Rafał Gaweł

Bazy Danych

Hibernate

II. Tworzenie tabeli Product

Klasa product:

```java
import javax.persistence.*;

@Entity
public class Product {
    public Product() {

    }

    public Product(String productName, Integer unitOnStock) {
        ProductName = productName;
        UnitOnStock = unitOnStock;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Integer id;

    private String ProductName;

    private Integer UnitOnStock;
}
```

Pytamy użytkownika o nazwę produktu i stan magazynu. Tworzymy na jego podstawie obiekt product i utrwalamy go w bazie danych z wykorzystaniem Hibernate.

```
9   public static void main(String[] args) {
10      sessionFactory = getSessionFactory();
11      Session session = sessionFactory.openSession();
12      Transaction tx = session.beginTransaction();
13
14      System.out.println("Podaj nazwe produktu");
15      Scanner inputScanner = new Scanner(System.in);
16      String prodName = inputScanner.nextLine();
17      System.out.println("Podaj liczbe");
18      Integer units = inputScanner.nextInt();
19
20      Product product = new Product(prodName, units);
21      session.save(product);
22
23      tx.commit();
24      session.close();
25  }
```

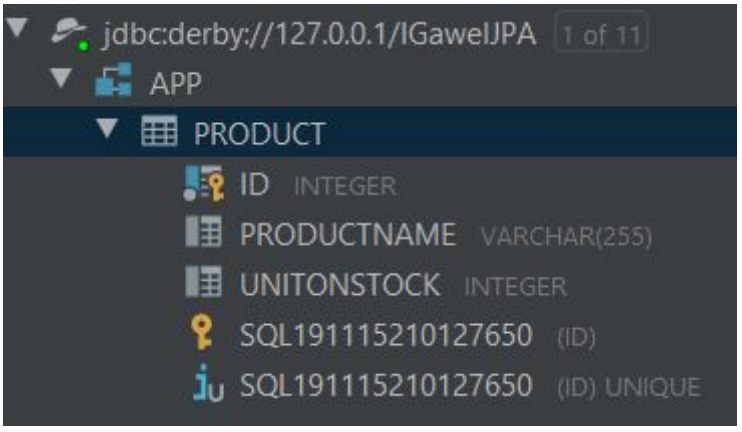
Hibernate tworzy tabele

```
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
lis 15, 2019 9:01:27 PM org.hibernate.resource.transaction.backend.jdbc.in
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibe
Hibernate:

    create table Product (
       id integer not null,
        ProductName varchar(255),
        UnitOnStock integer,
        primary key (id)
    )
```

Następnie dodaje do niej rekord:

```
Podaj nazwe produktu
banany
Podaj liczbe
5
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, id)
        values
            (?, ?, ?)
```

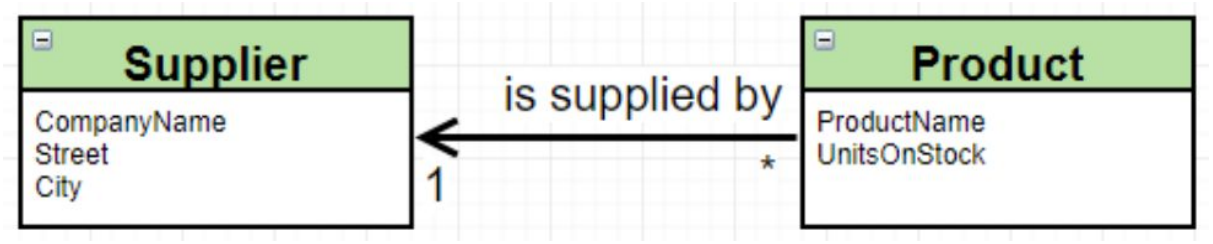
Po tym widzimy rezultat. Została dodana nowa tabele product



A do niej nasz rekord.

	ID	PRODUCTNAME	UNITONSTOCK
1	1	banany	5

III.Zmodyfikowanie modeli wprowadzając pojęcie Dostawcy



Tworzę klasę Supplier:

```
import javax.persistence.*;

@Entity
public class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    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;
    }
}
```

W klasie Product dodaję:

```java
    @ManyToOne
    private Supplier supplier;


    public Supplier getSupplier() {
        return supplier;
    }

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

A main:

```java
public static void main(String[] args) {
    sessionFactory = getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();

    Product product = new Product( productName: "Banany",  unitOnStock: 5);
    Supplier supplier = new Supplier( companyName: "Hurtownia",  street: "Czarnowiejska 2",  city: "Kraków");
    session.save(product);
    session.save(supplier);
    product.setSupplier(supplier);

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

Hibernate tworzy nowe tabele:

```
Hibernate:

    create table Product (
       id integer not null,
        ProductName varchar(255),
        UnitOnStock integer,
        supplier_id integer,
        primary key (id)
    )
Hibernate:

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

    alter table Product
       add constraint FK11uleikow9eaenolp88xnaudd
       foreign key (supplier_id)
       references Supplier
```

I wstawia do nich rekordy:

```
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, supplier_id, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
        */ insert
        into
            Supplier
            (city, companyName, street, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* update
        Product */ update
            Product
        set
            ProductName=?,
            UnitOnStock=?,
            supplier_id=?
        where
            id=?
```

Po tym nasza baza wygląda tak:

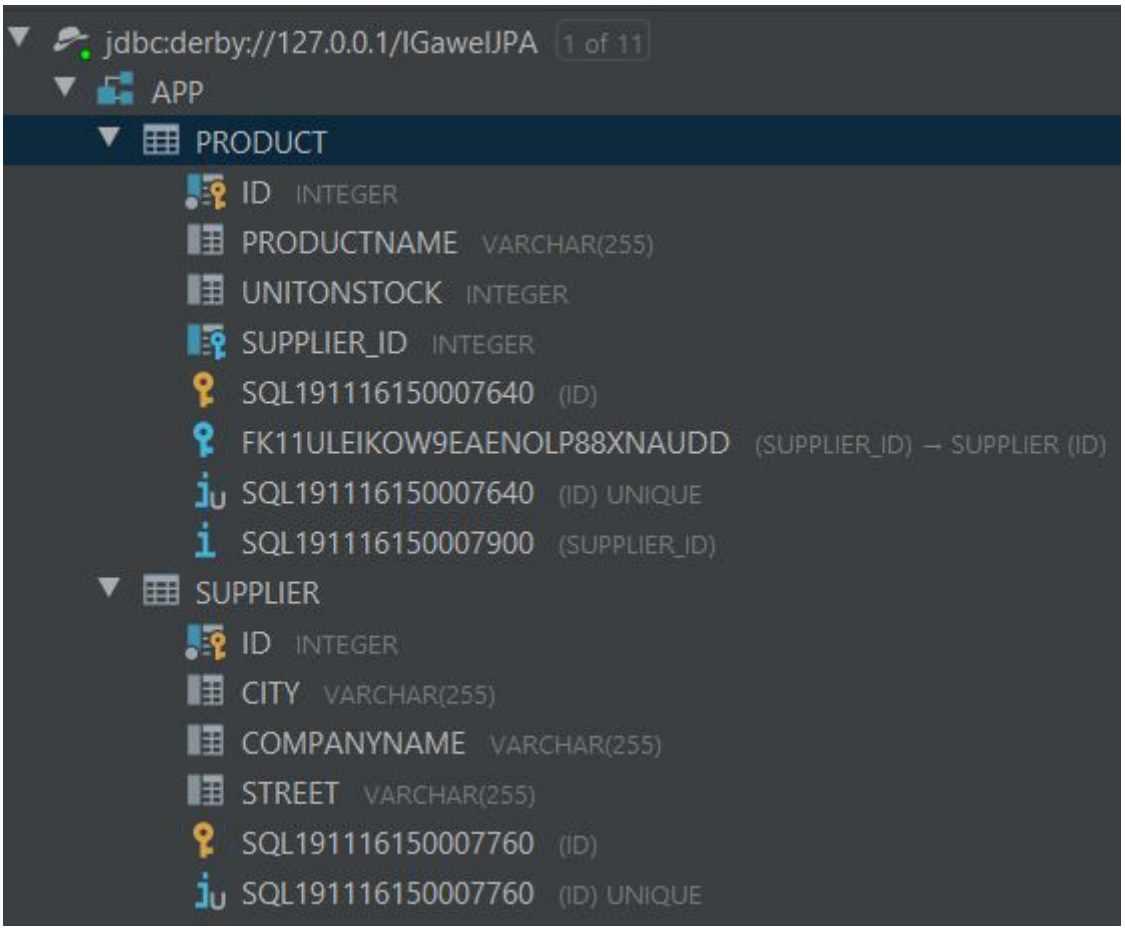


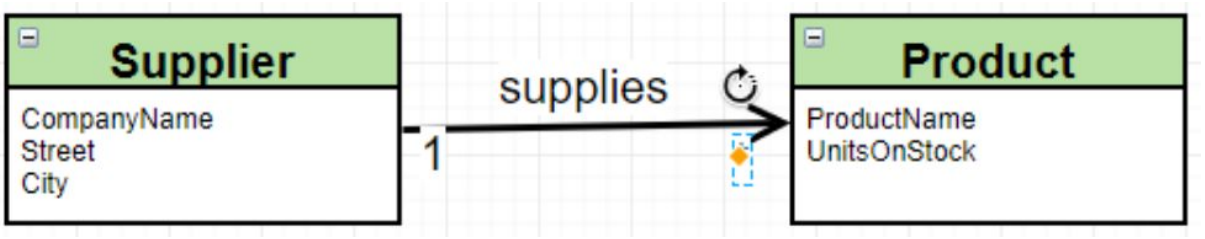
Tabela product

	ID	PRODUCTNAME	UNITONSTOCK	SUPPLIER_ID
1	1	Banany	5	2

Tabela supplier

	ID	CITY	COMPANYNAME	STREET
1	2	Kraków	Hurtownia	Czarnowiejska 2

IV.Odwróć relacji zgodnie z poniższym schematem



Wersja z tabelą łącznikową

W klasie Supplier dodaję:

```
@OneToMany
private List<Product> products;

public void add(Product product) {
    if (products == null) {
        products = new ArrayList<>();
    }
    products.add(product);
    product.setSupplier(this);
}
```

A w mainie:

```
public static void main(String[] args) {
    sessionFactory = getSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction tx = session.beginTransaction();

    Product product1 = new Product( productName: "Banany",  unitOnStock: 5);
    Product product2 = new Product( productName: "Jabłka",  unitOnStock: 3);
    Product product3 = new Product( productName: "Gruszki",  unitOnStock: 6);
    Supplier supplier = new Supplier( companyName: "Hurtownia",  street: "Czarnowiejska 2",  city: "Kraków");
    session.save(product1);
    session.save(product2);
    session.save(product3);
    session.save(supplier);
    supplier.add(product1);
    supplier.add(product2);
    supplier.add(product3);

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

Hibernate tworzy tabelę łącznikową:

```
Hibernate:

    create table Product (
       id integer not null,
        ProductName varchar(255),
        UnitOnStock integer,
        supplier_id integer,
        primary key (id)
    )
Hibernate:

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

    create table Supplier_Product (
       Supplier_id integer not null,
        products_id integer not null
    )
```

```
Hibernate:

    create table Supplier_Product (
       Supplier_id integer not null,
        products_id integer not null
    )
Hibernate:

    alter table Supplier_Product
       add constraint UK_cpp14o1ieorjjaagk8mkxwt6b unique (products_id)
Hibernate:

    alter table Product
       add constraint FK11uleikow9eaenolp88xnaudd
       foreign key (supplier_id)
       references Supplier
Hibernate:

    alter table Supplier_Product
       add constraint FKnsxquwfgqd1ktlok66v4wxo8m
       foreign key (products_id)
       references Product
Hibernate:

    alter table Supplier_Product
       add constraint FK1gam671f3qabh6mhfhkav4g7s
       foreign key (Supplier_id)
       references Supplier
```

I wstawia rekordy:

```
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, supplier_id, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, supplier_id, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, supplier_id, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
        */ insert
        into
            Supplier
            (city, companyName, street, id)
        values
            (?, ?, ?, ?)
```

```
Hibernate:
    /* update
        Product */ update
            Product
        set
            ProductName=?,
            UnitOnStock=?,
            supplier_id=?
        where
            id=?
Hibernate:
    /* update
        Product */ update
            Product
        set
            ProductName=?,
            UnitOnStock=?,
            supplier_id=?
        where
            id=?
Hibernate:
    /* update
        Product */ update
            Product
        set
            ProductName=?,
            UnitOnStock=?,
            supplier_id=?
        where
            id=?
```

```
Hibernate:
    /* insert collection
        row Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_id, products_id)
        values
            (?, ?)
Hibernate:
    /* insert collection
        row Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_id, products_id)
        values
            (?, ?)
Hibernate:
    /* insert collection
        row Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_id, products_id)
        values
            (?, ?)
```

Po tym możemy zaobserwować że powstała tabela łącznikowa SUPPLIER_PRODUCT

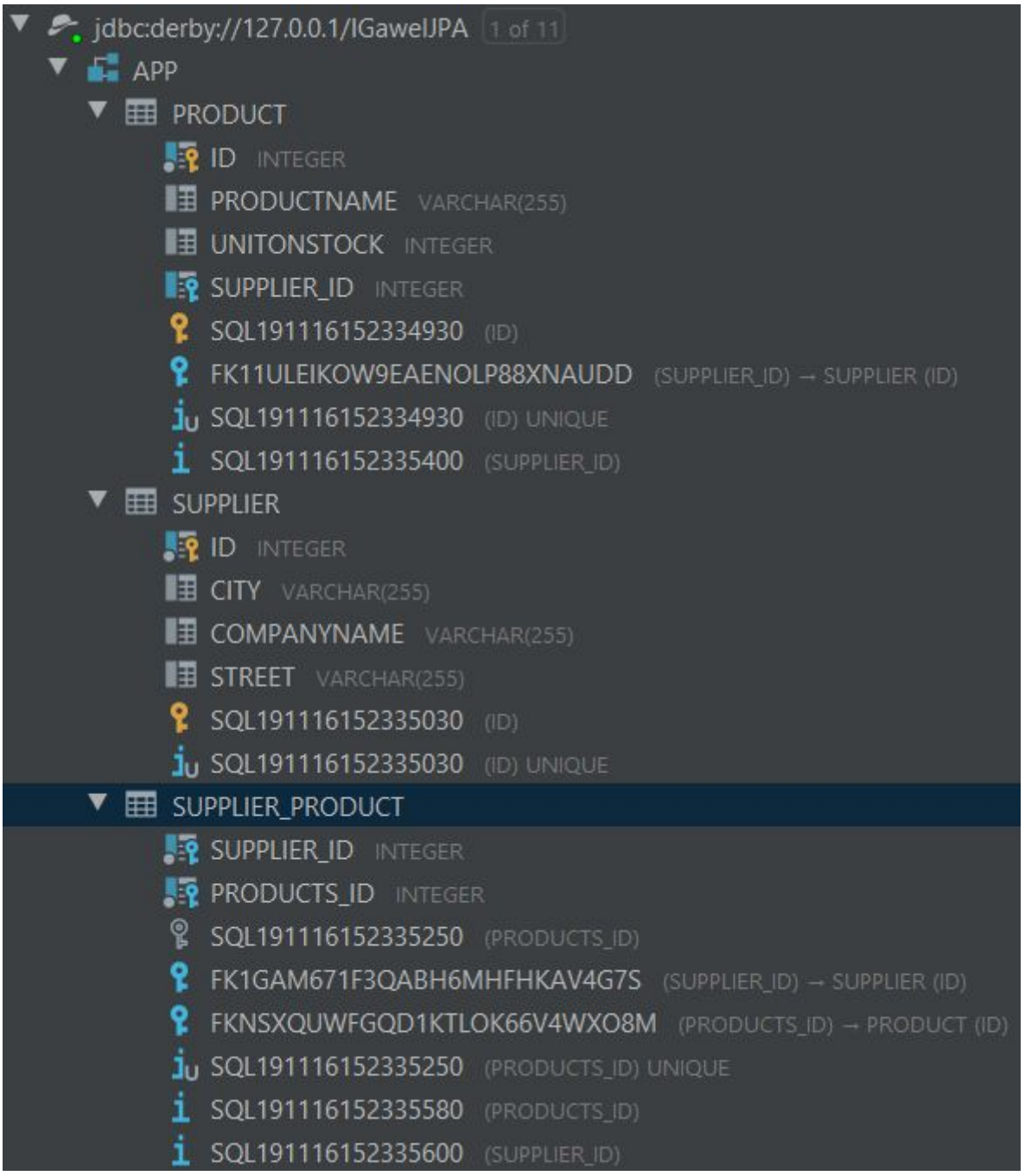


Tabela product:

	ID	PRODUCTNAME	UNITONSTOCK	SUPPLIER_ID
1	1	Banany	5	4
2	2	Jabłka	3	4
3	3	Gruszki	6	4

Tabela supplier:

	ID	CITY	COMPANYNAME	STREET
1	4	Kraków	Hurtownia	Czarnowiejska 2

Tabela łącznikowa:

	SUPPLIER_ID	PRODUCTS_ID
1	4	1
2	4	2
3	4	3

Wersja bez tabeli łącznikowej

Adnotacje @OneToMany uzupełniamy o @JoinColumn

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

W tym przypadku nie jest tworzona tabela łącznikowa, a do tabeli Product zostaje dodany klucz obcy:

```
Hibernate:

    create table Product (
       id integer not null,
        ProductName varchar(255),
        UnitOnStock integer,
        Supplier_Fk integer,
        primary key (id)
    )
Hibernate:

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

    alter table Product
       add constraint FKt8qmhs55061jp99akogvnrxjb
       foreign key (Supplier_Fk)
       references Supplier
```

Widzimy, że w bazie nie jest teraz tworzona tabela łącznikowa.

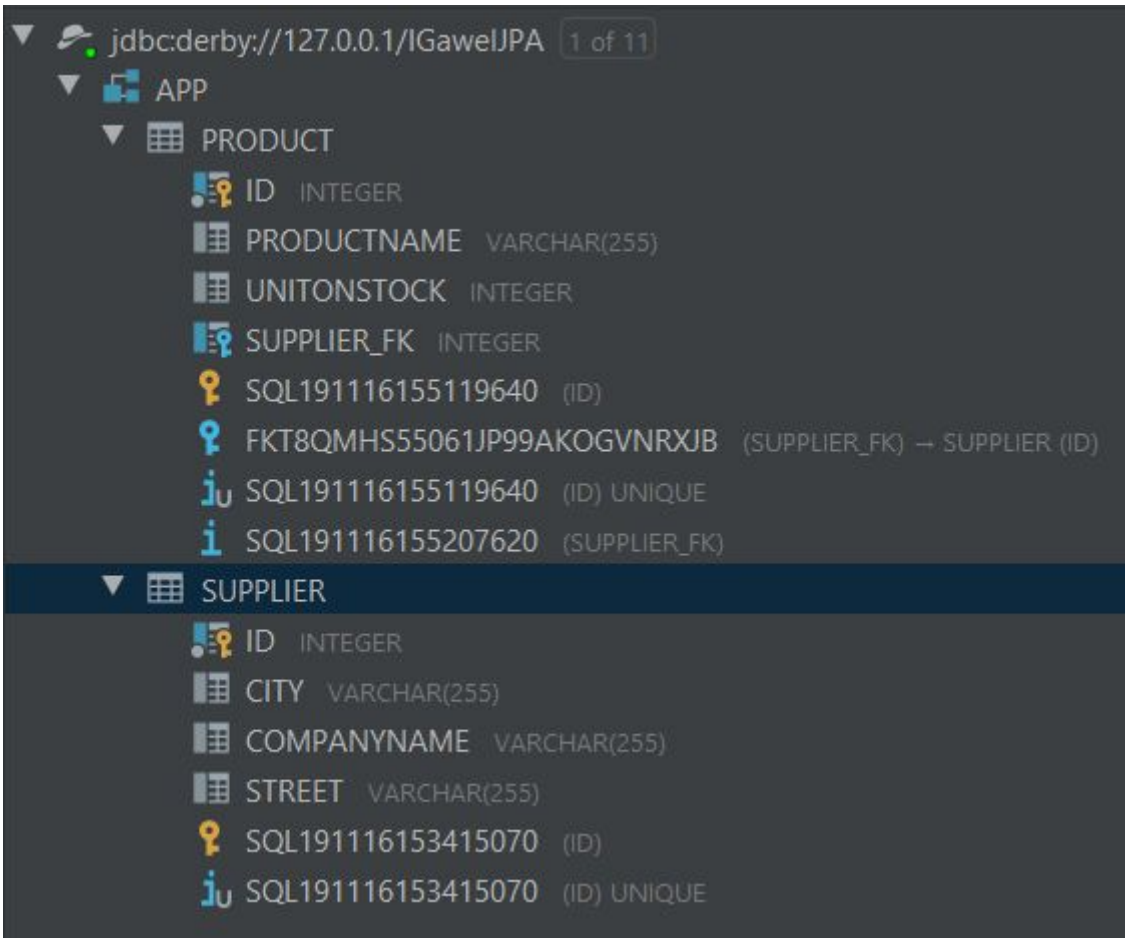


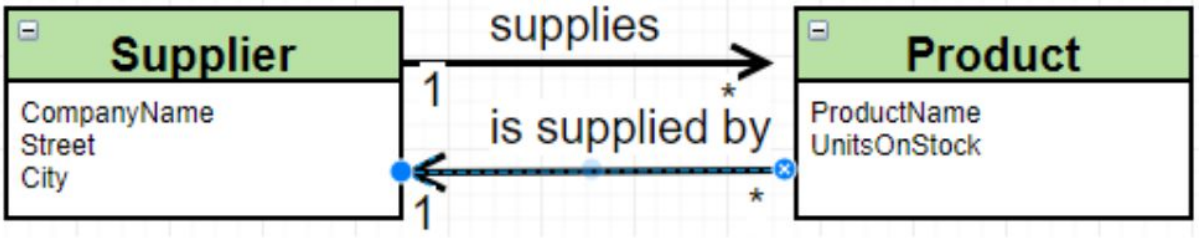
Tabela product:

	ID	PRODUCTNAME	UNITONSTOCK	SUPPLIER_FK
1	1	Banany	5	4
2	2	Jabłka	3	4
3	3	Gruszki	6	4

Tabela supplier:

	ID	CITY	COMPANYNAME	STREET
1	4	Kraków	Hurtownia	Czarnowiejska 2

V. Modelowanie relacji dwustronnej



W klasie Supplier:

```
@OneToMany(mappedBy = "supplier")
private List<Product> products;

public void add(Product product) {
    if (products == null) {
        products = new ArrayList<>();
    }
    products.add(product);
    product.setSupplier(this);
}
```

A w klasie Product:

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


public Supplier getSupplier() {
    return supplier;
}

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

Hibernate tworzy tabele:

```
Hibernate:

    create table Product (
       id integer not null,
        ProductName varchar(255),
        UnitOnStock integer,
        supplier_fk integer,
        primary key (id)
    )
Hibernate:

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

    alter table Product
       add constraint FKaneigr52a5j00iif9ts6daqjm
       foreign key (supplier_fk)
       references Supplier
```

I wstawia rekordy:

```
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, supplier_fk, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, supplier_fk, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitOnStock, supplier_fk, id)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
        */ insert
        into
            Supplier
            (city, companyName, street, id)
        values
            (?, ?, ?, ?)
```

```
Hibernate:
    /* update
        Product */ update
            Product
        set
            ProductName=?,
            UnitOnStock=?,
            supplier_fk=?
        where
            id=?
Hibernate:
    /* update
        Product */ update
            Product
        set
            ProductName=?,
            UnitOnStock=?,
            supplier_fk=?
        where
            id=?
Hibernate:
    /* update
        Product */ update
            Product
        set
            ProductName=?,
            UnitOnStock=?,
            supplier_fk=?
        where
            id=?
```

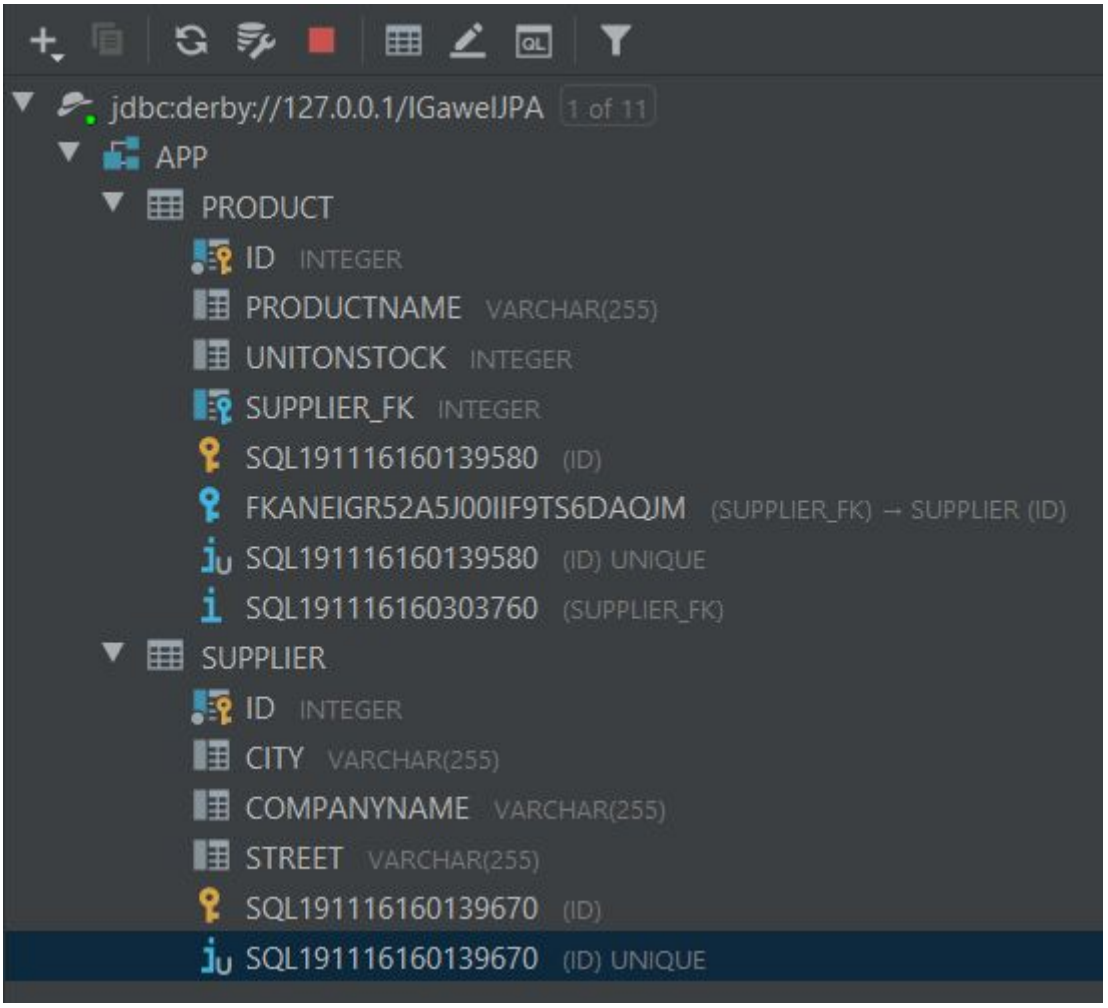


Tabela Supplier:

	ID	CITY	COMPANYNAME	STREET
1	4	Kraków	Hurtownia	Czarnowiejska 2

Tabela Product:

	ID	PRODUCTNAME	UNITONSTOCK	SUPPLIER_FK
1	1	Banany	5	4
2	2	Jabłka	3	4
3	3	Gruszki	6	4

VI.Dodaję klasę Category z property int CategoryID, String Name oraz listą produktow List<Product> Products

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

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

    private String Name ;

    @OneToMany(mappedBy = "category")
    public List<Product> Products;

    public void add(Product product) {
        if (Products == null) {
            Products = new ArrayList<>();
        }
        Products.add(product);
        product.setCategory(this);
    }

    public Category() {
    }

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

A do product dodaję:

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

public Category getCategory() {
    return category;
}

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

Otrzymujemy nową tabelę:

Tabela category:

	CATEGORYID	NAME
1	5	Owoce
2	6	Warzywa

Tabela produkt:

	ID	PRODUCTNAME	UNITONSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Banany	5	5	4
2	2	Jabłka	3	5	4
3	3	Marchewki	6	6	4

Pobranie produktów należących do danej kategorii:

```
int id = 5;
Category owoce = session.get(Category.class, id);
System.out.println(owoce.getProducts());
```

otrzymujemy:

```
[Product{ProductName='Banany', UnitOnStock=5}, Product{ProductName='Jabłka', UnitOnStock=3}]
```

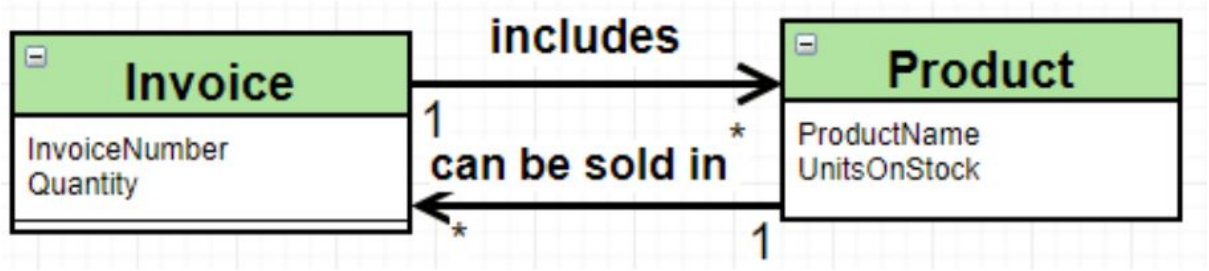
Wydobycie kategorii do której należy produkt:

```
int id = 3;
Product marchew = session.get(Product.class, id);
System.out.println(marchew.getCategory());
```

otrzymujemy:

```
Category{Name='Warzywa'}
```

VII. Modelowanie relacji wiele do wielu




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

    @ManyToMany(mappedBy = "canBeSoldIn")
    public List<Product> Includes;

    public Invoice() {
    }



    public void addProduct(Product product) {

        if (Includes == null) {
            Includes = new ArrayList<>();
        }

        Includes.add(product);

        List<Invoice> CanBeSoldIn = product.getCanBeSoldIn();
        if (CanBeSoldIn == null) {
            CanBeSoldIn = new ArrayList<>();
        }
        CanBeSoldIn.add(this);
        product.setCanBeSoldIn(CanBeSoldIn);
    }
```

Do klasy product dodaję:

```
@ManyToMany()
private List<Invoice> canBeSoldIn;

public List<Invoice> getCanBeSoldIn() {
    return canBeSoldIn;
}

public void setCanBeSoldIn(List<Invoice> canBeSoldIn) {
    this.canBeSoldIn = canBeSoldIn;
}
```

Produkty sprzedane w ramach wybranej faktury:

```
int invoice1Id = 7;
Invoice invoice = session.get(Invoice.class, invoice1Id);
System.out.println(invoice.getIncludes());
```

otrzymujemy:

```
[Product{ProductName='Banany', UnitOnStock=5}, Product{ProductName='Jabłka', UnitOnStock=3}]
```

Faktury w ramach których był sprzedany wybrany produkt:

```
int bananaId = 1;
Product product = session.get(Product.class, bananaId);
System.out.println(product.getCanBeSoldIn());
```

otrzymujemy:

```
[Invoice{InvoiceNumber=7, Includes=[Product{ProductName='Banany', UnitOnStock=5}, Product{ProductName='Jabłka', UnitOnStock=3}]},
Invoice{InvoiceNumber=8, Includes=[Product{ProductName='Banany', UnitOnStock=5}, Product{ProductName='Marchewki', UnitOnStock=6}]}]
```

