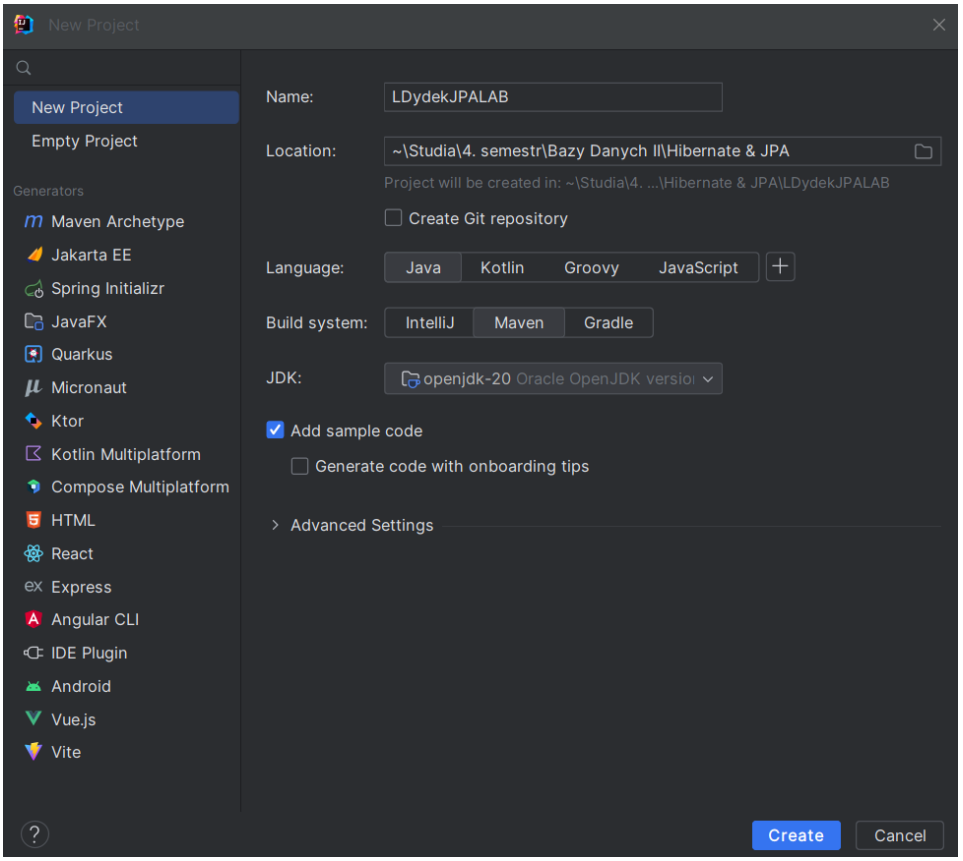
Hibernate i JPA

1. Konfiguracja

a. Pobrano i uruchomiono serwer bazodanowy Apache Derby:

```
Thu Jun 08 02:30:23 CEST 2023 : Serwer sieciowy Apache Derby - 10.16.1.1 - (1901046)
 uruchomiony i gotowy do zaakceptowania połączeń na porcie 1527 w {3}
```

b. Utworzono nowy projekt:



c. Uzupełniono plik konfiguracyjny Mavena pom.xml o potrzebne zależności w celu dołączenia Hibernate'a:

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.6.0.Final</version>
    </dependency>
</dependencies>
```

d. Utworzono przy okazji plik konfiguracyjny 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"/>
<property name="connection.driver_class"/>
<!-- <property name="connection.username"/> →
<!-- <property name="connection.password"/> →
<!-- DB schema will be updated if needed →
<!-- <property
name="hibernate.hbm2ddl.auto">update</property> →
</session-factory>
</hibernate-configuration>
```

e. Uzupełniono go o potrzebne wpisy w celu połączenia z Apache Derby:

```
<session-factory>
<property
name="connection.driver_class">org.apache.derby.jdbc.Clie
ntDriver</property>
<property
name="connection.url">jdbc:derby://127.0.0.1/LukaszDydekJ
PA;create=true</property>
<property
name="dialect">org.hibernate.dialect.DerbyTenSevenDialect
</property>
<property name="show_sql">true</property>
<property name="format_sql">true</property>
<property name="use_sql_comments">true</property>
<property name="hbm2ddl.auto">create-drop</property>
</session-factory>
```

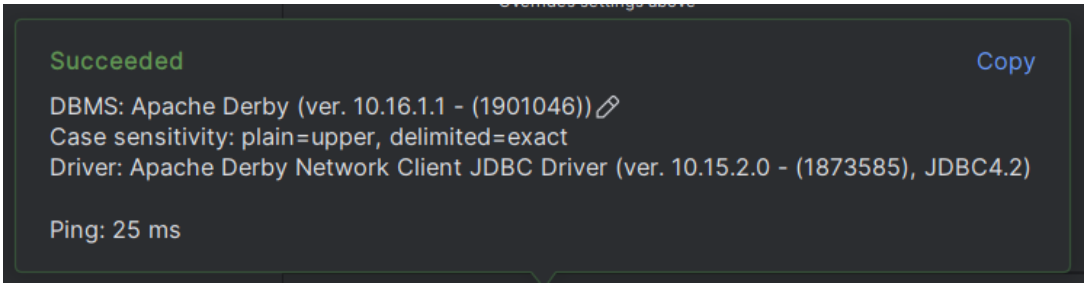
f. Dołączono do projektu pliki z rozszerzeniem jar związane z komunikacją z Derby.

g. Uruchomiono projekt:

```
INFO: HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
cze 08, 2023 2:33:42 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
cze 08, 2023 2:33:42 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: using driver [org.apache.derby.jdbc.ClientDriver] at URL [jdbc:derby://127.0.0.1/LukaszDydekJPA;create=true]
cze 08, 2023 2:33:42 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {}
cze 08, 2023 2:33:42 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
cze 08, 2023 2:33:42 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
cze 08, 2023 2:33:44 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.DerbyTenSevenDialect
cze 08, 2023 2:33:45 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]

Process finished with exit code 0
```

h. Połączanie z poziomu IntelliJ do serwera Derby zakończyło się pomyślnie:



2. Praca z modelem

1) Utworzenie klasy Product

a. Kody:

Klasa 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;
     }
 }
```

Metoda main klasy Main:

```
public static void main(final String[] args) throws Exception
{
     final Session session = getSession();
     Product product = new Product("Stół", 10);
     try {
          Transaction tx = session.beginTransaction();
          session.save(product);
```

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

Plik konfiguracyjny:

```
<?xml version='1.0' encoding='utf-8'?>
 <!DOCTYPE hibernate-configuration PUBLIC
       "-//Hibernate/Hibernate Configuration DTD//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
 <hibernate-configuration>
   <session-factory>
       <property
name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
       <property
name="connection.url">jdbc:derby://127.0.0.1/LukaszDydekJPA;create=true</prop
erty>
       <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="org.example.Product"/>
   </session-factory>
 </hibernate-configuration>
```

b. Utworzona tabela:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Stół	10

c. Logi SQL:

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

/* insert org.example.Product
        */ insert
        into
            Product
```

(productName, unitsOnStock, productID)
values
(?, ?, ?)

