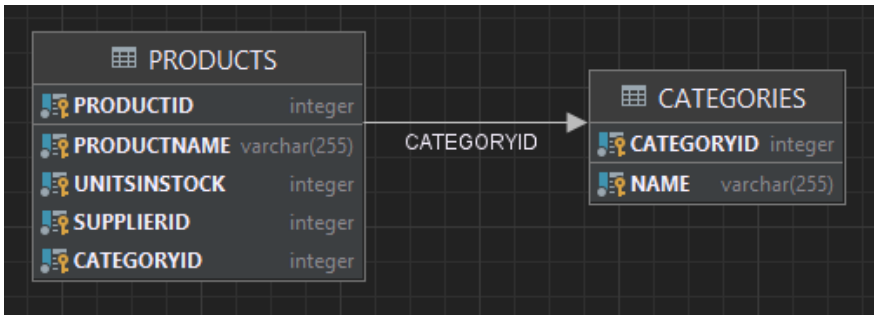
```
    */ insert
    into
        Products
        (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)

/* update
    com.matipl01.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?
```

```
    /*
from
    Category */ select
        category0_.categoryID as category1_0_,
        category0_.name as name2_0_
    from
        Categories category0_
```

5.3. Rezultat wykonania kodu

```
Kategoria: meble:
    - Krzesło (111 szt.),
    - Stół (23 szt.),
    - Szafa (44 szt.),
    - Komoda (53 szt.),
Kategoria: food:
    - Jabłko (59 szt.),
    - Chleb (232 szt.),
Jabłko (59 szt.) należy do kategorii: food
Stół (23 szt.) należy do kategorii: meble
```

5.4. Diagram bazy danych (tylko tabele z tego podpunktu)



5.5. Utworzone/zmodyfikowane tabele

5.5.1. Tabela Products

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIERID	CATEGORYID
1	1	Krzesło	111	1	102
2	2	Stół	23	2	102
3	3	Szafa	44	1	102
4	4	Komoda	53	1	102
5	52	Jabłko	59	<null>	103
6	53	Chleb	232	<null>	103

5.5.2. Tabela Categories

	CATEGORYID	NAME
1	102	meble
2	103	food

6. Relacja wiele do wielu (dodanie faktury)

6.1. Zaimplementowane klasy

6.1.1. Matoda main z klasy Main

```
public static void main(final String[] args) throws InvalidAttributeValueException
{
    Query query;

    try (Session session = getSession()) {
        Transaction tx = session.beginTransaction();

        // Create new products and assign a category to them
        Category electronics = new Category("elektronika");
        Product smartphone = new Product("Smartfon", electronics, 321);
        Product tablet = new Product("Tablet", electronics, 123);
        electronics.addProduct(smartphone);
        electronics.addProduct(tablet);

        // Sell existing products
        Invoice invoice1 = new Invoice();
        int soldCount = 55;
        query = session.createQuery("from Product where unitsInStock >" +
soldCount);
        List<Product> products = query.getResultList();

        products.forEach(p -> {
            try {
                p.sell(invoice1, soldCount);
            } catch (InvalidAttributeValueException e) {
                e.printStackTrace();
            }
        });

        // Sell new products
        smartphone.sell(invoice1, 3);

        Invoice invoice2 = new Invoice();
        smartphone.sell(invoice2, 2);
        smartphone.sell(invoice2, 1);

        session.save(invoice1);
        session.save(invoice2);
        session.save(electronics);
        session.save(smartphone);
        session.save(tablet);

        tx.commit();

        // Tests
        // Get the list of products sold in the specific invoice (I'm checking all
invoices)
        query = session.createQuery("from Invoice");
        query.getResultList().forEach(i -> {
            Invoice invoice = ((Invoice) i);
            List<Product> products_ = (List<Product>) invoice.getProducts();
            System.out.println("\nFaktura numer: " + i + ": ");
            System.out.println("\tŁączna liczba produktów: " +
invoice.getQuantity());
            products_.forEach(p -> System.out.println("\t- " + p.getName() + ","));
            if (products_.size() == 0) System.out.println("\tBrak");
        });

        // Get all invoices for a product
        query = session.createQuery("from Product");
```

```
            query.getResultList().forEach(p -> {
                Product product = ((Product) p);
                List<Invoice> invoices = (List<Invoice>) product.getInvoices();
                System.out.println("\nFaktury, na których występuje produkt: " +
product.getName() + ": ");
                invoices.forEach(i -> System.out.println("\t- " + i + ","));
                if (invoices.size() == 0) System.out.println("\tBrak");
            });
        }
    }
```

6.1.2. Klasa Product

```
package com.matipl01;

import javax.management.InvalidAttributeValueException;
import javax.persistence.*;
import java.util.Collection;
import java.util.HashSet;

@Entity
@Table(name = "Products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Product_GEN")
    @SequenceGenerator(name = "Product_GEN", sequenceName = "Product_SEQ")
    private int productID;

    private String productName;
    private int unitsInStock;

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

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

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

    public Product() {}

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

    @Override
    public String toString() {
        return productName + " (" + unitsInStock + " szt.)";
    }

    public String getName() {
        return productName;
    }

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

    public Supplier getSupplier() {
        return supplier;
    }

    public void setCategory(Category category) {
```

```
        this.category = category;
    }

    public Category getCategory() {
        return category;
    }

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

    public void sell(Invoice invoice, int quantity) throws
InvalidAttributeValueException {
        if (unitsInStock < quantity) {
            throw new InvalidAttributeValueException("Unable to sell " + quantity
+ " products");
        }
        unitsInStock -= quantity;
        invoice.addProduct(this, quantity);
        invoices.add(invoice);
    }
}
```