IX. JPA

Plik persistance.xml

```xml
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
    <persistence-unit name="myDatabaseConfig"
                      transaction-type="RESOURCE_LOCAL">
        <properties>
            <property name="hibernate.connection.driver_class" value="org.apache.derby.jdbc.ClientDriver"/>
            <property name="hibernate.connection.url" value="jdbc:derby://127.0.0.1/IGawelJPA;create=true;"/>
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.hbm2ddl.auto" value="create" />
        </properties>
    </persistence-unit>
</persistence>
```

```java
public class MainJPA {
    public static void main(String[] args) {
        EntityManagerFactory emf =
                Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");;
        EntityManager em = emf.createEntityManager();
        EntityTransaction transaction = em.getTransaction();
        transaction.begin();

        Product product1 = new Product( productName: "Banany",  unitOnStock: 5);
        Product product2 = new Product( productName: "Jabłka",  unitOnStock: 3);
        Product product3 = new Product( productName: "Marchewki",  unitOnStock: 6);
        Supplier supplier = new Supplier( companyName: "Hurtownia",  street: "Czarnowiejska 2",  city: "Kraków");
        Category category1 = new Category( name: "Owoce");
        Category category2 = new Category( name: "Warzywa");
        em.persist(product1);
        em.persist(product2);
        em.persist(product3);
        em.persist(supplier);
        em.persist(category1);
        em.persist(category2);
        supplier.add(product1);
        supplier.add(product2);
        supplier.add(product3);
        category1.add(product1);
        category1.add(product2);
        category2.add(product3);

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

Category

	CATEGORYID	NAME
1	5	Owoce
2	6	Warzywa

Product

	ID	PRODUCTNAME	UNITONSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Banany	5	5	4
2	2	Jabłka	3	5	4
3	3	Marchewki	6	6	4

Supplier

	ID	CITY	COMPANYNAME	STREET
1	4	Kraków	Hurtownia	Czarnowiejska 2

X. Kaskady

W klasie Product

```
@ManyToMany(cascade = CascadeType.PERSIST)
private List<Invoice> canBeSoldIn;
```

natomiast w klasie Invoice:

```
@ManyToMany(mappedBy = "canBeSoldIn", cascade = CascadeType.PERSIST)
public List<Product> Includes;
```

Spowoduje to, utrwalenie faktur automatycznie spowoduje utrwalenie wszystkich powiézanych produktów, a utrwalenie produktu spowoduje automatyczne utrwalenie wszystkich faktur.

XI. Embedded class

a.Dodaj do modelu klase adres. „Wbuduj” ją do tabeli Dostawców.

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

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

    }
}
```

W klasie Suppier zamiast pól city i street tworzymy pole typu Adress.

```
@Embedded
private Address address;
```

```
Hibernate:

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

	ID	CITY	STREET	COMPANYNAME
1	4	Kraków	Czarnowiejska 2	Hurtownia

c.Zmdyfikuj model w taki sposób, że dane adresoweznajdują się w klasie dostawców. Zmapuj to do dwóch osobnych tabel.

Modyfikujemy klasę Supplier:

```
@Entity
@SecondaryTable(name = "Address")
public class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    private String companyName;

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

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

    @OneToMany(mappedBy = "supplier")
    private List<Product> products;

    public void add(Product product) {
```

```
Hibernate:

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

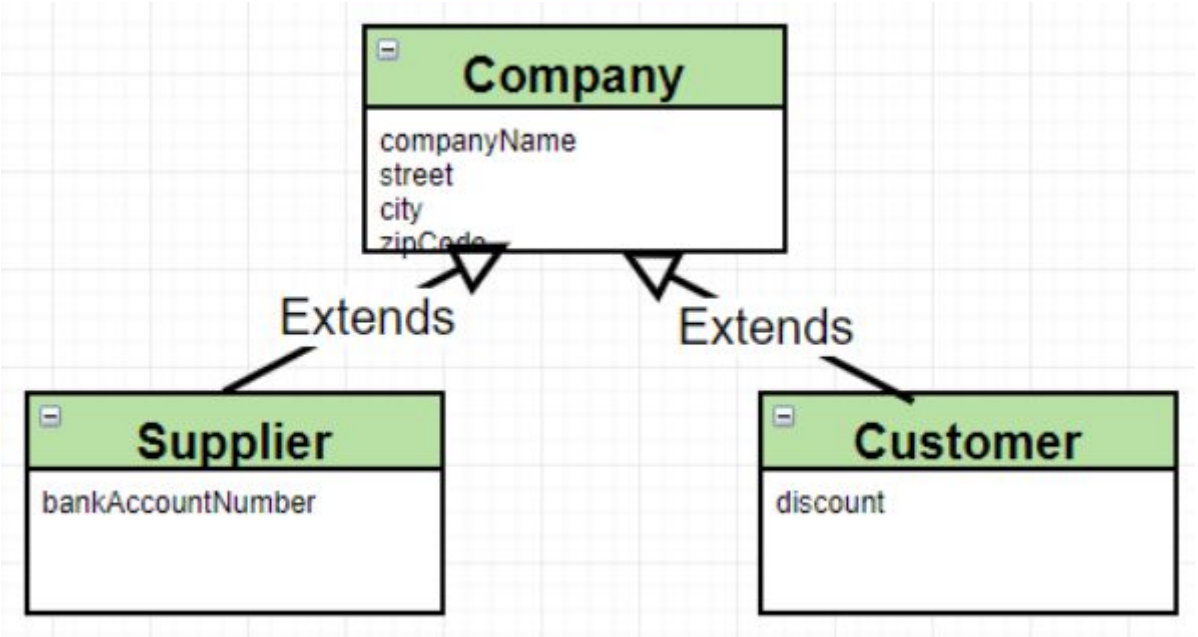
```
Hibernate:

    create table Supplier (
       id integer not null,
        companyName varchar(255),
        primary key (id)
    )
```

```
Hibernate:

    alter table Address
       add constraint FKj91l3o9613sfn00sb8yj237f2
       foreign key (id)
       references Supplier
```