2) **Wprowadzenie pojęcia dostawcy:**

a. Stworzono nowego dostawcę:

Kod klasy Supplier:

```
@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;
     }
 }
```

Zmodyfikowano kod klasy Product, dodając pole “supplier” tworzące relację:

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

     @ManyToOne
     @JoinColumn(name = "supplierID")
     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;
    }
}
```

Do pliku konfiguracyjnego Hibernate dodano linijkę:

```
<mapping class="org.example.Product"/>
```

oraz zmieniono property:

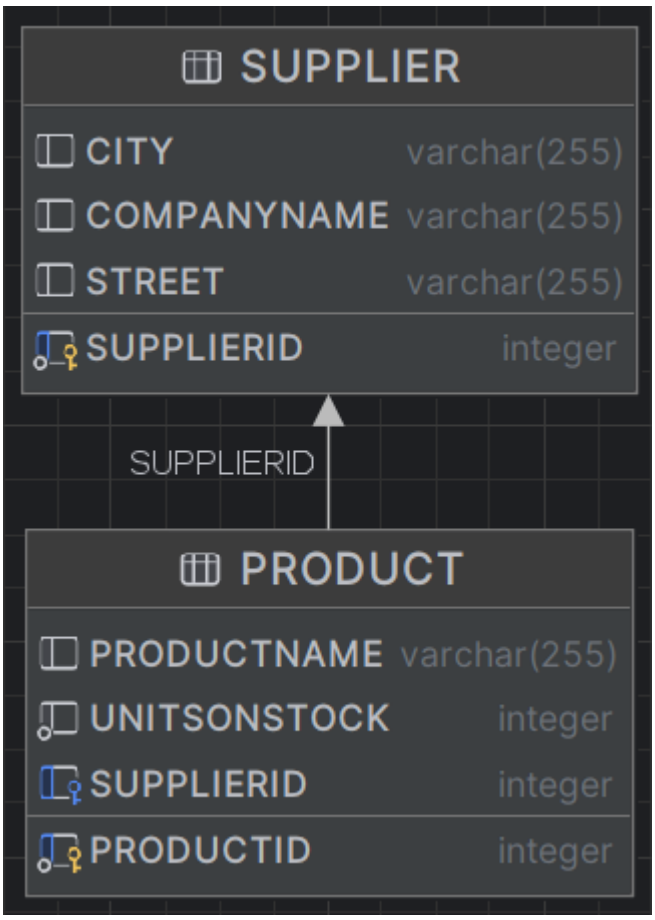
```
<!-- <property name="hbm2ddl.auto">create-drop</property>-->
<property name="hbm2ddl.auto">update</property>
```

b. Aktualne tabele w bazie:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIERID
1	1	Stół	10	2

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	2	Kraków	DPD	Atomowa

c. Schemat bazy danych:



d. Logi SQL:

```
select
        product0_.productID as producti1_0_0_,
        product0_.productName as productn2_0_0_,
        product0_.supplierID as supplier4_0_0_,
        product0_.unitsOnStock as unitsons3_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 hibernate_sequence

    /* insert org.example.Supplier
```

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

    /* update
        org.example.Product */ update
            Product
        set
            productName=?,
            supplierID=?,
            unitsOnStock=?
        where
            productID=?
```

3) Odwrócenie relacji

1. Z tabelą łącznikową

a. Zmieniono kod w klasach Supplier, Product oraz Main:

Klasa Supplier:

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

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

     public Supplier() {}

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

```
    public void addProducts(List<Product> products) {
        this.products.addAll(products);
    }
}
```

Klasa 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;
    }
}
```

Metoda main klasy Main:

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

    // creating products collection
    Product product1 = new Product("Szafa", 5);
    Product product2 = new Product("Ławka", 30);
    Product product3 = new Product("Komoda", 2);
    List<Product> productList = new ArrayList<>();
    productList.add(product1);
    productList.add(product2);
    productList.add(product3);
    try {
        Transaction tx = session.beginTransaction();

        // finding latest supplier
        Supplier supplier = session.get(Supplier.class, 1);
        supplier.addProducts(productList);
```

```
            session.save(product1);
            session.save(product2);
            session.save(product3);
            tx.commit();
        } finally {
            session.close();
        }
}
```

b. Aktualne tabele w bazie:

Tabela PRODUCT:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	4	Komoda	2
2	2	Szafa	5
3	3	Ławka	30

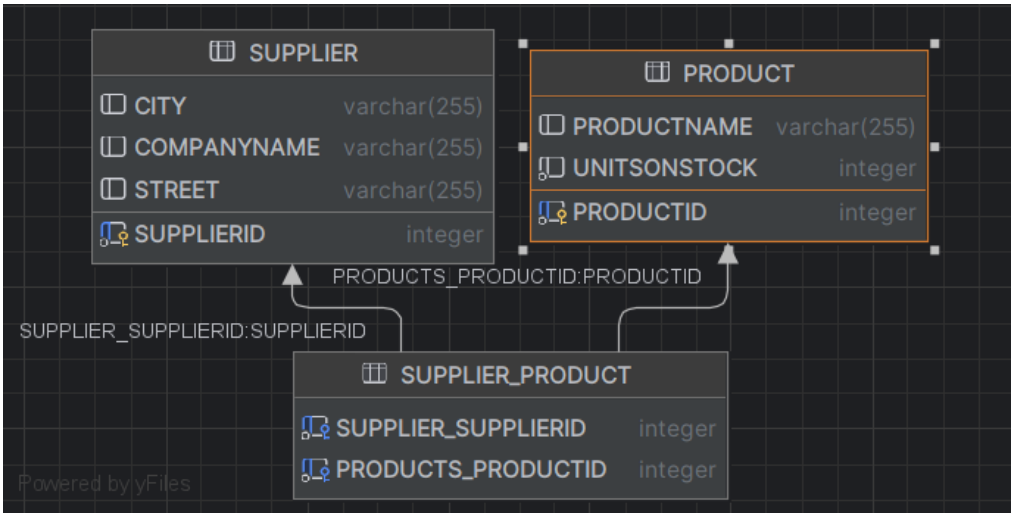
Tabela SUPPLIER:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	DPD	Atomowa

Tabela łącznikowa:

	SUPPLIER_SUPPLIERID	PRODUCTS_PRODUCTID
1	1	2
2	1	3
3	1	4

c. Diagram bazy danych:



d. Logi SQL:

```
create table Supplier_Product (
       Supplier_supplierID integer not null,
        products_productID integer not null
    )

alter table Supplier_Product
       add constraint UK_sd4mo32rnl54mui98qw7bn159 unique (products_productID)

alter table Supplier_Product
       add constraint FKar5fwoh7a3vqxo0f8fh1ey8ha
       foreign key (products_productID)
       references Product

alter table Supplier_Product
       add constraint FKjskj7cplt17tebkn930wt8ke6
       foreign key (Supplier_supplierID)
       references Supplier

select
        supplier0_.supplierID as supplier1_1_0_,
        supplier0_.city as city2_1_0_,
        supplier0_.companyName as companyn3_1_0_,
        supplier0_.street as street4_1_0_
    from
        Supplier supplier0_
    where
        supplier0_.supplierID=?

select
        products0_.Supplier_supplierID as supplier1_2_0_,
        products0_.products_productID as products2_2_0_,
        product1_.productID as producti1_0_1_,
        product1_.productName as productn2_0_1_,
        product1_.unitsOnStock as unitsons3_0_1_
    from
        Supplier_Product products0_
    inner join
        Product product1_
            on products0_.products_productID=product1_.productID
    where
        products0_.Supplier_supplierID=?

/* insert org.example.Product
        */ insert
        into
            Product
            (productName, unitsOnStock, productID)
```

```
        values
            (?, ?, ?)

    /* insert org.example.Product
        */ insert
        into
            Product
            (productName, unitsOnStock, productID)
        values
            (?, ?, ?)

    /* insert org.example.Product
        */ insert
        into
            Product
            (productName, unitsOnStock, productID)
        values
            (?, ?, ?)

    /* insert collection
        row org.example.Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_supplierID, products_productID)
        values
            (?, ?)

    /* insert collection
        row org.example.Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_supplierID, products_productID)
        values
            (?, ?)

    /* insert collection
        row org.example.Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_supplierID, products_productID)
        values
            (?, ?)
```

2. **Bez tabeli łącznikowej**
   a. Kod zmieniono tylko w klasie Supplier i zmiana ta dotyczyła jedynie dekoratorów kolekcji:

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

Kod wygląda teraz następująco:

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

b. Aktualne tabele w bazie:

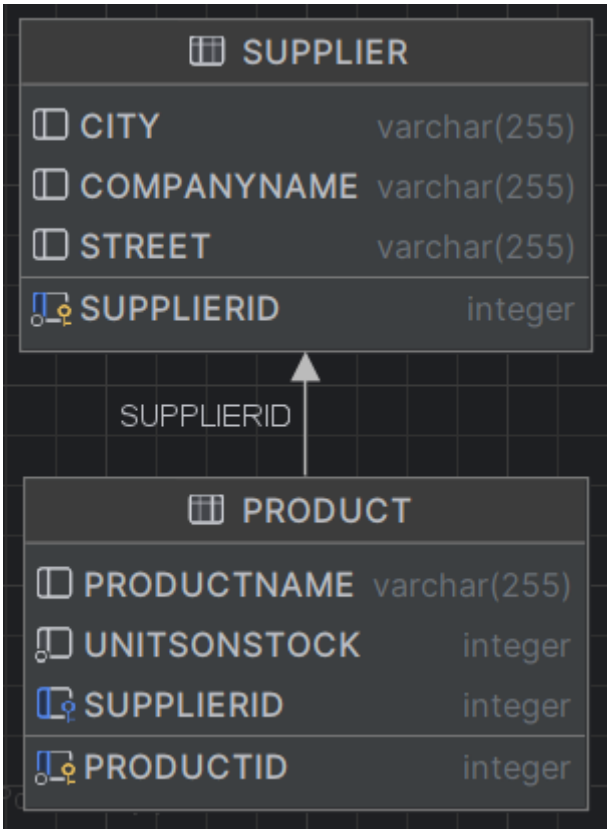
Tabela PRODUCT:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIERID
1	4	Komoda	2	1
2	2	Szafa	5	1
3	3	Ławka	30	1

Tabela SUPPLIER:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	DPD	Atomowa

c. Diagram bazy danych:



d. Logi SQL:

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

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
        supplier0_.supplierID as supplier1_1_0_,
        supplier0_.city as city2_1_0_,
        supplier0_.companyName as companyn3_1_0_,
        supplier0_.street as street4_1_0_
    from
        Supplier supplier0_
    where
        supplier0_.supplierID=?

select
        products0_.supplierID as supplier4_0_0_,
        products0_.productID as producti1_0_0_,
        products0_.productID as producti1_0_1_,
        products0_.productName as productn2_0_1_,
        products0_.unitsOnStock as unitsons3_0_1_
    from
        Product products0_
    where
        products0_.supplierID=?

/* insert org.example.Product
        */ insert
        into
            Product
            (productName, unitsOnStock, productID)
        values
```

```
            (?, ?, ?)

        /* insert org.example.Product
            */ insert
            into
                Product
                (productName, unitsOnStock, productID)
            values
                (?, ?, ?)

        /* insert org.example.Product
            */ insert
            into
                Product
                (productName, unitsOnStock, productID)
            values
                (?, ?, ?)

        /* create one-to-many row org.example.Supplier.products */ update
                Product
            set
                supplierID=?
            where
                productID=?
```

4) **Modelowanie relacji dwustronnej:**

a. Do klas mapowanych na tabele dodano adnotacje @Table(name = “table_name”), dzięki czemu nazwy tabel będą odtąd w liczbie mnogiej.

Aktualne kody klas wyglądają następująco:

Klasa Product:

```
@Entity
 @Table(name = "Products")
 public class Product {
     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     private int productID;
     private String productName;
     private int unitsOnStock;

     @ManyToOne
     @JoinColumn(name = "supplierID")
     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;
    }
}
```

Klasa Supplier:

```
@Entity
@Table(name = "Suppliers")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    private String companyName;
    private String street;
    private String city;

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

    public Supplier() {}

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

    public void addProducts(List<Product> products) {
        this.products.addAll(products);
    }
}
```

Klasa Main:

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

    // creating products collection
    Product product1 = new Product("Szafa", 5);
    Product product2 = new Product("Ławka", 30);
    Product product3 = new Product("Komoda", 2);
    List<Product> productList = new ArrayList<>();
    productList.add(product1);
    productList.add(product2);
    productList.add(product3);

    try {
        Transaction tx = session.beginTransaction();

        // finding latest supplier
        Supplier supplier = session.get(Supplier.class, 1);
        supplier.addProducts(productList);

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

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

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

b. Aktualne tabele w bazie:

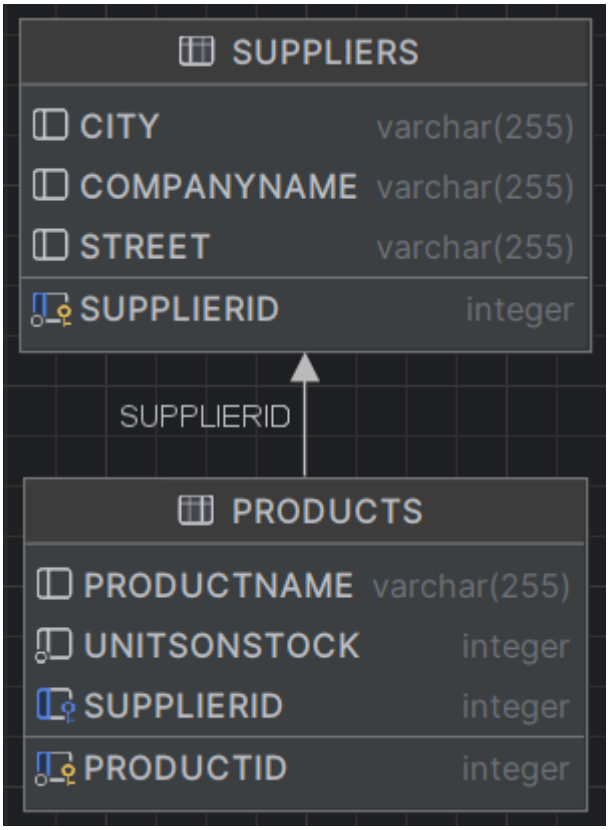
Tabela SUPPLIERS:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	DPD	Atomowa

Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIERID
1	2	Szafa	5	1
2	3	Ławka	30	1
3	4	Komoda	2	1

c. Diagram bazy danych:



d. Logi SQL:

```
create table Products (
      productID integer not null,
       productName varchar(255),
       unitsOnStock 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

select
    supplier0_.supplierID as supplier1_1_0_,
    supplier0_.city as city2_1_0_,
    supplier0_.companyName as companyn3_1_0_,
    supplier0_.street as street4_1_0_
  from
    Suppliers supplier0_
  where
    supplier0_.supplierID=?


select
    products0_.supplierID as supplier4_0_0_,
    products0_.productID as producti1_0_0_,
    products0_.productID as producti1_0_1_,
    products0_.productName as productn2_0_1_,
    products0_.supplierID as supplier4_0_1_,
    products0_.unitsOnStock as unitsons3_0_1_
  from
    Products products0_
  where
    products0_.supplierID=?

      /* insert org.example.Product
          */ insert
          into
            Products
            (productName, supplierID, unitsOnStock, productID)
          values
            (?, ?, ?, ?)

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

5) Utworzenie klasy Category

a. Kod z klas Category, Product oraz fragment z klasy Main poniżej:

Klasa Category:

```java
@Entity
 @Table(name = "Categories")
 public class Category {
     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     private int categoryID;
     private String name;
     @OneToMany
     @JoinColumn(name = "categoryID")
     private List<Product> products = new ArrayList<>();

     public Category() {}

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

     public void addProducts(List<Product> productList) {
          this.products.addAll(productList);
     }
 }
```

Klasa Product:

```java
@Entity
 @Table(name = "Products")
 public class Product {
     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     private int productID;
     private String productName;
     private int unitsOnStock;

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

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

     public Product() {}

     public Product(String productName, int unitsOnStock) {
```

```
            this.productName = productName;
            this.unitsOnStock = unitsOnStock;
      }

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

Metoda main klasy Main:

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

      // creating products collection
      Product wardrobe = new Product("Szafa", 5);
      Product bench = new Product("Ławka", 30);
      Product chestOfDrawers = new Product("Komoda", 2);
      Product water = new Product("Woda", 10);
      Product juice = new Product("Sok", 3);
      List<Product> furnitureList = new ArrayList<>();
      List<Product> beverageList = new ArrayList<>();
      furnitureList.add(wardrobe);
      furnitureList.add(bench);
      furnitureList.add(chestOfDrawers);
      beverageList.add(water);
      beverageList.add(juice);

      // creating categories
      Category furniture = new Category("Meble");
      Category beverages = new Category("Napoje");

      try {
            Transaction tx = session.beginTransaction();

            wardrobe.setCategory(furniture);
            bench.setCategory(furniture);
            chestOfDrawers.setCategory(furniture);
            water.setCategory(beverages);
            juice.setCategory(beverages);

            furniture.addProducts(furnitureList);
            beverages.addProducts(beverageList);
```

```
            session.save(wardrobe);
            session.save(bench);
            session.save(chestOfDrawers);
            session.save(water);
            session.save(juice);
            session.save(furniture);
            session.save(beverages);

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

b. Aktualne tabele w bazie:

Tabela CATEGORIES:

	CATEGORYID	NAME
1	6	Meble
2	7	Napoje

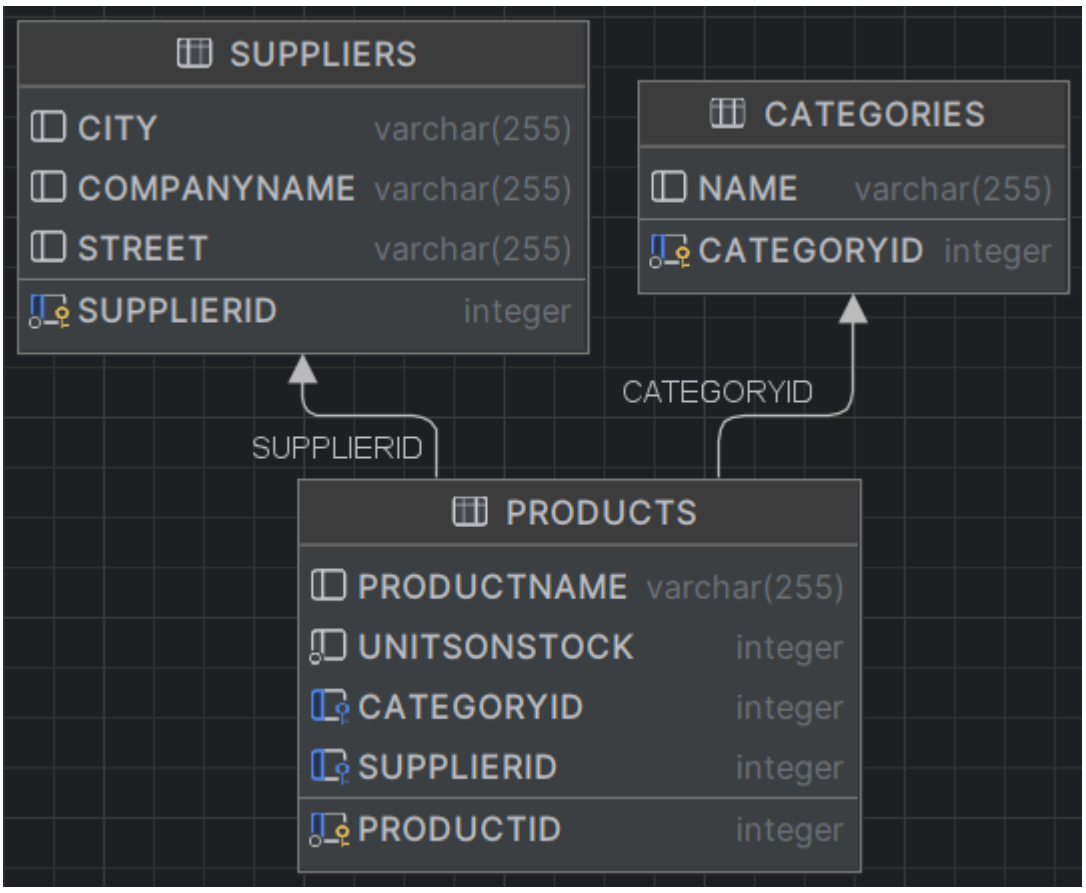
Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYID	SUPPLIERID
1	1	Szafa	5	6	<null>
2	2	Ławka	30	6	<null>
3	3	Komoda	2	6	<null>
4	4	Woda	10	7	<null>
5	5	Sok	3	7	<null>

Pusta tabela SUPPLIERS:

SUPPLIERID	CITY	COMPANYNAME	STREET

c. Diagram bazy danych:



d. Logi SQL:

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

create table Products (
      productID integer not null,
       productName varchar(255),
       unitsOnStock integer not null,
       categoryID integer,
       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 FKn4dvny5ajgqgw20l5nb7imd5t
       foreign key (categoryID)
       references Categories

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

/* insert org.example.Product
       */ insert
       into
          Products
          (categoryID, productName, supplierID, unitsOnStock, productID)
       values
          (?, ?, ?, ?, ?)

/* insert org.example.Product
       */ insert
       into
          Products
          (categoryID, productName, supplierID, unitsOnStock, productID)
       values
          (?, ?, ?, ?, ?)

/* insert org.example.Product
       */ insert
       into
          Products
          (categoryID, productName, supplierID, unitsOnStock, productID)
       values
          (?, ?, ?, ?, ?)

/* update
       org.example.Product */ update
          Products
       set
          categoryID=?,
          productName=?,
          supplierID=?,
          unitsOnStock=?
       where
          productID=?

/* update
       org.example.Product */ update
```

```
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsOnStock=?
    where
        productID=?

/* update
    org.example.Product */ update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsOnStock=?
    where
        productID=?

/* create one-to-many row org.example.Category.products */ update
        Products
   set
        categoryID=?
   where
        productID=?

/* create one-to-many row org.example.Category.products */ update
        Products
   set
        categoryID=?
   where
        productID=?
```

e. Wydobyto także produkty z wybranej kategorii oraz kategorię, do której należy wybrany produkt. Użyto do tego standardowego zapytania HQL.

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

     try {
          // produkty należące do poszczególnych kategorii
          Transaction tx = session.beginTransaction();
          Query query = session.createQuery("from Category");
          query.getResultList().forEach(c -> {
```

```
            System.out.println("Kategoria: " + c + ": ");
            ((Category) c).getProducts().forEach(p ->
System.out.println("\t- " + p + ","));
            });

            System.out.println();

            // kategoria, do której należy wybrany produkt
            Product product = session.get(Product.class, 5);
            System.out.println(product.toString() + " należy do
kategorii: " + product.getCategory());
            tx.commit();
        } finally {
            session.close();
        }
    }
```

Na standardowym wyjściu można było zobaczyć:

// początek
Kategoria: Meble:
- Szafa (5 szt.),
- Ławka (30 szt.),
- Komoda (2 szt.),

Kategoria: Napoje:
- Woda (10 szt.),
- Sok (3 szt.),

Sok (3 szt.) należy do kategorii: Napoje
// koniec

6) **Relacja wiele-do-wielu:**

a. Kody wybranych klas:

Klasa Product:

- dodano metodę odpowiedzialną za sprzedaż produktów

```
public void sell(Invoice invoice, int quantity) throws
Exception {
    if (this.unitsOnStock < quantity) {
        throw new Exception("Cannot sell " + quantity + "
products.");
    }
    this.unitsOnStock -= quantity;
```

```
        invoice.addProduct(this, quantity);
        invoices.add(invoice);
    }
```

- dodano adnotację odpowiedzialną za powstanie relacji wiele-do-wielu:

```
@ManyToMany
@JoinTable(name = "invoice_details",
            joinColumns = @JoinColumn(name = "invoiceNumber"),
            inverseJoinColumns = @JoinColumn(name =
"productID"))
 private List<Invoice> invoices;
```

Klasa Invoice:

```
@Entity
 public class Invoice {
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private int invoiceNumber;
        private int quantity = 0;

        @ManyToMany(mappedBy = "invoices")
        private List<Product> products;

        public Invoice() {}

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

Można zauważyć, że również w klasie Invoice jest obecna adnotacja odpowiedzialna za powstanie relacji wiele-do-wielu.

Metoda main klasy Main:

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

        Product armchair = new Product("Fotel", 7);
        Product table = new Product("Stół", 20);
        Product shelf = new Product("Półka", 5);
```

```
        List<Product> productList = new ArrayList<>();
        productList.add(armchair);
        productList.add(table);
        productList.add(shelf);


        Invoice invoice1 = new Invoice();
        Invoice invoice2 = new Invoice();

        try {
            Transaction tx = session.beginTransaction();

            // finding furniture category
            Category furniture = session.get(Category.class, 6);

            // relation between product and category
            armchair.setCategory(furniture);
            table.setCategory(furniture);
            shelf.setCategory(furniture);
            furniture.addProducts(productList);

            session.save(armchair);
            session.save(table);
            session.save(shelf);
            session.save(invoice1);
            session.save(invoice2);
            tx.commit();
        } finally {
            session.close();
        }
    }
```

Na samym początku tworzę kilka produktów oraz faktur i zapisuję je w bazie. Teraz przechodzę do sprzedaży produktów:

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

        try {
            Transaction tx = session.beginTransaction();
            Product armchair = session.get(Product.class, 9);
            Product table = session.get(Product.class, 10);
            Product shelf = session.get(Product.class, 11);
```

```
            Invoice invoice1 = session.get(Invoice.class, 12);
            Invoice invoice2 = session.get(Invoice.class, 13);
            armchair.sell(invoice1, 1);
            table.sell(invoice1, 2);
            table.sell(invoice2, 3);
            shelf.sell(invoice2, 1);
            tx.commit();
        } finally {
            session.close();
        }
}
```

b. Aktualne tabele w bazie:

- przed dokonaniem sprzedaży:

Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYID	SUPPLIERID
1	1	Szafa	8	9	13
2	2	Ławka	30	9	14
3	3	Komoda	2	9	14
4	4	Woda	10	10	13
5	5	Sok	10	10	13
6	6	Fotel	30	9	13
7	7	Stół	20	9	14
8	8	Półka	49	9	13
9	16	Smartfon	10	<null>	<null>
10	17	Telewizor	5	<null>	<null>
11	19	Tablet	15	<null>	<null>
12	109	Cegła	8	<null>	<null>
13	111	Deska	2	<null>	<null>

Tabela INVOICES:

	INVOICENUMBER	QUANTITY
1	11	6
2	12	0
3	15	3
4	18	4
5	110	2
6	112	3

Tabela INVOICE_DETAILS:

	PRODUCTS_PRODUCTID	INVOICES_INVOICENUMBER
1	109	110
2	111	112
3	19	11

- po dokonaniu sprzedaży:

  Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYID	SUPPLIERID
1	1	Szafa	8	9	13
2	2	Ławka	30	9	14
3	3	Komoda	2	9	14
4	4	Woda	10	10	13
5	5	Sok	10	10	13
6	6	Fotel	29	9	13
7	7	Stół	15	9	14
8	8	Półka	48	9	13
9	16	Smartfon	10	<null>	<null>
10	17	Telewizor	5	<null>	<null>
11	19	Tablet	15	<null>	<null>
12	109	Cegła	8	<null>	<null>
13	111	Deska	2	<null>	<null>

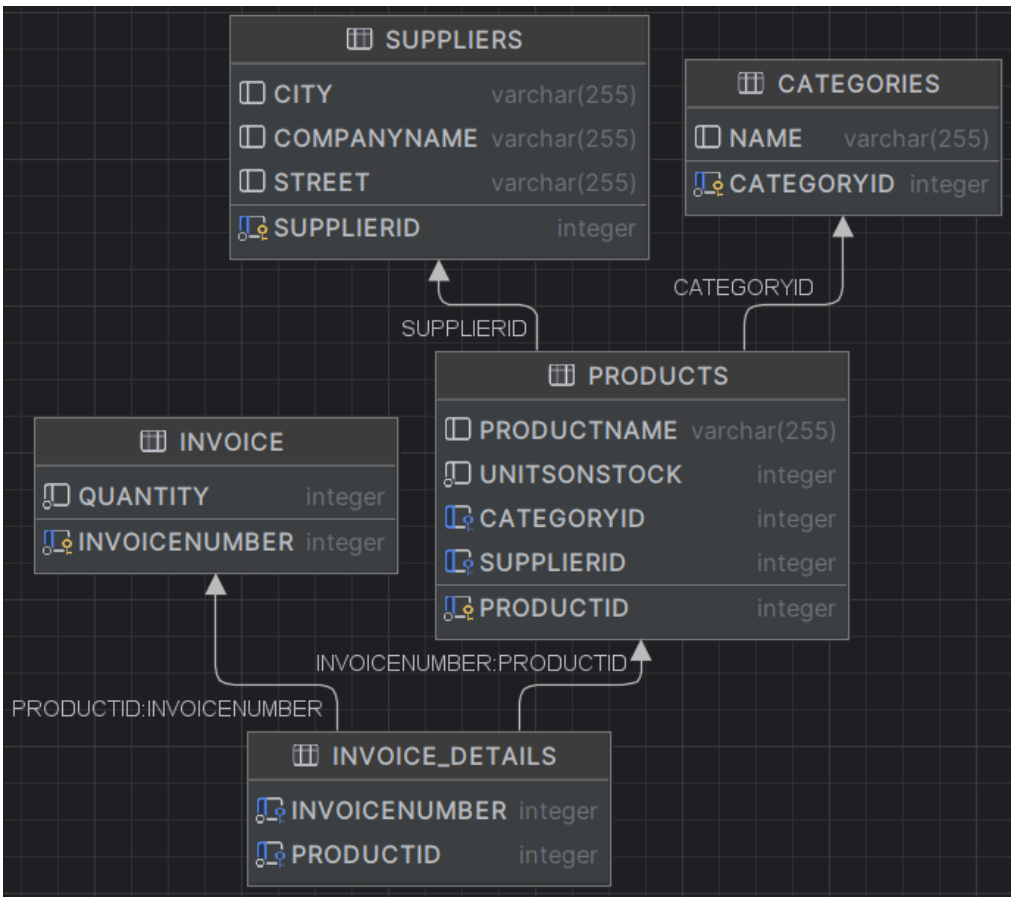
  Tabela INVOICES:

	INVOICENUMBER	QUANTITY
1	11	9
2	12	4
3	15	3
4	18	4
5	110	2
6	112	3

  Tabela INVOICE_DETAILS:

	PRODUCTS_PRODUCTID	INVOICES_INVOICENUMBER
1	109	110
2	111	112
3	19	11
4	6	11
5	7	11
6	7	12
7	8	12

c. Diagram bazy danych:



d. Logi SQL:

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

create table Products_Invoice (
       products_productID integer not null,
        invoices_invoiceID integer not null
    )

alter table Products_Invoice
       add constraint FKhry8ecq3t3p2kyl7ua77xkna8
       foreign key (invoices_invoiceID)
       references Invoice

alter table Products_Invoice
       add constraint FKms37fk2r2dy3rsx0jpk8nne3i
       foreign key (products_productID)
       references Products
```

```
/* insert org.example.Invoice
        */ insert
        into
            Invoice
            (quantity, invoiceNumber)
        values
            (?, ?)

/* insert org.example.Invoice
        */ insert
        into
            Invoice
            (quantity, invoiceNumber)
        values
            (?, ?)

/* insert org.example.Product
        */ insert
        into
            Products
            (categoryID, productName, supplierID, unitsOnStock, productID)
        values
            (?, ?, ?, ?, ?)

/* insert org.example.Product
        */ insert
        into
            Products
            (categoryID, productName, supplierID, unitsOnStock, productID)
        values
            (?, ?, ?, ?, ?)

/* insert org.example.Product
        */ insert
        into
            Products
            (categoryID, productName, supplierID, unitsOnStock, productID)
        values
            (?, ?, ?, ?, ?)
/* create one-to-many row org.example.Category.products */ update
        Products
    set
        categoryID=?
    where
        productID=?

/* insert collection
```

```
            row org.example.Product.invoices */ insert
            into
               invoice_details
               (invoiceNumber, productID)
            values
               (?, ?)

         /* insert collection
            row org.example.Product.invoices */ insert
            into
               invoice_details
               (invoiceNumber, productID)
            values
               (?, ?)

         /* insert collection
            row org.example.Product.invoices */ insert
            into
               invoice_details
               (invoiceNumber, productID)
            values
               (?, ?)

         /* insert collection
            row org.example.Product.invoices */ insert
            into
               invoice_details
               (invoiceNumber, productID)
            values
               (?, ?)
```

7) JPA

a. Dodano nowy plik konfiguracyjny:

```
<?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="myPersistenceUnit"
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/LukaszDydekJPA"/>
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true"
/>
            <property name="hibernate.hbm2ddl.auto"
value="update" />
            </properties>
        </persistence-unit>
 </persistence>
```

Kod klasy Main wygląda teraz następująco:

```
public class Main {
     private static final EntityManagerFactory
entityManagerFactory;

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

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

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

          try {
          EntityTransaction tx =
entityManager.getTransaction();
          tx.begin();

          Product tablet = entityManager.find(Product.class,
19);
          Invoice invoice = entityManager.find(Invoice.class,
```

```
11);
            tablet.sell(invoice, 5);

            tx.commit();
            } finally {
            entityManager.close();
            }

            entityManagerFactory.close();
        }
 }
```

b. Aktualne tabele w bazie:

- przed dokonaniem sprzedaży:

Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYID	SUPPLIERID
1	1	Szafa	8	9	13
2	2	Ławka	30	9	14
3	3	Komoda	2	9	14
4	4	Woda	10	10	13
5	5	Sok	10	10	13
6	6	Fotel	30	9	13
7	7	Stół	20	9	14
8	8	Półka	49	9	13
9	16	Smartfon	10	<null>	<null>
10	17	Telewizor	5	<null>	<null>
11	19	Tablet	20	<null>	<null>
12	109	Cegła	8	<null>	<null>
13	111	Deska	2	<null>	<null>

Tabela INVOICES:

	INVOICENUMBER	QUANTITY
1	11	1
2	12	0
3	15	3
4	18	4
5	110	2
6	112	3

Tabela INVOICE_DETAILS:

	PRODUCTS_PRODUCTID	INVOICES_INVOICENUMBER
1	109	110
2	111	112

- po dokonaniu sprzedaży:

Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYID	SUPPLIERID
1	1	Szafa	8	9	13
2	2	Ławka	30	9	14
3	3	Komoda	2	9	14
4	4	Woda	10	10	13
5	5	Sok	10	10	13
6	6	Fotel	30	9	13
7	7	Stół	20	9	14
8	8	Półka	49	9	13
9	16	Smartfon	10	<null>	<null>
10	17	Telewizor	5	<null>	<null>
11	19	Tablet	15	<null>	<null>
12	109	Cegła	8	<null>	<null>
13	111	Deska	2	<null>	<null>

Tabela INVOICES:

	INVOICENUMBER	QUANTITY
1	11	6
2	12	0
3	15	3
4	18	4
5	110	2
6	112	3

Tabela INVOICE_DETAILS:

	PRODUCTS_PRODUCTID	INVOICES_INVOICENUMBER
1	109	110
2	111	112
3	19	11

Widać że dane w stosunku do poprzedniego podpunktu zmieniły się o jeden sprzedany stół.

c. Logi SQL:

```
select
    product0_.productID as producti1_3_0_,
    product0_.categoryID as category4_3_0_,
    product0_.productName as productn2_3_0_,
    product0_.supplierID as supplier5_3_0_,
    product0_.unitsOnStock as unitsons3_3_0_,
    category1_.categoryID as category1_0_1_,
    category1_.name as name2_0_1_,
    supplier2_.supplierID as supplier1_4_2_,
    supplier2_.city as city2_4_2_,
    supplier2_.companyName as companyn3_4_2_,
```

```
        supplier2_.street as street4_4_2_
    from
        Products product0_
    left outer join
        Categories category1_
            on product0_.categoryID=category1_.categoryID
    left outer join
        Suppliers supplier2_
            on product0_.supplierID=supplier2_.supplierID
    where
        product0_.productID=?

select
        invoice0_.invoiceNumber as invoicen1_1_0_,
        invoice0_.quantity as quantity2_1_0_
    from
        Invoice invoice0_
    where
        invoice0_.invoiceNumber=?


select
        invoices0_.invoiceNumber as invoicen1_2_0_,
        invoices0_.productID as producti2_2_0_,
        invoice1_.invoiceNumber as invoicen1_1_1_,
        invoice1_.quantity as quantity2_1_1_
    from
        invoice_details invoices0_
    inner join
        Invoice invoice1_
            on invoices0_.productID=invoice1_.invoiceNumber
    where
        invoices0_.invoiceNumber=?

update
        Products
    set
        categoryID=?,
        productName=?,
        supplierID=?,
        unitsOnStock=?
    where
        productID=?

update
        Invoice
    set
        quantity=?
```

```
    where
        invoiceNumber=?

delete
    from
        invoice_details
    where
        invoiceNumber=?

insert
    into
        invoice_details
        (invoiceNumber, productID)
    values
        (?, ?)

insert
    into
        invoice_details
        (invoiceNumber, productID)
    values
        (?, ?)

insert
    into
        invoice_details
        (invoiceNumber, productID)
    values
        (?, ?)

insert
    into
        invoice_details
        (invoiceNumber, productID)
    values
        (?, ?)
```

8) Kaskady

1. tworzenie faktur wraz z nowymi produktami

a. Kody:
Do klas Product oraz Invoice dodano parametr “cascade” adnotacji @ManyToMany.

Klasa Product:

```
@ManyToMany(cascade = CascadeType.ALL)
 @JoinTable(name = "invoice_details",
          joinColumns = @JoinColumn(name = "invoiceNumber"),
          inverseJoinColumns = @JoinColumn(name =
"productID"))
 private List<Invoice> invoices;
```

Klasa Invoice:

```
@ManyToMany(cascade = CascadeType.ALL, mappedBy = "invoices")
private List<Product> products = new ArrayList<>();
```

Metoda main klasy Main:

```
public static void main(final String[] args) throws Exception
{
     final EntityManager entityManager = getEntityManager();

     Product smartphone = new Product("Smartfon", 10);
     Product television = new Product("Telewizor", 5);
     Product tablet = new Product("Tablet", 20);
     Invoice invoice1 = new Invoice();
     Invoice invoice2 = new Invoice();
     try {
          EntityTransaction tx =
entityManager.getTransaction();
          tx.begin();
          invoice1.addProduct(smartphone, 2);
          invoice1.addProduct(television, 1);
          invoice2.addProduct(tablet, 4);
          entityManager.persist(invoice1);
          entityManager.persist(invoice2);
          tx.commit();
     } finally {
          entityManager.close();
     }

     entityManagerFactory.close();
 }
```

Widzimy tutaj, iż w bazie zapisujemy tylko faktury a jak się zaraz przekonamy zostały w niej utrwalone także utworzone przedmioty.

b. Aktualne tabele w bazie po dokonaniu sprzedaży:

Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYID	SUPPLIERID
1	1	Szafa	8	9	13
2	2	Ławka	30	9	14
3	3	Komoda	2	9	14
4	4	Woda	10	10	13
5	5	Sok	10	10	13
6	6	Fotel	30	9	13
7	7	Stół	20	9	14
8	8	Półka	49	9	13
9	16	Smartfon	10	<null>	<null>
10	17	Telewizor	5	<null>	<null>
11	19	Tablet	20	<null>	<null>

Tabela INVOICES:

	INVOICENUMBER	QUANTITY
1	11	1
2	12	0
3	15	3
4	18	4

Widzimy tutaj jak na dłoni przykład kaskadowego dodawania danych, ponieważ najpierw do bazy została dodana pierwsza faktura i wszystkie zapisane w niej rzeczy. Następnie dopiero druga z produktami w niej zawartymi (numeracja ID dla faktur oraz produktów).

c. Logi SQL:

```
insert
   into
      Invoices
      (quantity, invoiceNumber)
   values
      (?, ?)

insert
   into
      Products
      (categoryID, productName, supplierID, unitsOnStock, productID)
   values
      (?, ?, ?, ?, ?)

insert
   into
      Products
      (categoryID, productName, supplierID, unitsOnStock, productID)
   values
      (?, ?, ?, ?, ?)
```

```
insert
   into
      Invoices
      (quantity, invoiceNumber)
   values
      (?, ?)

 insert
   into
      Products
      (categoryID, productName, supplierID, unitsOnStock, productID)
   values
      (?, ?, ?, ?, ?)
```

2. **tworzenie produktów wraz z nową fakturą**
   a. Kod metody main w klasie Main:

```
public static void main(final String[] args) throws Exception
{
     final EntityManager entityManager = getEntityManager();

     Product brick = new Product("Cegła", 10);
     Product board = new Product("Deska", 5);
     Invoice invoice1 = new Invoice();
     Invoice invoice2 = new Invoice();
     try {
          EntityTransaction tx =
entityManager.getTransaction();
          tx.begin();
          brick.sell(invoice1, 2);
          board.sell(invoice2, 3);
          entityManager.persist(brick);
          entityManager.persist(board);
          tx.commit();
     } finally {
          entityManager.close();
     }

     entityManagerFactory.close();
 }
```

   b. Aktualne tabele w bazie po dokonaniu sprzedaży:

      Tabela PRODUCTS:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORYID	SUPPLIERID
1	1	Szafa	8	9	13
2	2	Ławka	30	9	14
3	3	Komoda	2	9	14
4	4	Woda	10	10	13
5	5	Sok	10	10	13
6	6	Fotel	30	9	13
7	7	Stół	20	9	14
8	8	Półka	49	9	13
9	16	Smartfon	10	<null>	<null>
10	17	Telewizor	5	<null>	<null>
11	19	Tablet	20	<null>	<null>
12	101	Cegła	8	<null>	<null>
13	103	Deska	2	<null>	<null>

Tabela INVOICES:

	INVOICENUMBER	QUANTITY
1	11	1
2	12	0
3	15	3
4	18	4
5	110	2
6	112	3

Tabela INVOICE_DETAILS:

	PRODUCTS_PRODUCTID	INVOICES_INVOICENUMBER
1	109	110
2	111	112

c. Logi SQL:

```
insert
   into
      Products
      (categoryID, productName, supplierID, unitsOnStock, productID)
   values
      (?, ?, ?, ?, ?)


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

insert
```

```
    into
        Products
        (categoryID, productName, supplierID, unitsOnStock, productID)
    values
        (?, ?, ?, ?, ?)

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

insert
    into
        invoice_details
        (invoiceNumber, productID)
    values
        (?, ?)

insert
    into
        invoice_details
        (invoiceNumber, productID)
    values
        (?, ?)
```

9) Embedded class

1. Dodałem do modelu klasę “Address” i „wbudowałem” ją do tabeli dostawców.

   a. Kody klas:

Klasa Address:

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

    public Address() {}

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

Klasa Supplier:

```
@Entity
 @Table(name = "Suppliers")
 public class Supplier {
     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     private int supplierID;
     private String companyName;
     @Embedded
     private Address address;

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

     public Supplier() {}

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

Metoda main klasy Main:

```
   final EntityManager entityManager = getEntityManager();

     Supplier supplier = new Supplier("Raben", new
Address("Kraków", "Czarnowiejska"));
     try {
          EntityTransaction tx =
entityManager.getTransaction();
          entityManager.persist(supplier);
          tx.begin();
          tx.commit();
     } finally {
          entityManager.close();
     }

     entityManagerFactory.close();
 }
```

b. Aktualny stan tabeli SUPPLIERS:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	13	Kraków	DPD	Atomowa
2	14	Poznań	UPS	Jerozolimska
3	113	Kraków	Raben	Czarnowiejska

c. Logi SQL:

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

2. Zmodyfikowałem model w taki sposób, że dane adresowe znajdują się w klasie dostawców. Następnie zmapowałem to do dwóch osobnych tabel.

a. Kody klas:

Klasa Supplier:

```
@Entity
 @SecondaryTable(name="Addresses")
 @Table(name = "Suppliers")
 public class Supplier {
     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     private int supplierID;
     private String companyName;
     @Column(table = "Addresses")
     private String city;
     @Column(table = "Addresses")
     private String street;

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

     public Supplier() {}

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

```
    }
}
```

Klasa Main:

```
public static void main(final String[] args) throws Exception
{
    final EntityManager entityManager = getEntityManager();

    Supplier supplier = new Supplier("Topsped", "Warszawa",
"Kościuszki");
    try {
        EntityTransaction tx =
entityManager.getTransaction();
        entityManager.persist(supplier);
        tx.begin();
        tx.commit();
    } finally {
        entityManager.close();
    }

    entityManagerFactory.close();
}
```

b. Interesujące nas tabele:

Tabela SUPPLIERS:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	13	Kraków	DPD	Atomowa
2	14	Poznań	UPS	Jerozolimska
3	113	Kraków	Raben	Czarnowiejska
4	114	<null>	Topsped	<null>

Tabela Addresses:

	CITY	STREET	SUPPLIERID
1	Warszawa	Kościuszki	114

c. Logi SQL:

insert
into
Suppliers
(companyName, supplierID)
values

(?, ?)

insert
into
Addresses
(city, street, supplierID)
values
(?, ?, ?)

10) Dziedziczenie

1. Dziedziczenie jednej tabeli

a. Kody klas:

Klasa Company:

```
@Entity
 @Table(name = "Companies")
 @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
 public abstract class Company {
     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     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.street = street;
          this.city = city;
          this.zipCode = zipCode;
     }
 }
```

Klasa Customer:

```
@Entity
 @Table(name = "Customers")
 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;
    }
 }
```

Klasa Supplier:

```
@Entity
 @Table(name = "Suppliers")
 public class Supplier extends Company {
    private String backAccountNumber;

    public Supplier() {}

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

Metoda main klasy Main:

```
public static void main(final String[] args) throws Exception
{
    final EntityManager entityManager = getEntityManager();

    try {
        EntityTransaction tx =
entityManager.getTransaction();
        tx.begin();
        Supplier supplier1 = new Supplier("DPD", "Atomowa",
"Kraków",
            "01-123", "491020289222763005000000000");
        Supplier supplier2 = new Supplier("UPS",
"Jerozolimska", "Poznań",
            "01-124", "491020289222763005000000001");
```

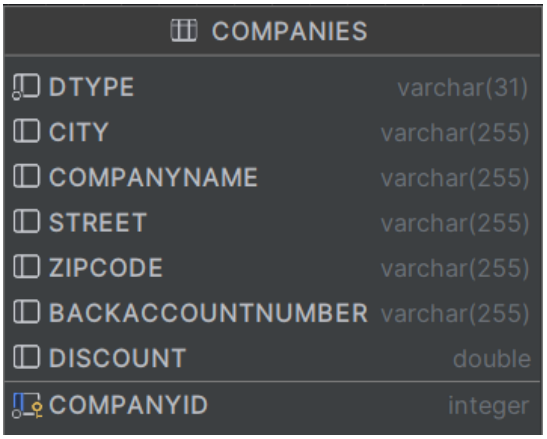
```
            Customer customer1 = new Customer("U Kowalczyka",
"Wiśniowa", "Chojnice", "02-234", 50);
            Customer customer2 = new Customer("U Janiny",
"Lawendowa", "Słupsk", "01-154", 10);
            entityManager.persist(supplier1);
            entityManager.persist(supplier2);
            entityManager.persist(customer1);
            entityManager.persist(customer2);
            tx.commit();
        } finally {
            entityManager.close();
        }

        entityManagerFactory.close();
    }
```

b. Powstała tabela COMPANIES:

	DTYPE	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BACKACCOUNTNUMBER	DISCOUNT
1	Supplier	214	Kraków	DPD	Atomowa	01-123	49102028922276300500000000	<null>
2	Supplier	215	Poznań	UPS	Jerozolimska	01-124	49102028922276300500000001	<null>
3	Customer	216	Chojnice	U Kowalczyka	Wiśniowa	02-234	<null>	50
4	Customer	217	Słupsk	U Janiny	Lawendowa	01-154	<null>	10

c. Diagram bazy danych:



d. Logi SQL:

```
create table Companies (
        DTYPE varchar(31) not null,
         companyID integer not null,
         city varchar(255),
         companyName varchar(255),
         street varchar(255),
         zipCode varchar(255),
         backAccountNumber varchar(255),
```

```
    discount double,
    primary key (companyID)
)

insert
  into
    Companies
    (city, companyName, street, zipCode, backAccountNumber, DTYPE,
companyID)
  values
    (?, ?, ?, ?, ?, 'Supplier', ?)

insert
  into
    Companies
    (city, companyName, street, zipCode, backAccountNumber, DTYPE,
companyID)
  values
    (?, ?, ?, ?, ?, 'Supplier', ?)

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

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

2. Dziedziczenie tabeli konkretnych klas

a. Kody klas:
W stosunku do poprzedniego podpunktu kod pozostał taki sam za wyjątkiem zmiany strategii dziedziczenia (dekorator klasy Company) na:

```
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
```

b. Interesujące nas tabele:

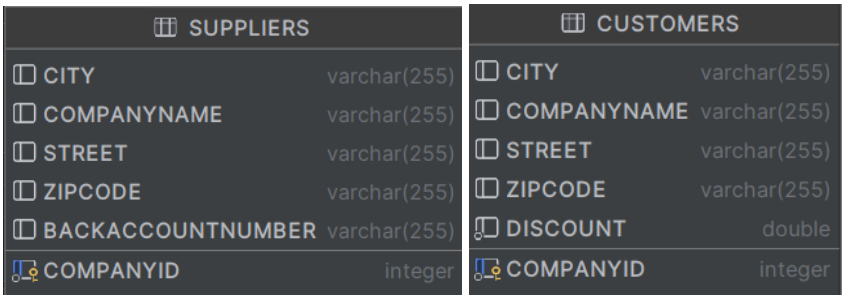
Tabela SUPPLIERS:

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BACKACCOUNTNUMBER
1	222	Kraków	DPD	Atomowa	01-123	49102028922276300500000000
2	223	Poznań	UPS	Jerozolimska	01-124	49102028922276300500000001

Tabela CUSTOMERS:

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT
1	224	Chojnice	U Kowalczyka	Wiśniowa	02-234	50
2	225	Słupsk	U Janiny	Lawendowa	01-154	10

c. Diagram bazy danych:



d. Logi SQL:

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

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

insert
    into
        Suppliers
```

```
        (city, companyName, street, zipCode, backAccountNumber, companyID)
    values
        (?, ?, ?, ?, ?, ?)

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

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

3. Dziedziczenie tabeli abstrakcyjnej klasy nadrzędnej

a. Kody klas:
W stosunku do poprzedniego podpunktu kod pozostał taki sam za wyjątkiem zmiany strategii dziedziczenia (dekorator klasy Company) na:

```
@Inheritance(strategy = InheritanceType.JOINED)
```

b. Interesujące nas tabele:

Tabela COMPANIES:

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	226	Kraków	DPD	Atomowa	01-123
2	227	Poznań	UPS	Jerozolimska	01-124
3	228	Chojnice	U Kowalczyka	Wiśniowa	02-234
4	229	Słupsk	U Janiny	Lawendowa	01-154

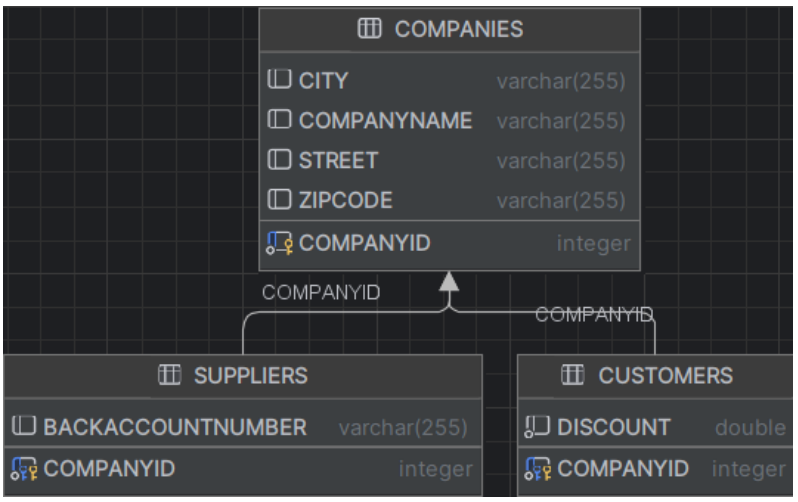
Tabela SUPPLIERS:

	BACKACCOUNTNUMBER	COMPANYID
1	49102028922276300500000000	226
2	49102028922276300500000001	227

Tabela CUSTOMERS:

	DISCOUNT	COMPANYID
1	50	228
2	10	229

c. Diagram bazy danych:



d. Logi SQL:

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

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

create table Suppliers (
       backAccountNumber varchar(255),
        companyID integer not null,
        primary key (companyID)
    )

alter table Customers
       add constraint FKtgbis8219tpno1etu5gnv9oyb
       foreign key (companyID)
       references Companies

alter table Suppliers
       add constraint FKolpan9qos5beolioamgtec5pf
       foreign key (companyID)
       references Companies

insert
```

```
    into
        Companies
        (city, companyName, street, zipCode, companyID)
    values
        (?, ?, ?, ?, ?)

insert
    into
        Suppliers
        (backAccountNumber, companyID)
    values
        (?, ?)

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

insert
    into
        Suppliers
        (backAccountNumber, companyID)
    values
        (?, ?)

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

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

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

```
insert
    into
        Customers
        (discount, companyID)
    values
        (?, ?)
```