6.1.3. Klasa Invoice

```
package com.matipl01;

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

@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Invoice_GEN")
    @SequenceGenerator(name = "Invoice_GEN", sequenceName = "Invoice_SEQ")
    private int invoiceNumber;

    private int quantity = 0;

    @ManyToMany
    @JoinColumn(name = "productID")
    private Collection<Product> products = new HashSet<>();

    public Invoice() {}

    @Override
    public String toString() {
        return String.valueOf(invoiceNumber);
    }

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

    public int getInvoiceNumber() {
        return invoiceNumber;
    }

    public int getQuantity() {
        return quantity;
    }

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

6.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
    create table Invoice (
       invoiceNumber integer not null,
        quantity integer not null,
        primary key (invoiceNumber)
    )

    create table Invoice_Products (
       invoices_invoiceNumber integer not null,
        products_productID integer not null
    )

    alter table Invoice_Products
       add constraint FKs9kojcr3iu3dm7fww8n0v442n
       foreign key (products_productID)
       references Products

    alter table Invoice_Products
       add constraint FKlgae5neonlp88wdmxtb6qdppw
       foreign key (invoices_invoiceNumber)
       references Invoice

    /*
from
    Product
where
    unitsInStock >55 */ select
        product0_.productID as producti1_3_,
        product0_.categoryID as category4_3_,
        product0_.productName as productn2_3_,
        product0_.supplierID as supplier5_3_,
        product0_.unitsInStock as unitsins3_3_
    from
        Products product0_
    where
        product0_.unitsInStock>55

    select
        category0_.categoryID as category1_0_0_,
        category0_.name as name2_0_0_
    from
        Categories category0_
    where
        category0_.categoryID=?

    select
        supplier0_.supplierID as supplier1_4_0_,
        supplier0_.city as city2_4_0_,
        supplier0_.companyName as companyn3_4_0_,
        supplier0_.street as street4_4_0_
    from
        Suppliers supplier0_
    where
        supplier0_.supplierID=?
```

```
select
    category0_.categoryID as category1_0_0_,
    category0_.name as name2_0_0_
from
    Categories category0_
where
    category0_.categoryID=?

/* insert com.matipl01.Invoice
    */ insert
    into
        Invoice
        (quantity, invoiceNumber)
    values
        (?, ?)

/* insert com.matipl01.Invoice
    */ insert
    into
        Invoice
        (quantity, invoiceNumber)
    values
        (?, ?)

/* insert com.matipl01.Invoice
    */ insert
    into
        Invoice
        (quantity, invoiceNumber)
    values
        (?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)

/* insert com.matipl01.Product
    */ insert
    into
        Products
        (categoryID, productName, supplierID, unitsInStock, productID)
    values
        (?, ?, ?, ?, ?)

/* update
    com.matipl01.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
```

```
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* update
    com.matipl01.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

/* insert collection
    row com.matipl01.Invoice.products */ insert
    into
        Invoice_Products
        (invoices_invoiceNumber, products_productID)
    values
        (?, ?)

/* insert collection
    row com.matipl01.Invoice.products */ insert
    into
        Invoice_Products
        (invoices_invoiceNumber, products_productID)
    values
        (?, ?)

/* insert collection
    row com.matipl01.Invoice.products */ insert
    into
        Invoice_Products
        (invoices_invoiceNumber, products_productID)
    values
        (?, ?)

/* insert collection
    row com.matipl01.Invoice.products */ insert
    into
        Invoice_Products
        (invoices_invoiceNumber, products_productID)
    values
        (?, ?)

/* insert collection
    row com.matipl01.Invoice.products */ insert
    into
```

```
            Invoice_Products
            (invoices_invoiceNumber, products_productID)
        values
            (?, ?)

    /*
from
    Invoice */ select
        invoice0_.invoiceNumber as invoicen1_1_,
        invoice0_.quantity as quantity2_1_
    from
        Invoice invoice0_

    /*
from
    Product */ select
        product0_.productID as producti1_3_,
        product0_.categoryID as category4_3_,
        product0_.productName as productn2_3_,
        product0_.supplierID as supplier5_3_,
        product0_.unitsInStock as unitsins3_3_
    from
        Products product0_

    select
        supplier0_.supplierID as supplier1_4_0_,
        supplier0_.city as city2_4_0_,
        supplier0_.companyName as companyn3_4_0_,
        supplier0_.street as street4_4_0_
    from
        Suppliers supplier0_
    where
        supplier0_.supplierID=?

    select
        invoices0_.products_productID as products2_2_0_,
        invoices0_.invoices_invoiceNumber as invoices1_2_0_,
        invoice1_.invoiceNumber as invoicen1_1_1_,
        invoice1_.quantity as quantity2_1_1_
    from
        Invoice_Products invoices0_
    inner join
        Invoice invoice1_
            on invoices0_.invoices_invoiceNumber=invoice1_.invoiceNumber
    where
        invoices0_.products_productID=?

    select
        invoices0_.products_productID as products2_2_0_,
        invoices0_.invoices_invoiceNumber as invoices1_2_0_,
        invoice1_.invoiceNumber as invoicen1_1_1_,
        invoice1_.quantity as quantity2_1_1_
    from
        Invoice_Products invoices0_
    inner join
        Invoice invoice1_
            on invoices0_.invoices_invoiceNumber=invoice1_.invoiceNumber
    where
        invoices0_.products_productID=?
```