	ID	COMPANYNAME
1	4	Hurtownia

	CITY	STREET	ID
1	Kraków	Czarnowiejska 2	4

XII.Dziedziczenie



InheritanceType.SINGLE_TABLE

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE")
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    protected int CompanyId;
    protected String CompanyName;
    protected String city;
    protected String street;
    protected String zipCode;

    public Company() {
    }
}
```

```java
@Entity
@DiscriminatorValue(value = "Customer")
public class Customer extends Company {
    private double discount;

    public Customer(){

    }

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

```java
@Entity
@DiscriminatorValue(value = "Supplier")
public class Supplier extends Company {
    private String bankAccountNumber;

    @OneToMany(mappedBy = "supplier")
    private List<Product> products;

    public void add(Product product) {
        if (products == null) {
            products = new ArrayList<>();
        }
        products.add(product);
        product.setSupplier(this);
    }

    public Supplier() {
```

```
Hibernate:

    create table Company (
        TYPE varchar(31) not null,
         CompanyId integer not null,
         CompanyName varchar(255),
         city varchar(255),
         street varchar(255),
         zipCode varchar(255),
         bankAccountNumber varchar(255),
         discount double,
         primary key (CompanyId)
    )
```

	TYPE	COMPANYID	COMPANYNAME	CITY	STREET	ZIPCODE	BANKACCOUNTNUMBER	DISCOUNT
1	Customer	1	Klient1	Kraków	Kawiory 1	11-111	<null>	0.1
2	Customer	2	Klient2	Kraków	Kawiory 2	11-111	<null>	0.05
3	Customer	3	Klient3	Kraków	Kawiory 3	11-111	<null>	0.15
4	Supplier	4	Dostawca1	Kraków	Czarnowiejska 3	22-222	PL61109010140000071219812874	<null>
5	Supplier	5	Dostawca2	Kraków	Czarnowiejska 4	22-222	PL61109010140000071219812885	<null>

InheritanceType.JOINED

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
Hibernate:

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

```
Hibernate:

    create table Supplier (
       bankAccountNumber varchar(255),
        CompanyId integer not null,
        primary key (CompanyId)
    )
Hibernate:

    alter table Customer
       add constraint FKrr2pxj29e8fmtv5s31r8fkx7k
       foreign key (CompanyId)
       references Company
```

Tabela Company:

	COMPANYID	COMPANYNAME	CITY	STREET	ZIPCODE
1	1	Klient1	Kraków	Kawiory 1	11-111
2	2	Klient2	Kraków	Kawiory 2	11-111
3	3	Klient3	Kraków	Kawiory 3	11-111
4	4	Dostawca1	Kraków	Czarnowiejska 3	22-222
5	5	Dostawca2	Kraków	Czarnowiejska 4	22-222

Tabela Customer:

	DISCOUNT	COMPANYID
1	0.1	1
2	0.05	2
3	0.15	3

Tabela Supplier:

	BANKACCOUNTNUMBER	COMPANYID
1	PL61109010140000071219812874	4
2	PL61109010140000071219812885	5

InheritanceType.TABLE_PER_CLASS

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
Hibernate:

    create table Company (
       CompanyId integer not null,
        CompanyName varchar(255),
        city varchar(255),
        street varchar(255),
        zipCode varchar(255),
        primary key (CompanyId)
    )
Hibernate:

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

```
Hibernate:

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

COMPANYID	COMPANYNAME	CITY	STREET	ZIPCODE

	COMPANYID	COMPANYNAME	CITY	STREET	ZIPCODE	DISCOUNT
1	1	Klient1	Kraków	Kawiory 1	11-111	0.1
2	2	Klient2	Kraków	Kawiory 2	11-111	0.05
3	3	Klient3	Kraków	Kawiory 3	11-111	0.15

	COMPANYID	COMPANYNAME	CITY	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	4	Dostawca1	Kraków	Czarnowiejska 3	22-222	PL61109010140000071219812874
2	5	Dostawca2	Kraków	Czarnowiejska 4	22-222	PL61109010140000071219812885

XIII.Aplikacja do zamawiania produktów

Tworzę nową klasę Order

```
@Entity
@Table(name="orders")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int OrderID;

    @ManyToOne()
    @JoinColumn(name = "customer_fk")
    private Customer customer;

    @ManyToMany
    private List<Product> products;

    public void addProduct(Product product) {

        if (products == null) {
            products = new ArrayList<>();
        }
        products.add(product);

        List<Order> orders = product.getOrders();
        if (orders == null) {
            orders = new ArrayList<>();
        }
        orders.add(this);
        product.setOrders(orders);


    }
```

```java
Order order = new Order();
System.out.println("Podaj nazwię swojej firmy: ");
Scanner inputScanner = new Scanner(System.in);
try{
    String companyName = inputScanner.nextLine();
    Query query = session.createQuery( s: "from Company where companyname = :name");
    query.setParameter( s: "name", companyName);
    Company klient = (Company) query.list().get(0);
    order.setCustomer(session.get(Customer.class, klient.CompanyId));
}catch (IndexOutOfBoundsException e){
    System.out.println("Nie posiadamy takiego takiego produktu");
}
boolean flag = true;
while (flag){
    System.out.println("Jaki produkt chcesz zamówić: ");
    try{
        String prodname = inputScanner.nextLine();
        Query query = session.createQuery( s: "from Product where productname = :name");
        query.setParameter( s: "name", prodname);
        Product prod = (Product) query.list().get(0);
        order.addProduct(prod);
    }catch (IndexOutOfBoundsException e){
        System.out.println("Nie posiadamy takiego takiego produktu");
    }

    System.out.println("Coś jeszcze?");
    inputScanner = new Scanner(System.in);
    String f = inputScanner.nextLine();
    if (!f.equals("Tak")) flag = false;
}
System.out.println("Przyjęto zamówienie");
session.save(order);
```

Rezultat działania programu

```
Podaj nazwię swojej firmy:
Klient1
Jaki produkt chcesz zamówić:
Banany
Coś jeszcze?
Tak
Jaki produkt chcesz zamówić:
Jabłka
Coś jeszcze?
Nie
Przyjęto zamówienie
```

Końcowy schemat bazy danych: