Bazy danych

Hibernate, JPA

Michał Szkarłat
czwartek 14:20, 07.11.19, **laboratorium 3**

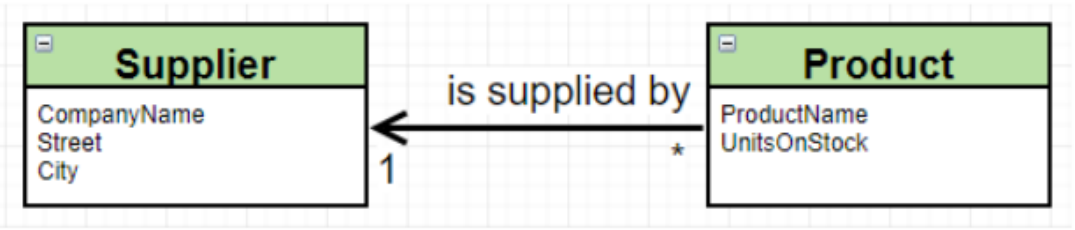
Spis treści

1. Praca na laboratoriach....3
2. Dalsza część instrukcji....3
  2.1 Odwróć relacje zgodnie z poniższym schematem....3
    2.1.1 Bez tabeli łącznikowej....4
    2.1.2 Z tabelą łącznikową....4
  2.3 Zamodeluj relacje dwustronną....5
  2.4 Dodaj klasę Category....7
  2.5 Zamodeluj relacje wiele-do-wielu....8
3. Zadanie domowe....10
  3.1 JPA....10
  3.2 Kaskady....11
  3.3 Embedded class....13
  3.4 Dziedziczenie....14
    3.4.1 SINGLE_TABLE....16
    3.4.2 TABLE_PER_CLASS....16
    3.4.3 JOINED....17
4. Aplikacja webowa....17

1. Praca na laboratoriach

Na zajęciach doszedłem do punktu III (włącznie):

III. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej



a. Stworz nowego dostawce.

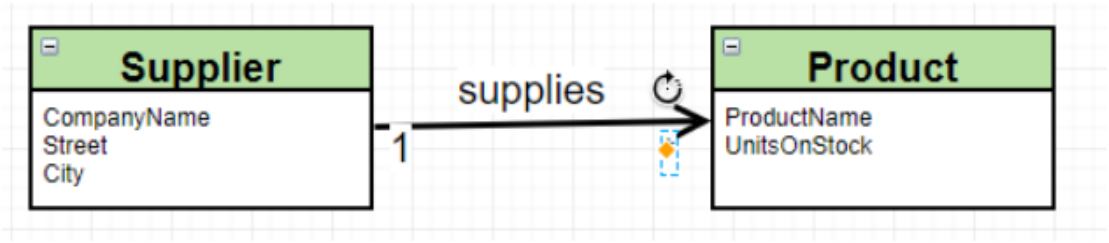
b. Znajdz poprzednio wprowadzony produkt i ustaw jego dostawce na właśnie dodanego.

c. **Udokumentuj wykonane kroki oraz uzyskany rezultat (ogi wywołań sqlowych,describe table/diagram z datagrip, select * from....)**

Dalsza część sprawozdania będzie przedstawiała wykonanie kolejnych podpunktów, a finalnie zadania domowego.

2. Dalsza część instrukcji

2.1 Odwróć relacje zgodnie z poniższym schematem



a. Zamodeluj powyższe w dwoch wariantach „z" i „bez" tabeli łącznikowej

b. Stworz kilka produktow

c. Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę

d. **Udokumentuj wykonane kroki oraz uzyskane rezultaty w obu wariantach (logi wywołań sqlowych, describe table/diagram z datagrip, select * from....)**

2.1.1 Bez tabeli łącznikowej

```java
import java.util.Set;

@Table(name = "Suppliers")
@Entity
public class Supplier {

    @Id
    @GeneratedValue
    private int supplierId;

    @Column(name = "CompanyName")
    private String companyName;

    @Column(name = "Street")
    private String street;

    @Column(name = "City")
    private String city;

    // zadanie I.IV
    @OneToMany
    @JoinColumn(name = "SuppliedBy")
    private Set<Product> products = new HashSet<>();

    public Supplier() {

    }

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

    // zadanie I.IV
    public void addSuppliedProduct(Product product) {
        products.add(product);
    }
}
```

```
values
    next value for hibernate_sequence
Hibernate:
    /* insert model.Product
        */ insert
        into
            Products
            (ProductName, SuppliedBy, UnitsOnStock, productId)
        values
            (?, ?, ?, ?)
New supplier name: Auchan
City: Krakow
Street: dluga
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    /* insert model.Supplier
        */ insert
        into
            Suppliers
            (City, CompanyName, Street, supplierId)
        values
            (?, ?, ?, ?)
Hibernate:
    /* create one-to-many row model.Supplier.products */ update
        Products
    set
        SuppliedBy=?
    where
        productId=?
Hibernate:
    /* create one-to-many row model.Supplier.products */ update
        Products
    set
        SuppliedBy=?
    where
        productId=?

Process finished with exit code 0
```

```
1 wiersz wybrany
ij(CONNECTION1)> select * from products;
PRODUCTID  |PRODUCTNAME                                                                                          |UNITSONSTOCK          |SUPPLIEDBY
--------------------------------------------------------------------------------------------------------------------------------------------------------------
1          |Cebula                                                                                               |5.0                   |NULL
2          |Mleko                                                                                                |6.0                   |4
3          |Chleb                                                                                                |4.0                   |4
```

Implementacja modelu dostawcy, logi zapytań sql oraz widok w bazie

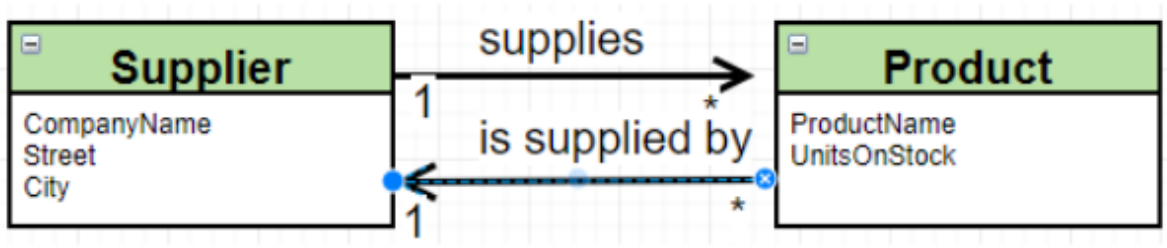
2.1.2 Z tabelą łącznikową

```java
// zadanie I.IV
@OneToMany
@JoinTable(
        name = "SuppliedBy",
        joinColumns = {@JoinColumn(name = "SupplierId")},
        inverseJoinColumns = {@JoinColumn(name = "ProductID")}
)
private Set<Product> products = new HashSet<>();
```

Zmianie uległa jedynie definicja adnotacji. W bazie powstała nowa tabela „SuppliedBy”:

```
2 wierszy wybranych
ij(CONNECTION1)> select * from suppliedby;
SUPPLIER_ID|PRODUCT_ID
-----------------------
4          |2
4          |3
```

2.3 Zamodeluj relacje dwustronną



a. Tradycyjnie: Stworz kilka produktow

b. Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę (pamiętaj o poprawnej obsłudze dwustronności relacji)

Polegało to na połączeniu dwóch poprzednich definicji (punkt III oraz IV) oraz modyfikacje poszczególnych setterów:

```java
@Table(name = "Products")
@Entity
public class Product {

    @Id
    @GeneratedValue
    private int productId;

    @Column(name = "ProductName")
    private String productName;

    @Column(name = "UnitsOnStock")
    private double unitsOnStock;

    // zadanie I.III
    @ManyToOne
    @JoinColumn(name = "SuppliedBy")
    private Supplier suppliedBy;

    public Product() {

    }

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

    // zadanie I.III
    public void setSuppliedBy(Supplier suppliedBy) {
        this.suppliedBy = suppliedBy;
        // zadanie I.V
        if (!suppliedBy.getProducts().contains(this)) {
            suppliedBy.addSuppliedProduct(this);
        }
    }
}
```

```java
@Entity
public class Supplier {

    @Id
    @GeneratedValue
    private int supplierId;

    @Column(name = "CompanyName")
    private String companyName;

    @Column(name = "Street")
    private String street;

    @Column(name = "City")
    private String city;

    // zadanie I.IV
    @OneToMany
    // z tabelą łącznikową
//    @JoinTable(
//            name = "SuppliedBy",
//            joinColumns = {@JoinColumn(name = "Supplier_Id")},
//            inverseJoinColumns = {@JoinColumn(name = "Product_ID")}
//    )
    @JoinTable(name = "SuppliedBy")
    private Set<Product> products = new HashSet<>();

    public Supplier() {

    }

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

    // zadanie I.IV
    public void addSuppliedProduct(Product product) {
        products.add(product);
        product.setSuppliedBy(this);
    }

    // zadanie I.V
    public Set<Product> getProducts() {
        return products;
    }
}
```

Ze względu na podobne działanie, kod aplikacji sprawdzającej został taki sam:

```java
// zadanie I.III
public Product updateProduct(Product product, Supplier supplier) {
    Session session = mainSession.getSession();
    Transaction tx = session.beginTransaction();

    product.setSuppliedBy(supplier);

    session.save(product);
    tx.commit();
    return product;
}

// zadanie I.IV
public Supplier updateSupplier(Supplier supplier, Product product) {
    Session session = mainSession.getSession();
    Transaction tx = session.beginTransaction();

    supplier.addSuppliedProduct(product);

    session.save(product);
    tx.commit();
    return supplier;
}
```

```
        operation.updateProduct(searchFirstProduct, firstSupplier);

    // zadanie I.IV i zadanie I.V
    System.out.print("Second product name: ");
    String secondProductName = inputScanner.nextLine();
    System.out.print("Amount: ");
    double secondProductAmount =
            Double.parseDouble(inputScanner.nextLine());
    Product secondProduct =
            operation.insertProduct(secondProductName, secondProductAmount);

    System.out.print("Third product name: ");
    String thirdProductName = inputScanner.nextLine();
    System.out.print("Amount: ");
    double thirdProductAmount = Double.parseDouble(inputScanner.nextLine());
    Product thirdProduct =
            operation.insertProduct(thirdProductName, thirdProductAmount);

    System.out.print("New supplier name: ");
    String secondSupplierName = inputScanner.nextLine();
    System.out.print("City: ");
    String secondSupplierCity = inputScanner.nextLine();
    System.out.print("Street: ");
    String secondSupplierStreet = inputScanner.nextLine();
    Supplier secondSupplier = operation
            .insertSupplier(secondSupplierName, secondSupplierCity,
                    secondSupplierStreet);

    operation.updateSupplier(secondSupplier, secondProduct);
    operation.updateSupplier(secondSupplier, thirdProduct);
}
```

Implementacja maina sprawdzającego działanie punktów IV oraz V

```
1  Hibernate: /* from Product */ select product0_.productId as productI1_0_, product0_.ProductN
   ame as ProductN2_0_, product0_.SuppliedBy as Supplied4_0_, product0_.UnitsOnStock as UnitsOn
   S3_0_ from Products product0_ executing: from Supplier Hibernate: /* 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_
2   First product name:
3  Woda Amount: 1
4   Hibernate: values next value for hibernate_sequence Hibernate: /* insert model.Product */ i
   nsert into Products (ProductName, SuppliedBy, UnitsOnStock, productId) values (?, ?, ?, ?)
5  Second product name: Wino
6  Amount: 2
7  Hibernate: values next value for hibernate_sequence
8  Hibernate: /* insert model.Product */ insert into Products (ProductName, SuppliedBy, UnitsOn
   Stock, productId) values (?, ?, ?, ?)
9  Third product name: Paczka
10 Amount: 3
11 Hibernate: values next value for hibernate_sequence Hibernate: /* insert model.Product */ in
   sert into Products (ProductName, SuppliedBy, UnitsOnStock, productId) values (?, ?, ?, ?)
12 New supplier name: Lewiatan
13 City: AGH
14 Street: AGH
15 Hibernate: values next value for hibernate_sequence Hibernate: /* insert model.Supplier */ i
   nsert into Suppliers (City, CompanyName, Street, supplierId) values (?, ?, ?, ?)
16 Hibernate: /* update model.Product */ update Products set ProductName=?, SuppliedBy=?, Units
   OnStock=? where productId=?
17 Hibernate: /* insert collection row model.Supplier.products */ insert into SuppliedBy (Suppl
   ier_supplierId, products_productId) values (?, ?) Hibernate: /* update model.Product */ upda
   te Products set ProductName=?, SuppliedBy=?, UnitsOnStock=? where productId=?
18 Hibernate: /* insert collection row model.Supplier.products */ insert into SuppliedBy (Suppl
   ier_supplierId, products_productId) values (?, ?)
```

Logi z Hibernate, skopiowane do notatnika i sformatowane, aby nie zawierały dożo miejsca.

2.4 Dodaj klasę Category

a. Zmodyfikuj produkty dodając wskazanie na kategorie do której należy.

b. Stworz kilka produktow i kilka kategorii

c. Dodaj kilka produktów do wybranej kategorii

d. Wydobądź produkty z wybranej kategorii oraz kategorię do której należy wybrany produkt

e. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

```java
@Entity
public class Category {

    @Id
    @GeneratedValue
    private int CategoryId;

    @Column(name = "Name")
    private String name;

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

    public Category() {
    }

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

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

    // zadanie VI
    public void addProduct(Product product) {
        products.add(product);
        product.setCategory(this);
    }
}
```

```java
    private Supplier suppliedBy;

    // zadanie VI
    @ManyToOne
    private Category category;

    // zadanei VI
    public Category getCategory() {
        return category;
    }
```

```
|UNITSONSTOCK           |CATEGORY_C
-----------------------------------------
|1.0                    |4
|1.0                    |4
|1.0                    |4
```

Implementacja modelu kategorii.

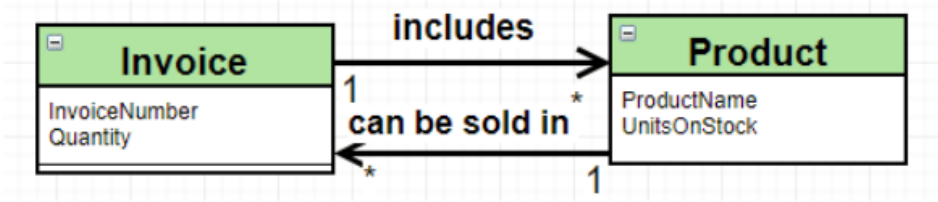
```java
public Product addCategoryToProduct(Product product, Category category) {
    Session session = mainSession.getSession();
    Transaction tx = session.beginTransaction();

    product.setCategory(category);

    session.save(product);
    tx.commit();
    return product;
}

public List<Product> getProducts(Category category) {
    CriteriaBuilder builder = mainSession.getSession().getCriteriaBuilder()
    CriteriaQuery<Product> query = builder.createQuery(Product.class);

    Root<Product> root = query.from(Product.class);
    Predicate pWhereCategory =
            builder.equal(root.get("category"), category);

    TypedQuery<Product> typedQuery = mainSession.getSession().createQuery(
            query.select(root).where(pWhereCategory)
    );
    return typedQuery.getResultList();
}
```

```
Hibernate:
    /* select
        generatedAlias0
    from
        Product as generatedAlias0
    where
        generatedAlias0.category=:param0 */ select
            product0_.productId as productI1_2_,
            product0_.category_CategoryId as category4_2_,
            product0_.ProductName as ProductN2_2_,
            product0_.SuppliedBy as Supplied5_2_,
            product0_.UnitsOnStock as UnitsOnS3_2_
        from
            Products product0_
        where
            product0_.category_CategoryId=?
Product list:
model.Product@f8f56b9
model.Product@29852487
model.Product@260f2144
```

Implementacja zapytania zwracającego listę produktów danej kategorii wraz z wykonaniem

2.5 Zamodeluj relacje wiele-do-wielu

VII. Zamodeluj relacje wiele-do-wielu, jak poniżej:



a. Stórz kilka produktów I “sprzedaj” je na kilku transakcjach.

b. Pokaż produkty sprzedane w ramach wybranej faktury/transakcji

c. Pokaż faktury w ramach których był sprzedany wybrany produkt

d. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

```java
@Entity
public class Invoice {

    @Id
    @GeneratedValue
    @Column(name = "InvoiceNumber")
    private int invoiceNumber;

    @Column(name = "Quantity")
    private int quantity;

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

    public Invoice() {
    }

    public Invoice(int quantity) {
        this.quantity = 0;
    }

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

    public int getInvoiceNumber() { return invoiceNumber; }

    public int getQuantity() { return quantity; }

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

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

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

// zadanie VII
public Set<Invoice> getCanBeSoldIn() {
    return canBeSoldIn;
}

public String getProductName() {
    return productName;
}
```

```
INVOICENUM&|QUANTITY
-----------------------
1          |9
2          |7
```

```
SYSIBM                         |SYSDUMMY1                       |
APP                            |CATEGORY                        |
APP                            |CATEGORY_PRODUCTS               |
APP                            |INVOICE                         |
APP                            |INVOICE_PRODUCTS                |
APP                            |PRODUCTS                        |
APP                            |SUPPLIEDBY                      |
APP                            |SUPPLIERS                       |
```

Implementacja modelu relacji wiele do wielu, zrzut z bazy danych

```
    public Invoice insertInvoice(Invoice inv) {
        Session session = mainSession.getSession();
        Transaction tx = session.beginTransaction();

        session.save(inv);
        tx.commit();
        return inv;
    }

    public void addProductToInvoice(Invoice inv, Product product,
                                    int quantity) {
        Session session = mainSession.getSession();
        Transaction tx = session.beginTransaction();

        inv.addProduct(product, quantity);

        session.save(inv);
        tx.commit();
    }
}
```

Operacja dodawania produktu do faktury

```
    Invoice inv1 = operation.insertInvoice(new Invoice());
    Invoice inv2 = operation.insertInvoice(new Invoice());
    Product p1 = operation.insertProduct(p1Name, p1Amount);
    Product p2 = operation.insertProduct(p2Name, p2Amount);
    Product p3 = operation.insertProduct(p2Name, p3Amount);

    operation.addProductToInvoice(inv1, p1,  quantity: 5);
    operation.addProductToInvoice(inv1, p2,  quantity: 4);
    operation.addProductToInvoice(inv2, p3,  quantity: 7);

    System.out.println("Products for invoice inv1:");
    for (Product prod : inv1.getProducts()) {
        System.out.println(prod.getProductName());
    }
}
```

Poprawna implementacja pobierania produktów z faktury

```
            where
                InvoiceNumber=?
Hibernate:
    /* insert collection
        row model.Invoice.includesProducts */ insert
        into
            Invoice_Products
            (canBeSoldIn_InvoiceNumber, includesProducts_productId)
        values
            (?, ?)
Hibernate:
    /* update
        model.Invoice */ update
            Invoice
        set
            Quantity=?
        where
            InvoiceNumber=?
Hibernate:
    /* insert collection
        row model.Invoice.includesProducts */ insert
        into
            Invoice_Products
            (canBeSoldIn_InvoiceNumber, includesProducts_productId)
        values
            (?, ?)
Products for invoice inv1:
p1
p2

Process finished with exit code 0
```

Logi Hibernate dla pobierania produktów

3. Zadanie domowe

3.1 JPA

```
public static EntityManagerFactory entityManagerFactory;

public static void main(String... args) {
    EntityManager manager = getEntityManager();

    EntityTransaction transaction = manager.getTransaction();
    transaction.begin();

    Product p1 = new Product( productName: "Ziemniaki",  unitsOnStock: 100);
    Product p2 = new Product( productName: "Pietruszka",  unitsOnStock: 200);
    Product p3 = new Product( productName: "Daktyle",  unitsOnStock: 213);
    Product p4 = new Product( productName: "Gruszka",  unitsOnStock: 7);

    Category c1 = new Category( name: "Warzywa");
    Category c2 = new Category( name: "Owoce");

    c1.addProduct(p1);
    c1.addProduct(p2);
    c2.addProduct(p3);
    c2.addProduct(p4);

    manager.persist(p1);
    manager.persist(p2);
    manager.persist(p3);
    manager.persist(p4);
    manager.persist(c1);
    manager.persist(c2);
    transaction.commit();

    transaction = manager.getTransaction();
    transaction.begin();
```

```
    System.out.println("Categories: ");
    for (Category category : manager
            .createQuery( s: "from Category", Category.class)
            .getResultList()) {
        System.out.println(
                " " + category.getCategoryId() + " " + category.getName());
        for (Product product : category.getProducts()) {
            System.out.println(" " + product.getProductName());
        }
    }

    System.out.println("Products: ");
    for (Product p :
            (List<Product>) manager.createQuery(
                    s: "SELECT p from Product p LEFT JOIN FETCH p" +
                            ".category").getResultList()) {

        if (p.getCategory() != null) {
            System.out.println(String.format("Product %s is in category %s",
                    p.getProductName(),
                    p.getCategory().getName()));
        } else {
            System.out.println(
                    String.format("Product %s has no category assigned",
                            p.getProductName()));
        }
    }
    transaction.commit();

    manager.close();
    entityManagerFactory.close();
}

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

Stworzony main wraz obsługą przypadku kategorii i produktów dla JPA

IX. JPA

a. Stwórz nowego maina w którym zrobisz to samo co w punkcie VI ale z wykorzystaniem JPA

b. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

```
            on product0_.category_CategoryId=category1_.CategoryId
Product p1 has no category assigned
Product p2 has no category assigned
Product p2 has no category assigned
Product Ziemniaki is in category Warzywa
Product Pietruszka is in category Warzywa
Product Daktyle is in category Owoce
Product Gruszka is in category Owoce
```

Rezultat z wykonaniu metody main→ porawnie pobrane dane z tabeli

PRODUCTID	PRODUCTNAME
3	p1
4	p2
5	p2
8	Ziemniaki
9	Pietruszka
10	Daktyle
11	Gruszka

	UNITSONSTOCK	CATEGORY_C&	SUPPLIEDBY
	10.0	NULL	NULL
	10.0	NULL	NULL
	10.0	NULL	NULL
	100.0	12	NULL
	200.0	12	NULL
	213.0	13	NULL
	7.0	13	NULL

Stan bazy danych po wykonaniu metody main

3.2 Kaskady

X. Kaskady

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

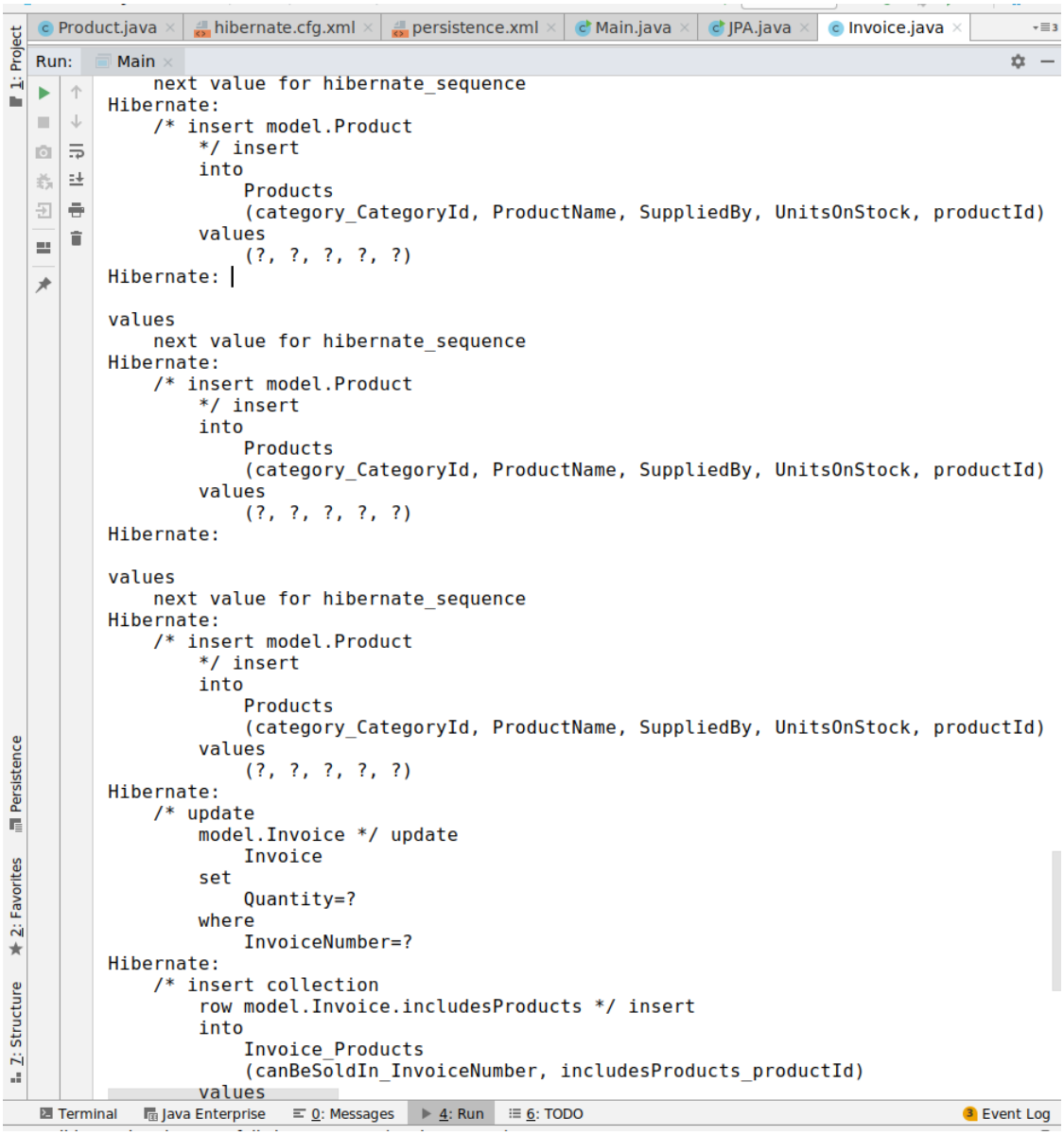
b. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

Głowna zmiana polegała na dodaniu odpowiedniego atrybutu w adnotacji łączącej ze sobą tabele produktów i faktur:

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

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

Adnotacje umożliwiające kaskadowe tworzenie faktur



Hibernate łączy kaskadowo operację z powiązanymi produktami i je utrzymuje (cascadetype.PERSIST).

3.3 Embedded class

XI. Embedded class

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

b. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

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

d. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

```java
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;
    }
}
```

```java
    // zadanie XI
    // replace with Address
//    @Column(name = "street")
//    private String street;

    // replace with Address
//    @Column(name = "city")
//    private String city;

    @Embedded
    @AttributeOverrides({
            @AttributeOverride(name = "street", column = @Column(name = "Street")),
            @AttributeOverride(name = "city", column = @Column(name = "City"))
    })
    private Address address;
```

Dodanie do modelu wbudowanej tabeli Address

```java
public Supplier insertSupplier(String supplierName, String city,
                               String street) {
    Session session = mainSession.getSession();
    Transaction tx = session.beginTransaction();

    // zadanie XI
    Address address = new Address(city, street);

    Supplier supplier = new Supplier(supplierName, address);
    session.save(supplier);

    tx.commit();
    return supplier;
}
```

Modyfikacja dodawania dostawy do bazy → używa wbudowanej klasy Address

```
values
    next value for hibernate_sequ
Hibernate:
    /* insert model.Supplier
        */ insert
        into
            Suppliers
            (City, Street, CompanyName, supplierId)
        values
            (?, ?, ?, ?)
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    /* insert model.Supplier
        */ insert
        into
            Suppliers
            (City, Street, CompanyName, supplierId)
        values
            (?, ?, ?, ?)

Process finished with exit code 0
```

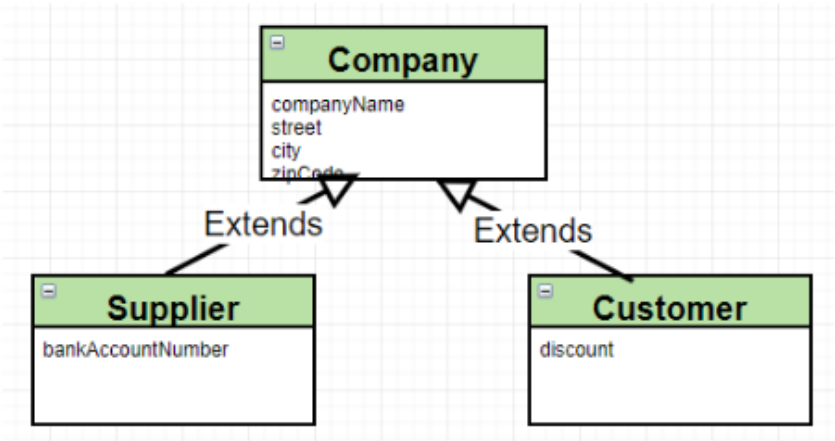
```
SUPPLIERID |CITY                                                    |STREET
                                                                    |COMPANYNAME
-------------------------------------------------------------------------------
1          |Krakow                                                  |dluga
                                                                    |Zabka
2          |Zielona Gora                                            |krotka
                                                                    |Biedronka
```

Logi z aplikacji wraz z wartością tabeli po zakończeniu pracy metody main

3.4 Dziedziczenie

XII. Dziedziczenie

a. Wprowadź do modelu następującą hierarchie:



b. Dodaj i pobierz z bazy kilka firm obu rodzajów stosując po kolei trzy różne strategie mapowania dziedziczenia.

c. **Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/diagram z datagrip, select * from….)**

```java
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Company {

    @Id
    @GeneratedValue
    private int companyId;

    @Column(name = "CompanyName")
    private String companyName;

    @Column(name = "Street")
    private String street;

    @Column(name = "City")
    private String city;

    public Company() {
    }

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

    public String getCompanyName() {
        return companyName;
    }

    public String getStreet() {
        return street;
    }

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

```java
import java.util.Set;

@Table(name = "Suppliers")
@Entity
public class Supplier extends Company {

    @Column(name = "BankAccountNumber")
    public String bankAccountNumber;

    // zadanie XI
```

```java
@Entity
public class Customer extends Company {

    @Column(name = "Discount")
    private double discount;

    public Customer() { super(); }

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

Modyfikacje w modelu

```java
    public static void main(String... args) {
        EntityManager manager = getEntityManager();

        EntityTransaction transaction = manager.getTransaction();
        transaction.begin();

...

        Customer cus1 = new Customer( companyName: "Zaczek",  street: "Budryka",  city: "Krakow",  discount: 12);
        Customer cus2 = new Customer( companyName: "Krakus",  street: "Rondo",  city: "Krakow",  discount: 4);
        Customer cus3 = new Customer( companyName: "Olimp",  street: "Aleje",  city: "Krakow",  discount: 6);
        manager.persist(cus1);
        manager.persist(cus2);
        manager.persist(cus3);

...
        transaction.commit();

//...

        transaction = manager.getTransaction();
        transaction.begin();

        List<Customer> customers =
                (List<Customer>) manager.createQuery( s: "SELECT s FROM Customer s")
                        .getResultList();
        System.out.println("Customers: ");
        for (Customer customer : customers) {
            System.out.println(customer.getCompanyName());
        }
        transaction.commit();
```

Kod testujący → dodaje trzech różnych klientów, którzy dziedziczą po klasie Company

Kod testowałem dla trzech różnych strategi mapowania dziedziczenia:

```java
@Entity
@Inheritance(strategy = InheritanceType.)
public abstract class Company {

    @Id
    @GeneratedValue
```

```
TABLE_PER_CLASS   InheritanceType
JOINED            InheritanceType
SINGLE_TABLE      InheritanceType
class
```

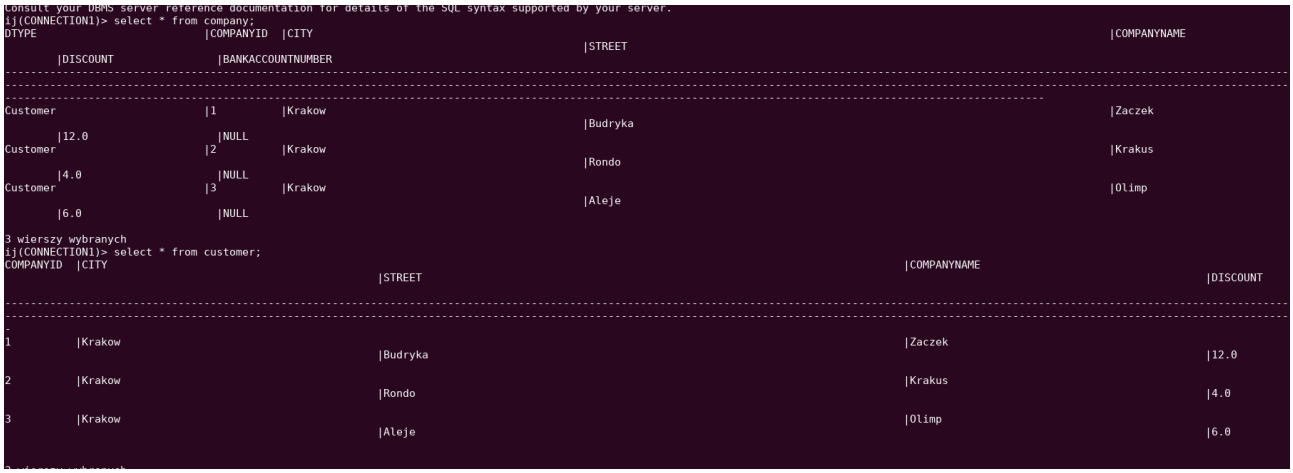
SINGLE_TABLE → jednostki z różnych klas o wspólnym przodku są umieszczane w jednej tabeli

JOINED → każda klasa ma swoją tabelę, a zapytanie o encję podklasy wymaga dołączenia do tabel

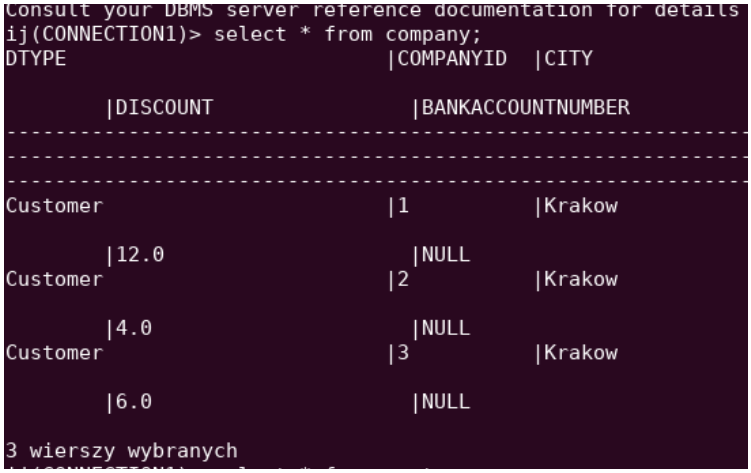
TABLE_PER_CLASS → wszystkie właściwości klasy znajdują się w jej tabeli, więc nie jest wymagane łączenie

Każda strategia skutkuje inną strukturą bazy danych

3.4.1 SINGLE_TABLE



*Baza danych po użyciu typu **single_table** → utworzyła się tabela **COMPANY***



3.4.2 TABLE_PER_CLASS

W tym przypadku nie tworzy się tabela **COMPANY**, jedynie tabela **CUSTOMER**

3.4.3 JOINED

Tutaj znowu pojawiają się tabele **CUSTOMER** i **COMPANY**, ale jedynie tabela COMPANY posiada pełne wpisy. Tabela CUSTOMER służy jako tabela łącznikowa:

```
BŁĄD 42X05: Tabela/widok 'CUSTOMERS' nie istnieje.
ij(CONNECTION1)> select * from customer;
DISCOUNT                |COMPANYID
------------------------------------
12.0                    |1
4.0                     |2
6.0                     |3

3 wierszy wybranych
ij(CONNECTION1)>
```

```
ij(CONNECTION1)> select * from company;
COMPANYID  |CITY                                                                                    |COMPANYNAME
                                                      |STREET
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
1          |Krakow                                                                                  |Zaczek
                                                      |Budryka
2          |Krakow                                                                                  |Krakus
                                                      |Rondo
3          |Krakow                                                                                  |Olimp
                                                      |Aleje
```

Zrzut z bazy pokazujący działanie dziedziczenia typu JOINED

4. Aplikacja webowa

XIII. Aplikacja do zamawiania produktów

a. Rozszerz projekt/zaimplementuj aplikacje wprowadzajac możliwosc składania zamowien na produkty (ew inne funkcjonalności wg własnego pomysłu stosownie do umiejętności/czasu/preferencji/chęci poznania).

b. Zwróć szczególną uwagę na wykorzystanie mechanizmów Hibernate/JPA

c. Zadanie (stosownie do umiejętności/czasu/preferencji/chęci poznania) może zostać zrealizowane jako:

   i. Rozszerzenie projektu rozpoczętego na zajęciach

   ii. Aplikacja „okienkowa”

   iii. Aplikacja webowa

d. Przygotuj sprawozdanie omawiające przygotowaną aplikację, wykorzystane mechanizmy I sposób ich dzialania (wraz ze screenami).

Aplikację przygotowałem przy użyciu Spring Boot 2 (back-end), Bootstrapa i jQuery (front-end).

Przygotowany projekt przepisałem na aplikację webową przy użyciu Spring Boota oraz Spring Boot JPA – biblioteki wspierającej Java Persistant API wykorzystującej Hibernate.

4.1 Konfiguracja kontrolerów

Przygotowałem kontrolery RESTowe do **tworzenia**, **edytowania** oraz **usuwania** rekordów dla tabeli *Products* oraz *Category*. Poza tą funkcjonalnością jest również end-point do pobierania wszystkich rekordów z danej tabeli i **listowania** ich w formie HTMLowej listy.

```
    @RequestMapping({"/product/list", "/product"})
    public String listProducts(Model model) {
        model.addAttribute( s: "products", productService.listAll());
        return "product/list";
    }

    @RequestMapping("/product/show/{id}")
    public String getProduct(@PathVariable String id, Model model) {
        model.addAttribute( s: "product", productService.getById(Long.valueOf(id)));
        return "product/show";
    }

    @RequestMapping("product/edit/{id}")
    public String edit(@PathVariable String id, Model model) {
        Product product = productService.getById(Long.valueOf(id));
        ProductForm productForm = productToProductForm.convert(product);

        model.addAttribute( s: "productForm", productForm);
        return "product/productform";
    }

    @RequestMapping("/product/new")
    public String newProduct(Model model) {
        model.addAttribute( s: "productForm", new ProductForm());
        return "product/productform";
    }

    @RequestMapping(value = "/product", method = RequestMethod.POST)
    public String saveOrUpdateProduct(@Valid ProductForm productForm,
                                      BindingResult bindingResult) {

        Product savedProduct =
                productService.saveOrUpdateProductForm(productForm);

        return "redirect:/product/show/" + savedProduct.getId();
    }

    @RequestMapping("/product/delete/{id}")
    public String delete(@PathVariable String id) {
        productService.delete(Long.valueOf(id));
        return "redirect:/product/list";
    }
}
```

Przygotowane kontrolery do obsługi interakcja z tabelą produktów

```
@Component
public class ProductFormToProduct implements Converter<ProductForm, Product> {

    @Override
    public Product convert(ProductForm productForm) {
        Product product = new Product();
        if (productForm.getId() != null &&
                !StringUtils.isEmpty(productForm.getId())) {
            product.setId(new Long(productForm.getId()));
        }
        product.setProductName(productForm.getProductName());
        product.setUnitsOnStock(productForm.getUnitsOnStock());
        product.setCategory(productForm.getCategory());
        return product;
    }
}
```

Mapowanie wartości z formularza na obiekt modelu

4.2 Front-end

Front end został zaimplentowany w bootstrapie 4 oraz jQuery. Przygotowałem 3 strony działające na moich end-pointach

- **list**.html
- **productform**.html
- **categoryform**.html

```
</head>
<body>
<div class="container">

    <h2>Products </h2>
            <div class="col-md-2">
        </div>      <div class="col-md-8">

        <form class="form-horizontal" th:object="${productForm}"
              th:action="@{/product}" method="post">

            <input type="hidden" th:field="*{id}"/>
            <div class="form-group"
                 th:class="${#fields.hasErrors('productName')} ? 'form-group has-error' : 'form-group'">
                <label class="col-sm-2 control-label">Product Name:</label>
                <div class="col-sm-10">
                    <input type="text" class="form-control"
                           th:field="*{productName}" th:errorclass="has-error"/>
                </div>
            </div>

            <div class="form-group"
                 th:class="${#fields.hasErrors('unitsOnStock')} ? 'form-group has-error' : 'form-group'">
                <label class="col-sm-2 control-label">Units on stock:</label>
                <div class="col-sm-10">
                    <input type="number" min="0" max="5000" step="0.01"
                           class="form-control" th:field="*{unitsOnStock}"
                           th:errorclass="has-error"/>
                </div>
            </div>

            <div class="form-group"
                 th:class="${#fields.hasErrors('category')} ? 'form-group has-error' : 'form-group'">
                <label class="col-sm-2 control-label">Category:</label>
                <div class="col-sm-10">
                    <input type="text" class="form-control"
                           th:field="*{category}" th:errorclass="has-error"/>
                </div>
            </div>
            <div class="row">
                <button type="submit" class="btn btn-default">Submit</button>
            </div>
        </form>
    </div>
            <div class="col-md-2">
        </div>
</div>
```

Kod formularza dla dodawania produktów, productform.html

Kod dp formularza dla kategorii jest zrobiony analogicznie (zawiera tylko jedno pole z nazwą). Poza formularzami znajduje się też strona do listowania zawartości bazy danych:

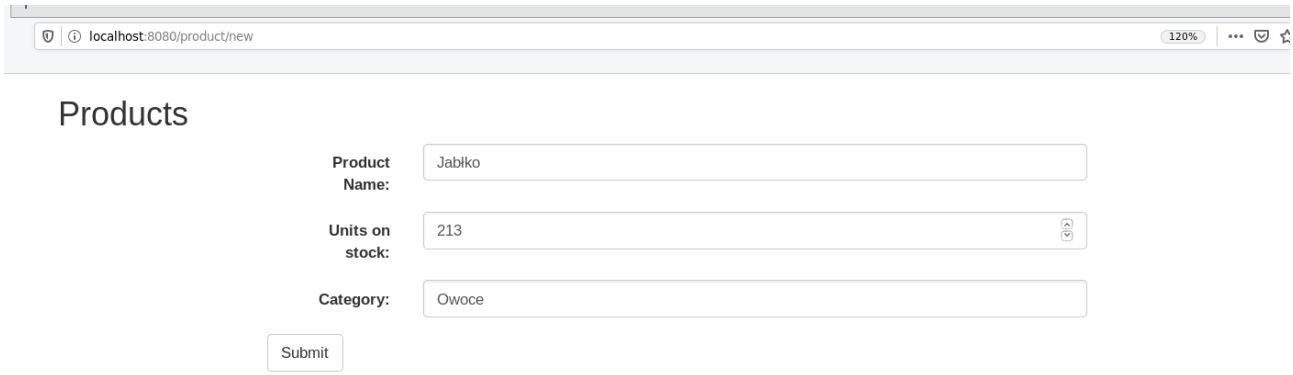
```
            <th>Delete</th>
        </tr>
        <tr th:each="product : ${products}">
            <td th:text="${product.id}"></td>
            <td th:text="${product.productName}"></td>
            <td th:text="${product.unitsOnStock}"></td>
            <td th:text="${product.category}"></td>
            <td><a th:href="${'/product/show/' + product.id}">Vie
            <td><a th:href="${'/product/edit/' + product.id}">Edi
```

Listowanie na stronę wartości z modelu, list.html

4.3 Prezentacja

```
2019-11-21 18:21:06.749 ERROR 24209 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport  : HHH000389: Unsuccessful: drop table product
2019-11-21 18:21:06.749 ERROR 24209 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport  : Schema 'SA' does not exist
Hibernate: create table product (id bigint generated by default as identity, category varchar(255), product_name varchar(255), units_on_stock varchar(255), primary key (id))
2019-11-21 18:21:06.782  WARN 24209 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : SQL Warning Code: 10000, SQLState: 01J01
2019-11-21 18:21:06.782  WARN 24209 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : Database 'memory:testdb' not created, connection made to existing database
 instead.
2019-11-21 18:21:06.782  INFO 24209 --- [           main] org.hibernate.tool.hbm2ddl.SchemaExport  : HHH000230: Schema export complete
2019-11-21 18:21:06.798  INFO 24209 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2019-11-21 18:21:07.187  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.context.embedded
 .AnnotationConfigEmbeddedWebApplicationContext@82de64a: startup date [Thu Nov 21 18:21:04 CET 2019]; root of context hierarchy
2019-11-21 18:21:07.224  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/product/delete/{id}]}" onto public java.lang.String guru
 .springframework.controllers.ProductController.delete(java.lang.String)
2019-11-21 18:21:07.225  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/product/edit/{id}]}" onto public java.lang.String guru
 .springframework.controllers.ProductController.edit(java.lang.String,org.springframework.ui.Model)
2019-11-21 18:21:07.225  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/product/new]}" onto public java.lang.String guru.springframework
 .controllers.ProductController.newProduct(org.springframework.ui.Model)
2019-11-21 18:21:07.226  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/product],methods=[POST]}" onto public java.lang.String guru
 .springframework.controllers.ProductController.saveOrUpdateProduct(guru.springframework.commands.ProductForm,org.springframework.validation.BindingResult)
2019-11-21 18:21:07.226  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/]}" onto public java.lang.String guru.springframework.controllers
 .ProductController.redirToList()
2019-11-21 18:21:07.226  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/product/list || /product]}" onto public java.lang.String
 guru.springframework.controllers.ProductController.listProducts(org.springframework.ui.Model)
2019-11-21 18:21:07.226  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/product/show/{id}]}" onto public java.lang.String guru
 .springframework.controllers.ProductController.getProduct(java.lang.String,org.springframework.ui.Model)
2019-11-21 18:21:07.229  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java
 .util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
2019-11-21 18:21:07.230  INFO 24209 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web
 .servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2019-11-21 18:21:07.252  INFO 24209 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework
 .web.servlet.resource.ResourceHttpRequestHandler]
2019-11-21 18:21:07.252  INFO 24209 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web
 .servlet.resource.ResourceHttpRequestHandler]
2019-11-21 18:21:07.280  INFO 24209 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org
 .springframework.web.servlet.resource.ResourceHttpRequestHandler]
2019-11-21 18:21:07.612  INFO 24209 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2019-11-21 18:21:07.645  INFO 24209 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2019-11-21 18:21:07.648  INFO 24209 --- [           main] g.s.SpringBootDerbyApplication           : Started SpringBootDerbyApplication in 4.393 seconds (JVM running for 5.453)
```

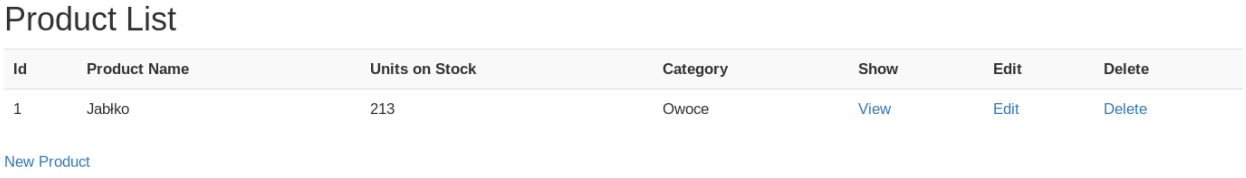
Logi po uruchomieniu aplikacji → poprawnie utworzone end-pointy („Mapped …)



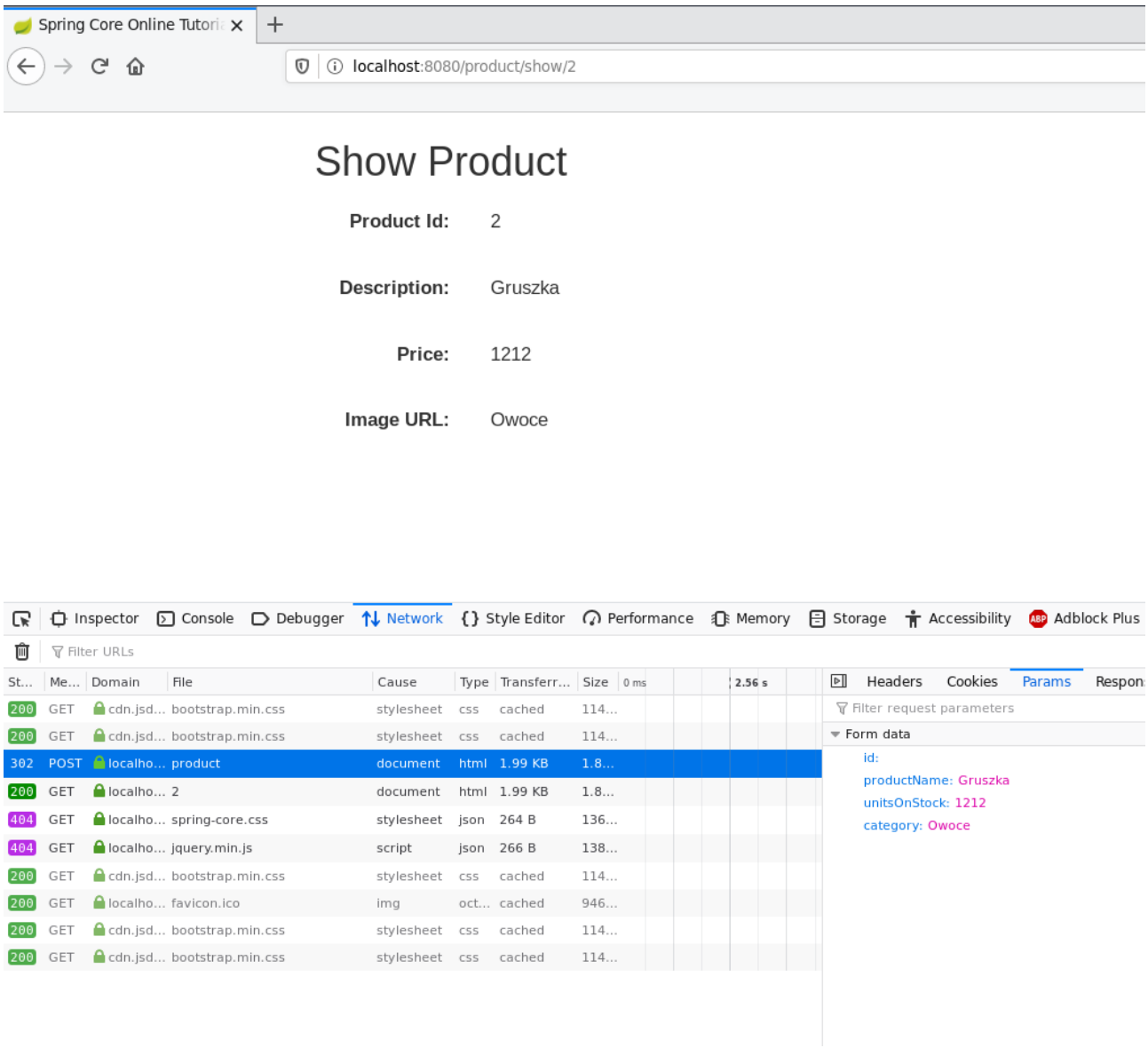
Wygląd formularza dla produktów

```
2019-11-21 18:26:12.009  INFO 24209 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization started
2019-11-21 18:26:12.017  INFO 24209 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization completed in 8 ms
2019-11-21 18:26:12.071  INFO 24209 --- [nio-8080-exec-2] o.h.h.i.QueryTranslatorFactoryInitiator  : HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select product0_.id as id1_0_, product0_.category as category2_0_, product0_.product_name as product_3_0_, product0_.units_on_stock as units_on4_0_ from product
 product0_
Hibernate: insert into product (id, category, product_name, units_on_stock) values (default, ?, ?, ?)
Hibernate: values identity_val_local()
Saved Product Id: 1
Hibernate: select product0_.id as id1_0_0_, product0_.category as category2_0_0_, product0_.product_name as product_3_0_0_, product0_.units_on_stock as units_on4_0_0_ from
 product product0_ where product0_.id=?
```

Reakcja aplikacji na wysłanie zapytania POST z danymi produktu → dodanie obiektu do bazy



*Rezultat strony **list**.html po wysłaniu POSTa*



Pokazanie zapytania w narzędziach programistycznych → strona działa poprawnie