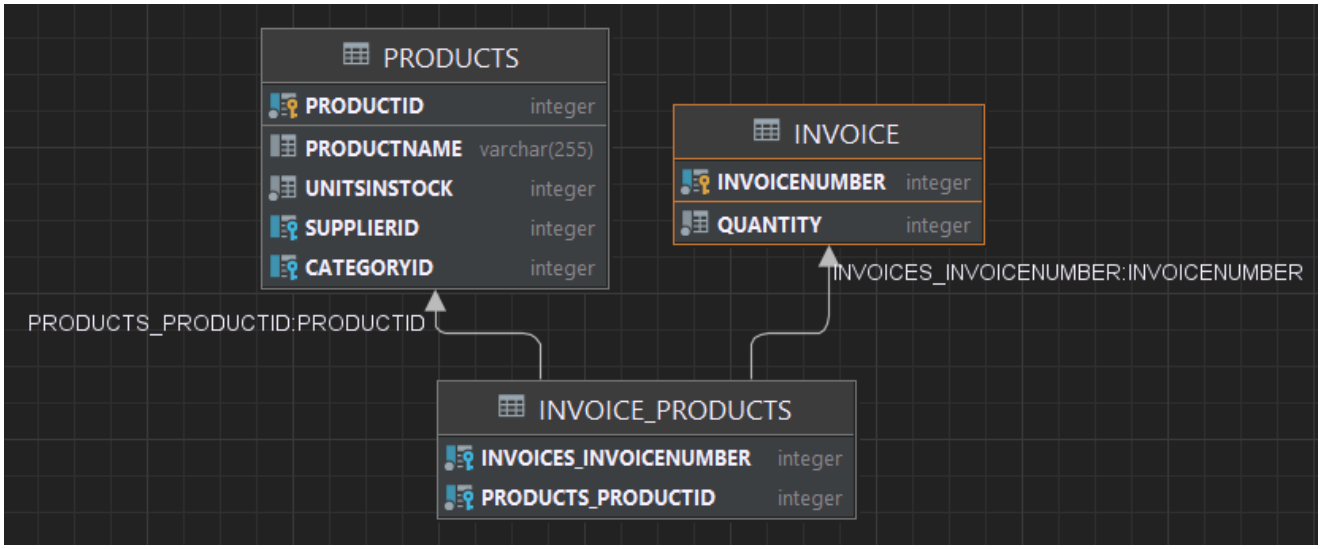
```
select
    invoices0_.products_productID as products2_2_0_,
    invoices0_.invoices_invoiceNumber as invoices1_2_0_,
    invoice1_.invoiceNumber as invoicen1_1_1_,
    invoice1_.quantity as quantity2_1_1_
from
    Invoice_Products invoices0_
inner join
    Invoice invoice1_
        on invoices0_.invoices_invoiceNumber=invoice1_.invoiceNumber
where
    invoices0_.products_productID=?

select
    invoices0_.products_productID as products2_2_0_,
    invoices0_.invoices_invoiceNumber as invoices1_2_0_,
    invoice1_.invoiceNumber as invoicen1_1_1_,
    invoice1_.quantity as quantity2_1_1_
from
    Invoice_Products invoices0_
inner join
    Invoice invoice1_
        on invoices0_.invoices_invoiceNumber=invoice1_.invoiceNumber
where
    invoices0_.products_productID=?

select
    invoices0_.products_productID as products2_2_0_,
    invoices0_.invoices_invoiceNumber as invoices1_2_0_,
    invoice1_.invoiceNumber as invoicen1_1_1_,
    invoice1_.quantity as quantity2_1_1_
from
    Invoice_Products invoices0_
inner join
    Invoice invoice1_
        on invoices0_.invoices_invoiceNumber=invoice1_.invoiceNumber
where
    invoices0_.products_productID=?

select
    invoices0_.products_productID as products2_2_0_,
    invoices0_.invoices_invoiceNumber as invoices1_2_0_,
    invoice1_.invoiceNumber as invoicen1_1_1_,
    invoice1_.quantity as quantity2_1_1_
from
    Invoice_Products invoices0_
inner join
    Invoice invoice1_
        on invoices0_.invoices_invoiceNumber=invoice1_.invoiceNumber
where
    invoices0_.products_productID=?
```

6.3. Rezultat wykonania kodu

Aby rezultat był lepiej widoczny, zmieniłem w pliku konfiguracyjnym wartość `show_sql` z na **false**. Wówczas, otrzymujemy na konsoli poniższy rezultat:

```
Faktura numer: 102:
    Łączna liczba produktów: 168
    - Smartfon,
    - Krzesło,
    - Jabłko,
    - Chleb,

Faktura numer: 103:
    Łączna liczba produktów: 3
    - Smartfon,

Faktury, na których występuje produkt: Krzesło:
    - 102,

Faktury, na których występuje produkt: Stół:
    Brak

Faktury, na których występuje produkt: Szafa:
    Brak

Faktury, na których występuje produkt: Komoda:
    Brak

Faktury, na których występuje produkt: Jabłko:
    - 102,

Faktury, na których występuje produkt: Chleb:
    - 102,

Faktury, na których występuje produkt: Smartfon:
    - 102,
    - 103,

Faktury, na których występuje produkt: Tablet:
    Brak
```

6.4. Diagram bazy danych (tylko tabele z tego podpunktu)



6.5. Utworzone/zmodyfikowane tabele

6.5.1. Tabela Products

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIERID	CATEGORYID
1	1	Krzesło	56	1	302
2	2	Stół	23	2	302
3	3	Szafa	44	1	302
4	4	Komoda	53	1	302
5	52	Jabłko	4	<null>	303
6	53	Chleb	177	<null>	303
7	102	Smartfon	315	<null>	352
8	103	Tablet	123	<null>	352

6.5.2. Tabela Categories

	CATEGORYID	NAME
1	302	meble
2	303	food
3	352	elektronika

JPA

7. Relacja wiele do wielu (faktura)

7.1. Zaimplementowane klasy

7.1.1. Klasa Main

```
package com.matipl01;

import javax.management.InvalidAttributeValueException;
import javax.persistence.*;
import java.util.List;

class Main {
    private static final EntityManagerFactory emf;

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

    public static EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

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

        // Add products only
        etx.begin();
        Product product1 = new Product("Smartfon", 25);
        Product product2 = new Product("Tablet", 45);
        Product product3 = new Product("Konsola", 11);
        em.persist(product1);
        em.persist(product2);
        em.persist(product3);
        etx.commit();

        etx.begin();
        // Create invoices
        Invoice invoice1 = new Invoice();
        Invoice invoice2 = new Invoice();

        // Sell existing products
        int soldCount = 20;
        query = em.createQuery("from Product where unitsInStock >" + soldCount);
        List<Product> products = query.getResultList();
        products.forEach(p -> {
            if (((Product) p).getUnitsInStock() >= soldCount) {
                try {
                    ((Product) p).sell(invoice1, soldCount);
                } catch (InvalidAttributeValueException e) {
                    e.printStackTrace();
                }
            }
```

```
        });

        // Create the new products and sell them
        product1 = new Product("Mikrofalówka", 4);
        product2 = new Product("Lodówka", 14);
        product3 = new Product("Wirówka", 17);

        try {
            product1.sell(invoice2, 3);
            product2.sell(invoice1, 11);
            product2.sell(invoice2, 2);
            product3.sell(invoice2, 17);
        } catch (InvalidAttributeValueException e) {
            e.printStackTrace();
        }

        em.persist(invoice1);
        em.persist(invoice2);
        etx.commit();

        // Tests
        // Get the list of products sold in the specific invoice (I'm checking all
invoices)
        query = em.createQuery("from Invoice");
        query.getResultList().forEach(i -> {
            Invoice invoice = ((Invoice) i);
            List<Product> products_ = (List<Product>) invoice.getProducts();
            System.out.println("\nFaktura numer: " + i + ": ");
            System.out.println("\tŁączna liczba produktów: " +
invoice.getQuantity());
            products_.forEach(p -> System.out.println("\t- " + p.getName() +
","));
            if (products_.size() == 0) System.out.println("\tBrak");
        });

        // Get all invoices for a product
        query = em.createQuery("from Product");
        query.getResultList().forEach(p -> {
            Product product = ((Product) p);
            List<Invoice> invoices = (List<Invoice>) product.getInvoices();
            System.out.println("\nFaktury, na których występuje produkt: " +
product.getName() + ": ");
            invoices.forEach(i -> System.out.println("\t- " + i + ","));
            if (invoices.size() == 0) System.out.println("\tBrak");
        });

        em.close();
    }
}
```

7.1.2. Klasa Product

```
package com.matipl01;

import javax.management.InvalidAttributeValueException;
import javax.persistence.*;
import java.util.Collection;
import java.util.HashSet;

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

    private String productName;
    private int unitsInStock;
```

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

    public Product() {}

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

    @Override
    public String toString() {
        return productName + " (" + unitsInStock + " szt.)";
    }

    public String getName() {
        return productName;
    }

    public int getUnitsInStock() {
        return unitsInStock;
    }

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

    public void sell(Invoice invoice, int quantity) throws
InvalidAttributeValueException {
        if (unitsInStock < quantity) {
            throw new InvalidAttributeValueException("Unable to sell " + quantity
+ " products");
        }
        unitsInStock -= quantity;
        invoice.addProduct(this, quantity);
        invoices.add(invoice);
    }
}
```

7.1.3. Klasa Invoice

```
package com.matipl01;

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

@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Invoice_GEN")
    @SequenceGenerator(name = "Invoice_GEN", sequenceName = "Invoice_SEQ")
    private int invoiceNumber;

    private int quantity = 0;

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

    public Invoice() {}

    @Override
    public String toString() {
        return String.valueOf(invoiceNumber);
    }

    public void addProduct(Product product, int quantity) {
        this.products.add(product);
```

```
        this.quantity += quantity;
    }

    public int getInvoiceNumber() {
        return invoiceNumber;
    }

    public int getQuantity() {
        return quantity;
    }

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

7.2. Plik konfiguracyjny persistence.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
 http://java.sun.com/xml/ns/persistence/persistence_2)0.xsd" version="2.0">

    <persistence-unit name="derby" transaction-type="RESOURCE_LOCAL">
        <properties>
            <property name="hibernate.dialect"
value="org.hibernate.dialect.DerbyTenSevenDialect" />
            <property name="hibernate.connection.driver_class"
                      value="org.apache.derby.jdbc.ClientDriver"/>
            <property name="hibernate.connection.url"
                      value="jdbc:derby://127.0.0.1/LopacinskiMateuszJPA"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
        </properties>
    </persistence-unit>
</persistence>
```

7.3. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
    create table Invoice (
       invoiceNumber integer not null,
        quantity integer not null,
        primary key (invoiceNumber)
    )

    create table Invoice_Product (
       invoices_invoiceNumber integer not null,
        products_productID integer not null
    )

    create table Product (
       productID integer not null,
        productName varchar(255),
        unitsInStock integer not null,
        primary key (productID)
    )

    alter table Invoice_Product
       add constraint FK2mn08nt19nrqagr12grh5uho0
       foreign key (products_productID)
       references Product
```

```
alter table Invoice_Product
   add constraint FKcbqyl9u4eh1tws13u6pk5j2nt
   foreign key (invoices_invoiceNumber)
   references Invoice

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

select
    product0_.productID as producti1_2_,
    product0_.productName as productn2_2_,
    product0_.unitsInStock as unitsins3_2_
from
    Product product0_
where
    product0_.unitsInStock>20

insert
into
    Invoice
    (quantity, invoiceNumber)
values
    (?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

insert
into
    Invoice
    (quantity, invoiceNumber)
values
    (?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
```

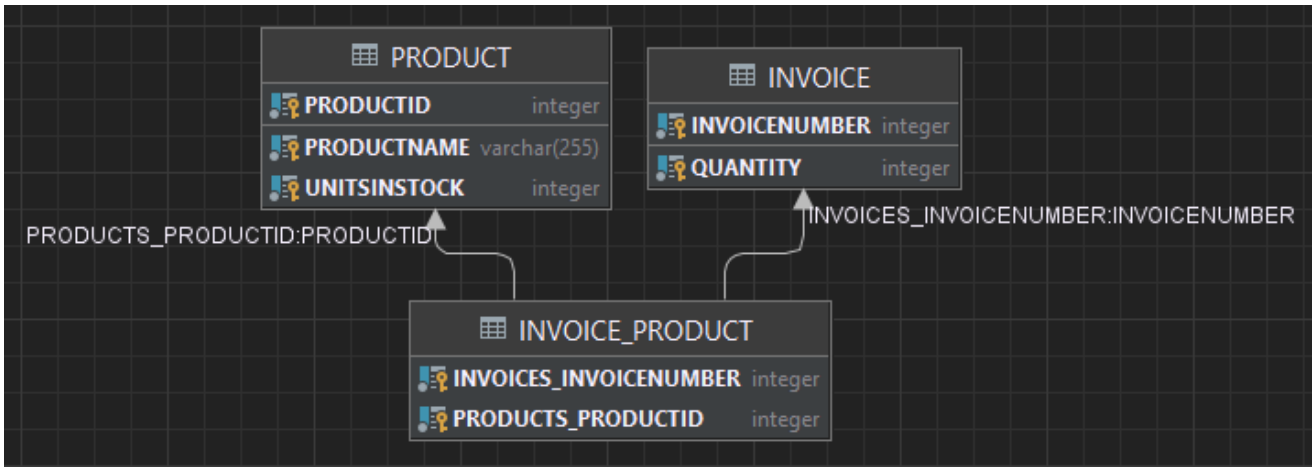
```
values
    (?, ?, ?)

update
    Product
set
    productName=?,
    unitsInStock=?
where
    productID=?

update
    Product
set
    productName=?,
    unitsInStock=?
where
    productID=?

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

select
    invoice0_.invoiceNumber as invoicen1_0_,
    invoice0_.quantity as quantity2_0_
from
    Invoice invoice0_
```

7.4. Rezultat wykonania kodu

```
Faktura numer: 1:
    Łączna liczba produktów: 51
    - Lodówka,
    - Tablet,
    - Smartfon,

Faktura numer: 2:
    Łączna liczba produktów: 22
    - Wirówka,
    - Mikrofalówka,
    - Lodówka,
```

```
Faktury, na których występuje produkt: Smartfon:
    - 1,

Faktury, na których występuje produkt: Tablet:
    - 1,

Faktury, na których występuje produkt: Konsola:
    Brak

Faktury, na których występuje produkt: Lodówka:
    - 2,
    - 1,

Faktury, na których występuje produkt: Wirówka:
    - 2,

Faktury, na których występuje produkt: Mikrofalówka:
    - 2,
```

7.5. Diagram bazy danych



7.6. Utworzone/zmodyfikowane tabele

7.6.1. Tabela Product

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK
1	1	Smartfon	5
2	2	Tablet	25
3	3	Konsola	11
4	4	Lodówka	1
5	5	Wirówka	0
6	6	Mikrofalówka	1

7.6.2. Tabela Invoice

	INVOICENUMBER	QUANTITY
1	1	51
2	2	22

7.6.3. Tabela Invoice_Product

	INVOICES_INVOICENUMBER	PRODUCTS_PRODUCTID
1	1	4
2	1	2
3	1	1
4	2	5
5	2	6
6	2	4

8. Kaskady

8.1. Zaimplementowane klasy

8.1.1. Metoda main z klasy Main

Ponownie zmieniona została tylko metoda `public static void main(String[] args)`, dlatego poniżej umieszczam tylko kod tej metody.

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

    // Create products
    Product product1 = new Product("Smartfon", 25);
    Product product2 = new Product("Tablet", 45);
    Product product3 = new Product("Konsola", 11);
    Product product4 = new Product("Smartwatch", 32);

    // Create invoices
    Invoice invoice1 = new Invoice();
    Invoice invoice2 = new Invoice();

    // Add products to invoices
    etx.begin();
    invoice1.addProduct(product1, 5);
    invoice2.addProduct(product3, 7);
```

```
    invoice1.addProduct(product2, 8);
    invoice1.addProduct(product3, 7);
    em.persist(invoice1);
    em.persist(invoice2);
    etx.commit();

    // Add invoices to products
    etx.begin();
    product1.sell(invoice1, 11);
    product2.sell(invoice2, 12);
    product3.sell(invoice2, 3);
    product4.sell(invoice1, 7);
    em.persist(product1);
    em.persist(product2);
    em.persist(product3);
    em.persist(product4);
    etx.commit();
    em.close();
}
```

8.1.2. Klasa Product

Zmiana dotyczyła jedynie dekoratora `@ManyToMany` oraz metody `public void sell(Invoice invoice, int quantity)`.

```
package com.matipl01;

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

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"Product_GEN")
    @SequenceGenerator(name = "Product_GEN", sequenceName = "Product_SEQ")
    private int productID;

    private String productName;
    private int unitsInStock;

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

    public Product() {}

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

    @Override
    public String toString() {
        return productName + " (" + unitsInStock + " szt.)";
    }

    public String getName() {
        return productName;
    }

    public Collection<Invoice> getInvoices() {
```

```
        return invoices;
    }

    public int getUnitsInStock() {
        return unitsInStock;
    }

    public void sell(Invoice invoice, int quantity) throws
IllegalArgumentException {
        if (unitsInStock < quantity) {
            throw new IllegalArgumentException("Unable to sell " + quantity + "
products");
        }
        invoices.add(invoice);
        invoice.updateQuantity(quantity);
    }
}
```

8.1.3. Klasa Invoice

Zmiana dotyczyła jedynie dekoratora `@ManyToMany`.

```
package com.matipl01;

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

@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"Invoice_GEN")
    @SequenceGenerator(name = "Invoice_GEN", sequenceName = "Invoice_SEQ")
    private int invoiceNumber;

    private int quantity = 0;

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

    public Invoice() {}

    @Override
    public String toString() {
        return String.valueOf(invoiceNumber);
    }

    public void addProduct(Product product, int quantity) throws
IllegalArgumentException {
        if (product.getUnitsInStock() < quantity) {
            throw new IllegalArgumentException("Unable to sell " + quantity + "
products");
        }
        this.products.add(product);
        this.quantity += quantity;
    }

    public int getQuantity() {
        return quantity;
    }

    public void updateQuantity(int quantity) {
        this.quantity += quantity;
    }

    public Collection<Product> getProducts() {
        return products;
```

```
        }
    }
```

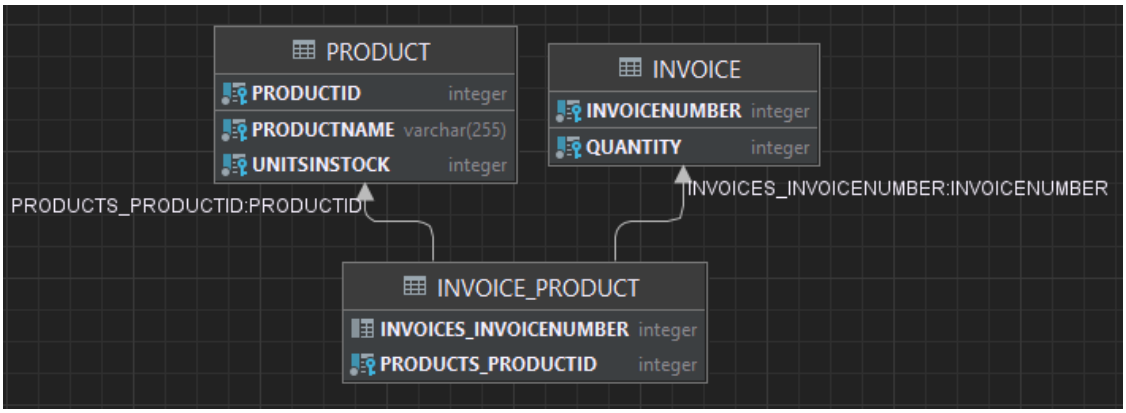
8.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
create table Invoice (
   invoiceNumber integer not null,
    quantity integer not null,
    primary key (invoiceNumber)
)

create table Invoice_Product (
   invoices_invoiceNumber integer not null,
    products_productID integer not null
)

create table Product (
   productID integer not null,
    productName varchar(255),
    unitsInStock integer not null,
    primary key (productID)
)

alter table Invoice_Product
   add constraint FK2mn08nt19nrqagr12grh5uho0
   foreign key (products_productID)
   references Product

alter table Invoice_Product
   add constraint FKcbqyl9u4eh1tws13u6pk5j2nt
   foreign key (invoices_invoiceNumber)
   references Invoice

insert
into
    Invoice
    (quantity, invoiceNumber)
values
    (?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

insert
into
    Invoice
    (quantity, invoiceNumber)
values
```

```
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Invoice_Product
    (invoices_invoiceNumber, products_productID)
values
    (?, ?)

insert
into
    Product
    (productName, unitsInStock, productID)
values
    (?, ?, ?)

update
    Invoice
set
    quantity=?
where
    invoiceNumber=?

update
    Invoice
set
    quantity=?
where
    invoiceNumber=?
```

8.3. Diagram bazy danych



8.4. Utworzone/zmodyfikowane tabele

8.4.1. Tabela Product

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK
1	1	Tablet	45
2	2	Smartfon	25
3	3	Konsola	11
4	4	Smartwatch	32

8.4.2. Tabela Invoice

	INVOICENUMBER	QUANTITY
1	1	38
2	2	22

8.4.3. Tabela Invoice_Product

	INVOICES_INVOICENUMBER	PRODUCTS_PRODUCTID
1	1	1
2	1	2
3	1	3
4	2	3

9. Embedded class

9a. Adresy wbudowane w tabelę dostawców

9.1. Zaimplementowane klasy

9.1.1. Metoda main z klasy Main

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

    // Create Suppliers
        Supplier supplier1 = new Supplier("Owocowy Raj", new Address("Kraków",
"Miodowa"));
    Supplier supplier2 = new Supplier("Elektro", new Address("Warszawa", "Długa"));

    // Save suppliers
    etx.begin();
    em.persist(supplier1);
    em.persist(supplier2);
    etx.commit();
    em.close();
}
```

9.1.2. Klasa Supplier

```
package com.matipl01;

import javax.persistence.*;

@Entity
public class Supplier {
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"Supplier_GEN")
    @SequenceGenerator(name = "Supplier_GEN", sequenceName = "Supplier_SEQ")
    public int supplierID;

    private String companyName;

    @Embedded
    private Address address;

    public Supplier() {}

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

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

    public void setAddress(Address address) {
        this.address = address;
    }

    public Address getAddress() {
        return address;
    }
}
```

9.1.3. Klasa Address

```
package com.matipl01;

import javax.persistence.Embeddable;

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

    public Address() {}

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

    @Override
    public String toString() {
        return city + ", ul. " + street;
    }

    public String getStreet() {
        return street;
    }

    public String getCity() {
        return city;
    }

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

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

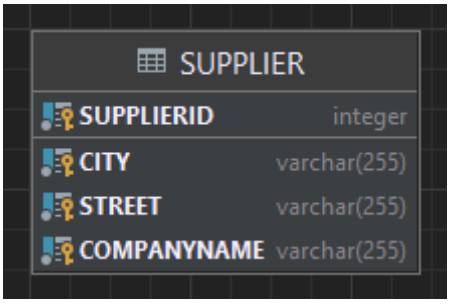
```
    }
}
```

9.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
create table Supplier (
   supplierID integer not null,
    city varchar(255),
    street varchar(255),
    companyName varchar(255),
    primary key (supplierID)
)

insert
into
    Supplier
    (city, street, companyName, supplierID)
values
    (?, ?, ?, ?)

insert
into
    Supplier
    (city, street, companyName, supplierID)
values
    (?, ?, ?, ?)
```

9.3. Diagram bazy danych



9.4. Utworzone/zmodyfikowane tabele

9.4.1. Tabela Supplier

	SUPPLIERID	CITY	STREET	COMPANYNAME
1	1	Miodowa	Kraków	Owocowy Raj
2	2	Długa	Warszawa	Elektro

9b. Adresy w osobnej tabeli

9.1. Zaimplementowane klasy

9.1.1. Metoda main z klasy Main

Identyczna jak poprzednio.

9.1.2. Klasa Supplier

```
package com.matipl01;

import javax.persistence.*;
```

```java
@Entity
@SecondaryTable(name = "Address")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"Supplier_GEN")
    @SequenceGenerator(name = "Supplier_GEN", sequenceName = "Supplier_SEQ")
    public int supplierID;

    private String companyName;

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

    public Supplier() {}

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

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

    public String getCity() {
        return city;
    }

    public String getStreet() {
        return street;
    }
}
```

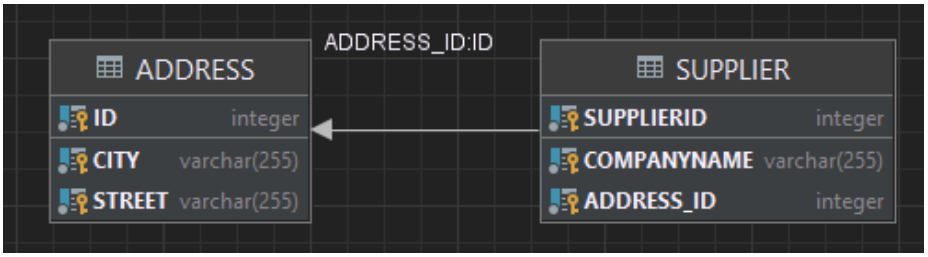
9.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```sql
create table Address (
    id integer not null,
     city varchar(255),
     street varchar(255),
     primary key (id)
);

create table Supplier (
    supplierID integer not null,
     companyName varchar(255),
     address_id integer,
     primary key (supplierID)
);

alter table Supplier
     add constraint FKcbqyl9u4eh1tws13u6pk5j2nt
     foreign key (address_id)
     references Address

insert
    into
        Address
        (city, street, id)
    values
        (?, ?, ?)
```

```
insert
    into
        Supplier
        (companyName, address_id, supplierID)
    values
        (?, ?, ?)

insert
    into
        Address
        (city, street, id)
    values
        (?, ?, ?)

insert
    into
        Supplier
        (companyName, address_id, supplierID)
    values
        (?, ?, ?)
```

9.3. Diagram bazy danych



9.4. Utworzone/zmodyfikowane tabele

9.4.1. Tabela Supplier

	SUPPLIERID	COMPANYNAME	ADDRESS_ID
1	1	Owocowy Raj	1
2	2	Elektro	2

9.4.2. Tabela Address

	ID	CITY	STREET
1	1	Kraków	Miodowa
2	2	Warszawa	Długa

10. Dziedziczenie

10a. Type Per Class

10.1. Zaimplementowane klasy

10.1.1. Metoda main z klasy Main

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

    // Save suppliers
```

```
        etx.begin();

        Customer customer1 = new Customer("Klient 1", "Hofmana Vlastimila", "Kraków",
"30-210", 5.5);
        Customer customer2 = new Customer("Klient 2", "3 Maja Al.", "Kraków", "30-063",
9.75);

        Supplier supplier1 = new Supplier("Dostawca 1", "Mikołaja Kopernika 3",
"Warszawa", "00-367", "123123123123123123123123");
        Supplier supplier2 = new Supplier("Dostawca 2", "Oboźna", "Kraków", "30-011",
"999888777666555444333222");

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

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

10.1.2. Klasa Company

```
package com.matipl01;

import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "Company_GEN")
    @SequenceGenerator(name = "Company_GEN", sequenceName = "Company_SEQ")
    private int companyID;

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

    public Company() {}

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

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

    public String getCompanyName() {
        return companyName;
    }

    public String getStreet() {
        return street;
    }

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

```
    public String getZipCode() {
        return zipCode;
    }
}
```

10.1.3. Klasa Supplier

```
package com.matipl01;

import javax.persistence.Entity;

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

    public Supplier() {}

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

10.1.4. Klasa Customer

```
package com.matipl01;

import javax.persistence.Entity;

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

    public Customer() {}

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

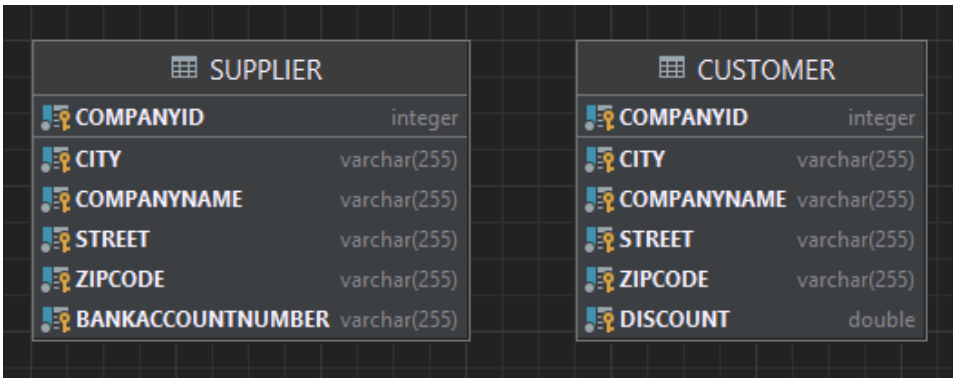
10.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
create table Customer (
   companyID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    zipCode varchar(255),
    discount double not null,
    primary key (companyID)
)

create table Supplier (
   companyID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    zipCode varchar(255),
    bankAccountNumber varchar(255),
    primary key (companyID)
)
```

```
insert
into
    Customer
    (city, companyName, street, zipCode, discount, companyID)
values
    (?, ?, ?, ?, ?, ?)

insert
into
    Customer
    (city, companyName, street, zipCode, discount, companyID)
values
    (?, ?, ?, ?, ?, ?)

insert
into
    Supplier
    (city, companyName, street, zipCode, bankAccountNumber, companyID)
values
    (?, ?, ?, ?, ?, ?)

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

10.3. Diagram bazy danych



10.4. Utworzone/zmodyfikowane tabele

10.4.1. Tabela Customer

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT
1	1	Kraków	Klient 1	Hofmana Vlastimila	30-210	5.5
2	2	Kraków	Klient 2	3 Maja Al.	30-063	9.75

10.4.2. Tabela Supplier

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	3	Warszawa	Dostawca 1	Mikołaja Kopernika 3	00-367	12312312312312312312312 3
2	4	Kraków	Dostawca 2	Oboźna	30-011	999888777666555444333222

10b. Single Table

10.1. Zaimplementowane klasy

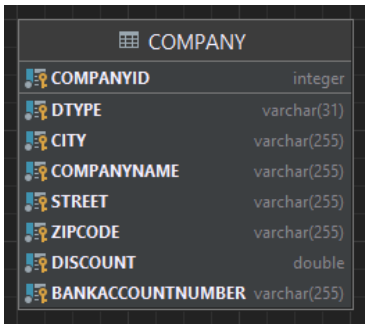
Takie same, jak poprzednio. Zmieniony został jedynie dekorator klasy `public abstract class Company` na `@Inheritance(strategy = InheritanceType.SINGLE_TABLE)`.

10.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

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

insert
into
    Company
    (city, companyName, street, zipCode, discount, DTYPE, companyID)
values
    (?, ?, ?, ?, ?, 'Customer', ?)

insert
into
    Company
    (city, companyName, street, zipCode, discount, DTYPE, companyID)
values
    (?, ?, ?, ?, ?, 'Customer', ?)

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

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

10.3. Diagram bazy danych



10.4. Utworzone/zmodyfikowane tabele

10.4.1. Tabela Company

	DTYPE	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT	BANKACCOUNTNUMBER
1	Customer	1	Kraków	Klient 1	Hofmana Vlastimila	30-210	5.5	<null>
2	Customer	2	Kraków	Klient 2	3 Maja Al.	30-063	9.75	<null>
3	Supplier	3	Warszawa	Dostawca 1	Mikołaja Kopernika 3	00-367	<null>	123123123123123123123123
4	Supplier	4	Kraków	Dostawca 2	Oboźna	30-011	<null>	999888777666555444333222

10c. Joined

10.1. Zaimplementowane klasy

Takie same, jak poprzednio. Zmieniony został jedynie dekorator klasy `public abstract class Company` na `@Inheritance(strategy = InheritanceType.JOINED)`.

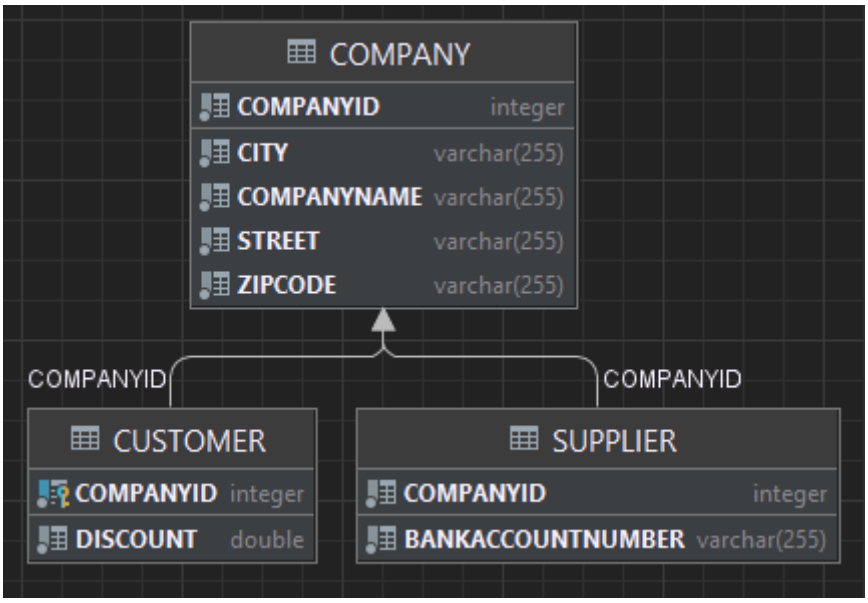
10.2. Logi SQL

Pomijam **DROP TABLE** oraz kod odpowiedzialny za tworzenie i korzystanie z sekwencji.

```
create table Company (
   companyID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    zipCode varchar(255),
    primary key (companyID)
)

create table Customer (
   discount double not null,
    companyID integer not null,
    primary key (companyID)
)

create table Supplier (
   bankAccountNumber varchar(255),
    companyID integer not null,
    primary key (companyID)
)

alter table Customer
   add constraint FKn7fvr687iixps0s6i5casr6f3
   foreign key (companyID)
   references Company

alter table Supplier
   add constraint FKpinunrb4v5p4aemt2k4fnkjp8
   foreign key (companyID)
   references Company

insert
into
    Company
    (city, companyName, street, zipCode, companyID)
values
    (?, ?, ?, ?, ?)

insert
into
    Customer
    (discount, companyID)
values
    (?, ?)

insert
into
    Company
    (city, companyName, street, zipCode, companyID)
values
    (?, ?, ?, ?, ?)

insert
into
```

```
    Customer
    (discount, companyID)
values
    (?, ?)

insert
into
    Company
    (city, companyName, street, zipCode, companyID)
values
    (?, ?, ?, ?, ?)

insert
into
    Supplier
    (bankAccountNumber, companyID)
values
    (?, ?)

insert
into
    Company
    (city, companyName, street, zipCode, companyID)
values
    (?, ?, ?, ?, ?)

insert
into
    Supplier
    (bankAccountNumber, companyID)
values
    (?, ?)
```

10.3. Diagram bazy danych



10.4. Utworzone/zmodyfikowane tabele

10.4.1. Tabela Company

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	Kraków	Klient 1	Hofmana Vlastimila	30-210
2	2	Kraków	Klient 2	3 Maja Al.	30-063
3	3	Warszawa	Dostawca 1	Mikołaja Kopernika 3	00-367
4	4	Kraków	Dostawca 2	Obożna	30-011

10.4.2. Tabela Customer

	DISCOUNT	COMPANYID
1	5.5	1
2	9.75	2

10.4.3. Tabela Supplier

	BANKACCOUNTNUMBER	COMPANYID
1	123123123123123123123123	3
2	999888777666555444333222	4