

Sprawozdanie z Hibernate/JPA

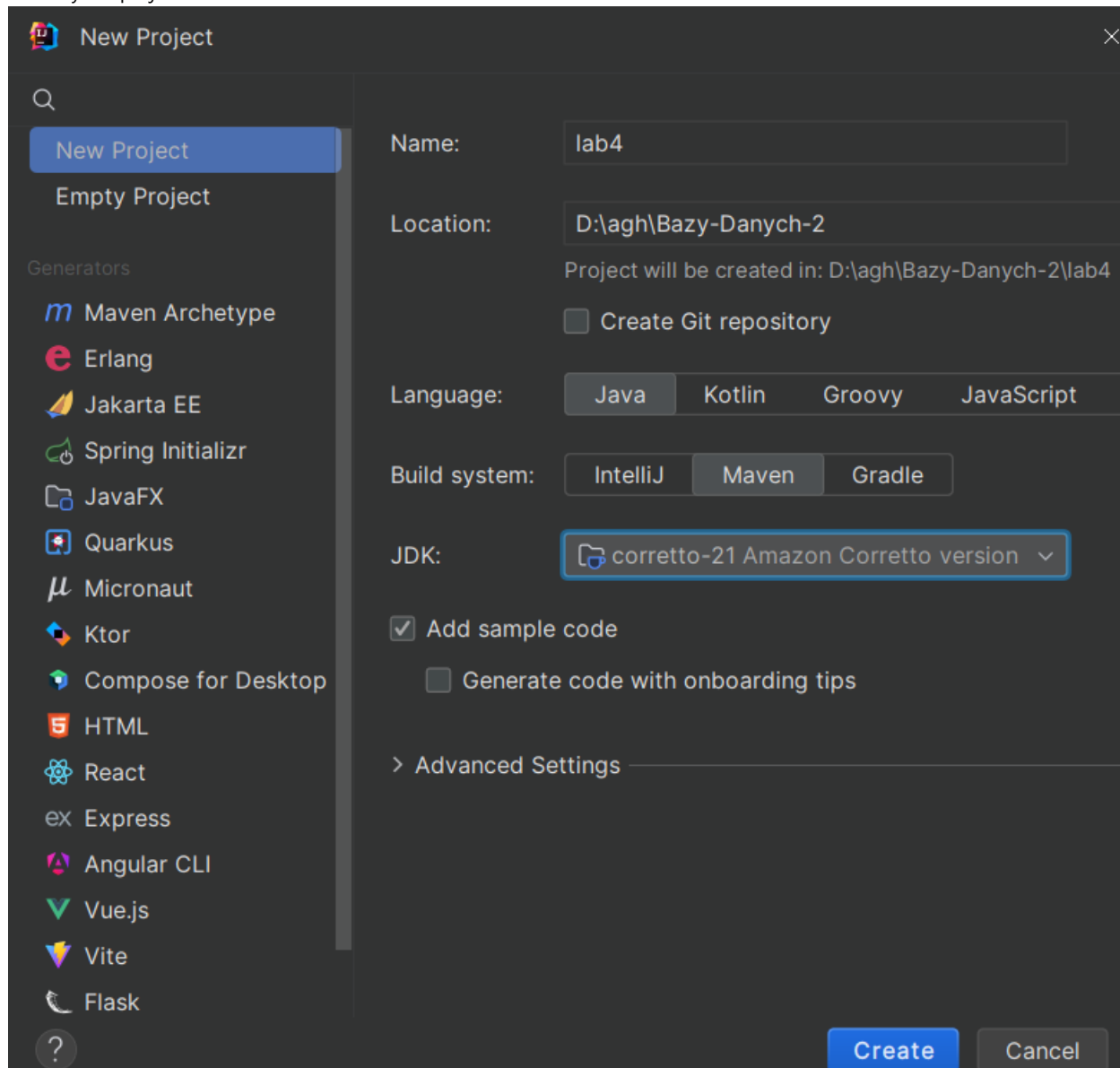
Szymon Żuk

Część 1

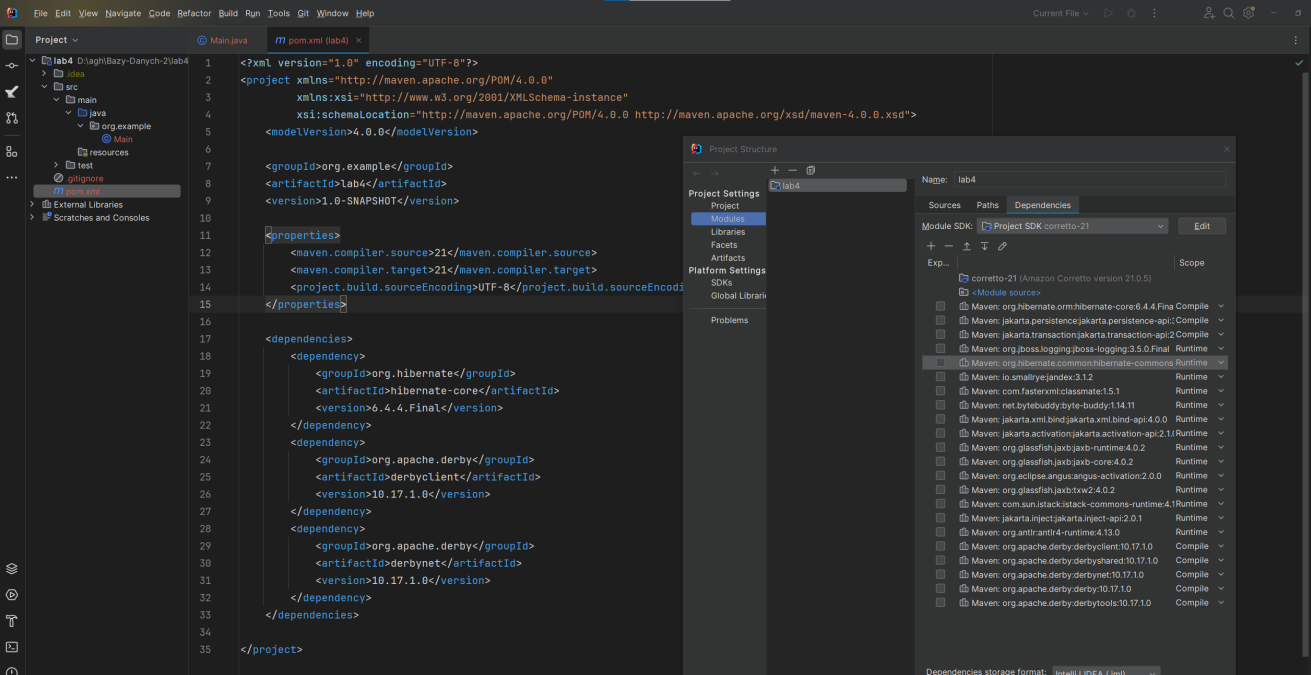
1. Pobrałem i uruchomiłem serwer Derby

```
PS D:\Program Files\db-derby-10.17.1.0-bin\bin> .\startNetworkServer
Sun Jun 01 13:20:32 CEST 2025 : Serwer sieciowy Apache Derby - 10.17.1.0 - (19132
17) uruchomiony i gotowy do zaakceptowania połączeń na porcie 1527 w {3}
```

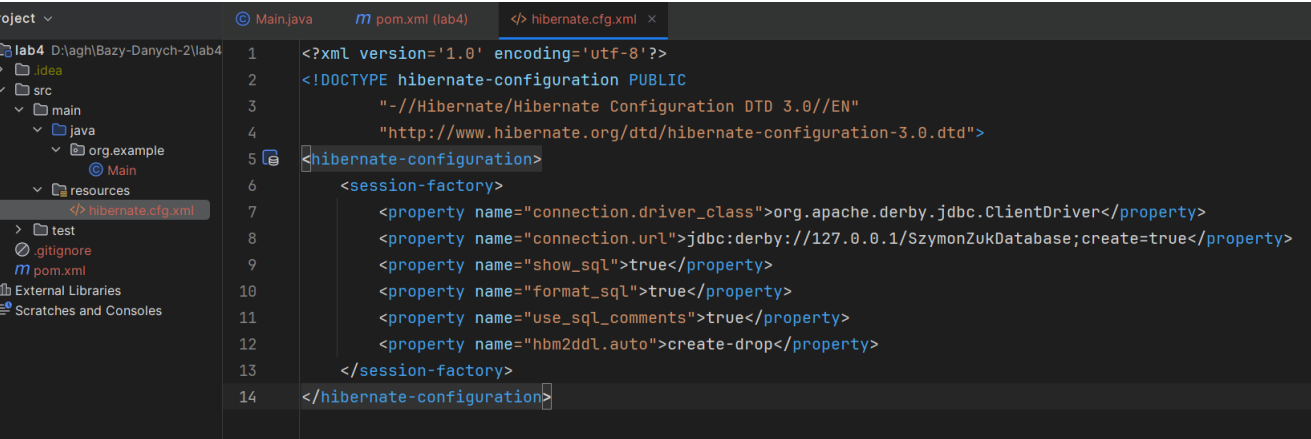
2. Utworzyłem projekt



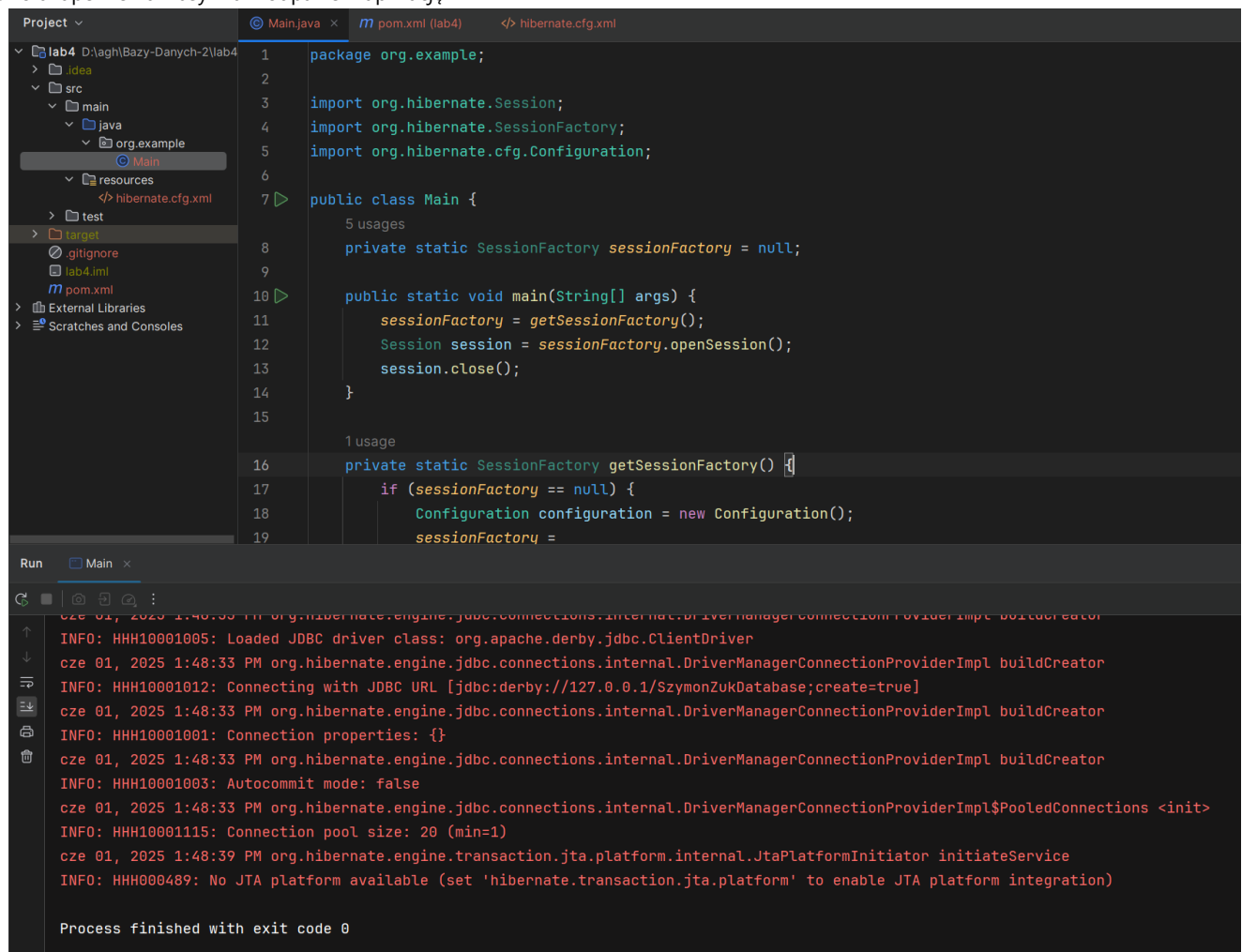
3. Dodałem zależności Hibernate i Apache-Derby



4. Dodałem plik konfiguracyjny Hibernate



5. Po uzupełnieniu klasy Main odpaliłem aplikację



The screenshot shows an IDE with a project named 'lab4'. The 'Main.java' file is open, showing the following code:

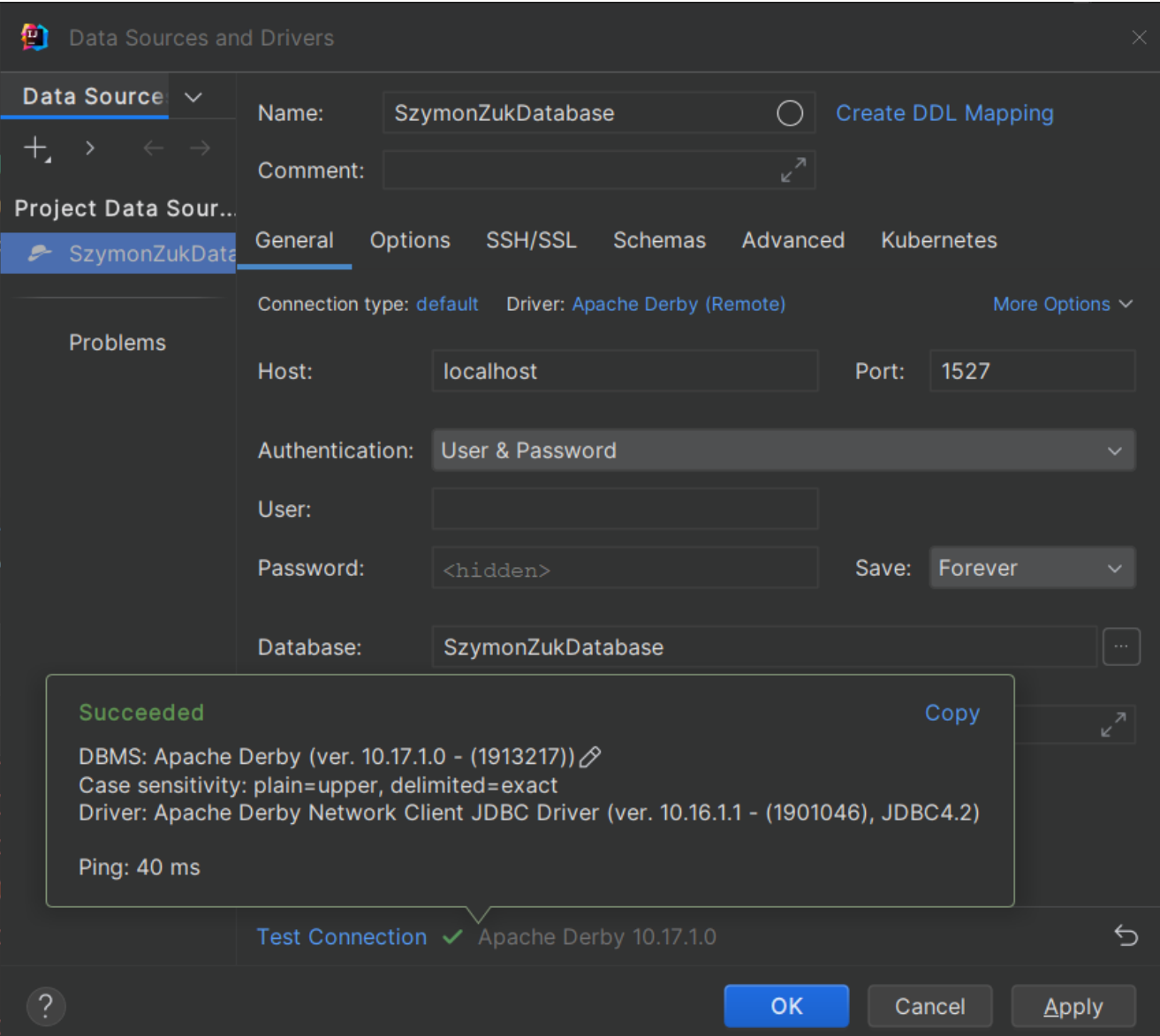
```
1 package org.example;
2
3 import org.hibernate.Session;
4 import org.hibernate.SessionFactory;
5 import org.hibernate.cfg.Configuration;
6
7 public class Main {
8     private static SessionFactory sessionFactory = null;
9
10    public static void main(String[] args) {
11        sessionFactory = getSessionFactory();
12        Session session = sessionFactory.openSession();
13        session.close();
14    }
15
16    private static SessionFactory getSessionFactory() {
17        if (sessionFactory == null) {
18            Configuration configuration = new Configuration();
19            sessionFactory =
```

The 'Run' button is highlighted, and the 'Run' window shows the following output:

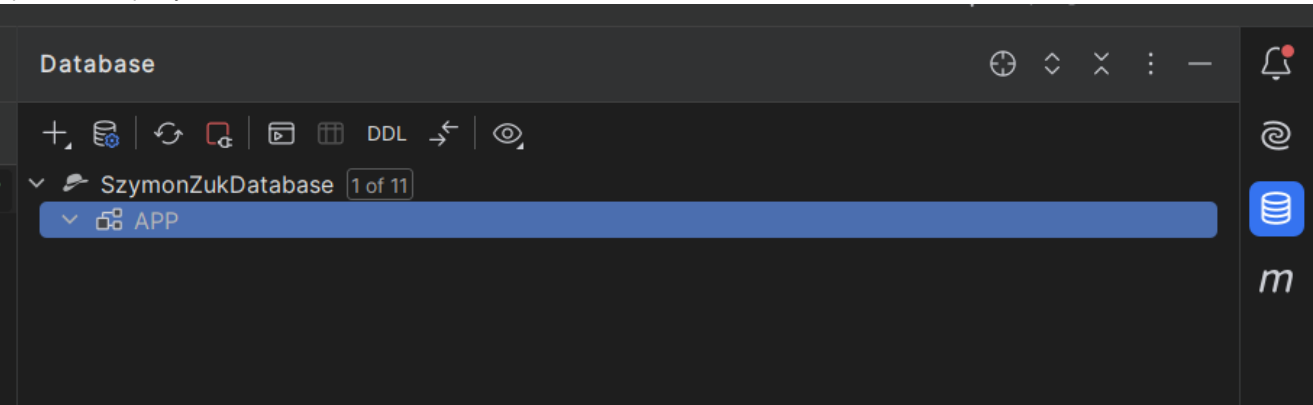
```
cze 01, 2025 1:48:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: Loaded JDBC driver class: org.apache.derby.jdbc.ClientDriver
cze 01, 2025 1:48:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001012: Connecting with JDBC URL [jdbc:derby://127.0.0.1/SzymonZukDatabase;create=true]
cze 01, 2025 1:48:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {}
cze 01, 2025 1:48:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
cze 01, 2025 1:48:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH10001115: Connection pool size: 20 (min=1)
cze 01, 2025 1:48:39 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)

Process finished with exit code 0
```

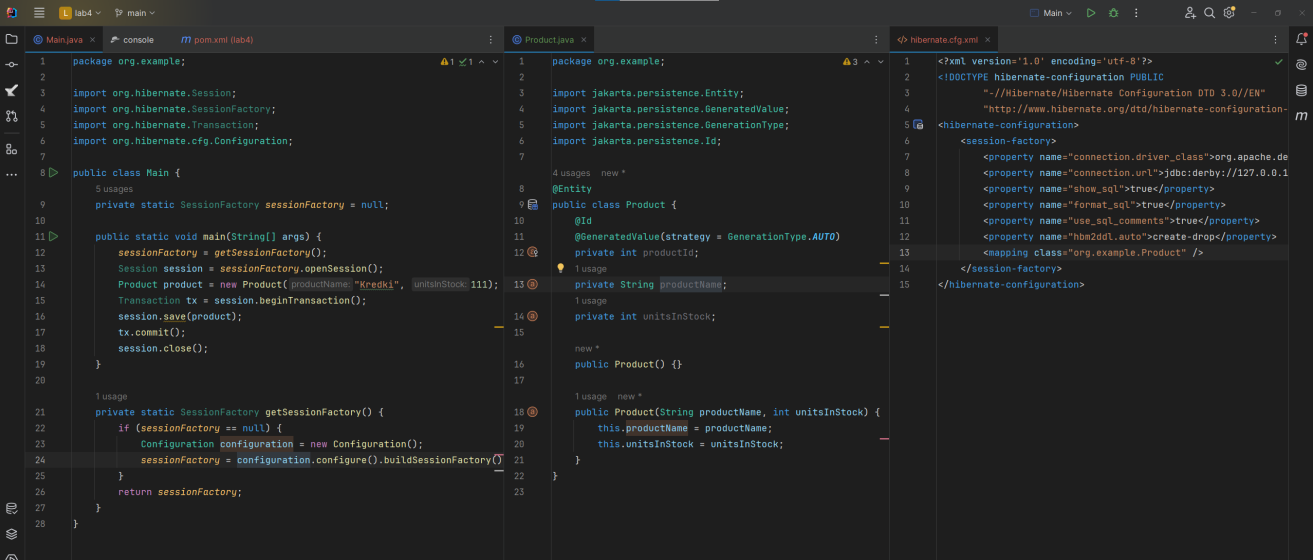
6. Zweryfikowałem bazę danych z poziomu IntelliJ



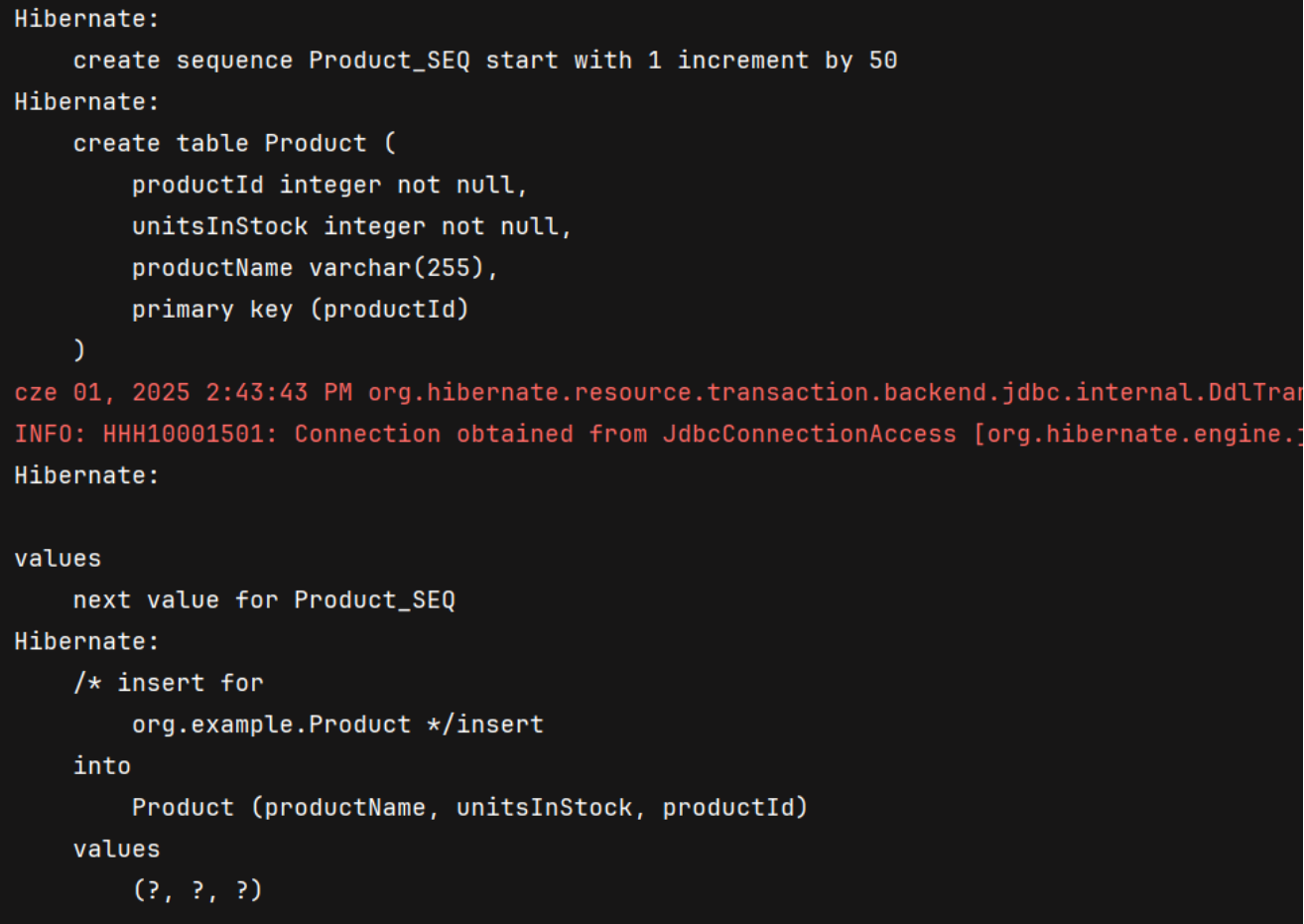
7. Sprawdziłem pusty schemat APP



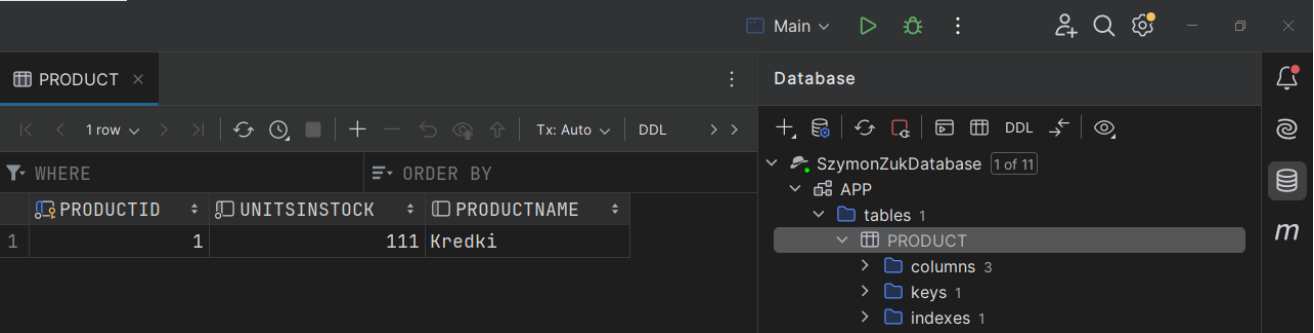
8. Dodałem klasę produktu, mapping i kod w mainie dodający produkt



9. Odpałem aplikację



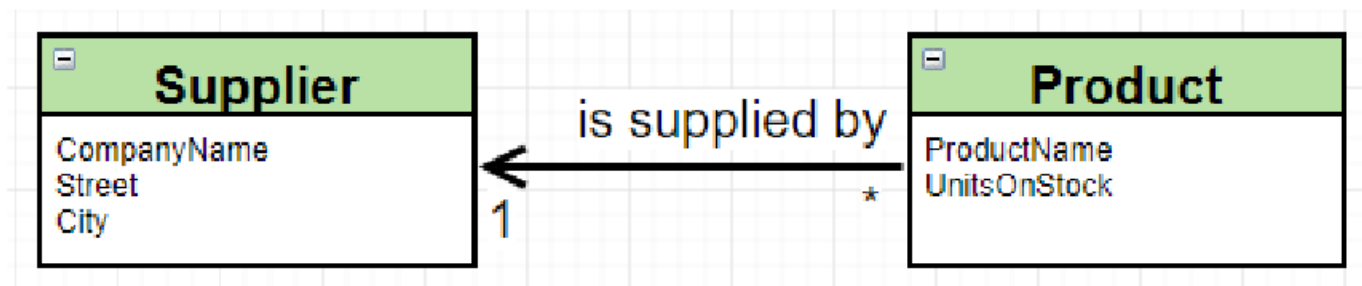
10. Sprawdziłem, czy produkt został dodany do bazy danych



Część 2

Podpunkt 1

Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej



- Stworz nowego dostawce.
- Znajdz poprzednio wprowadzony produkt i ustaw jego dostawce na właśnie dodanego.
- Udokumentuj wykonane kroki oraz uzyskany rezultat (ogi wywołań sqlowych, describe table/schemat bazy danych, select * from....)**

Modyfikacje modelu danych

```
// Dodałem klasę dostawcy (i mapping do niej)
@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;
    }
}
```

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsInStock;
    // Dodałem pole tworzące relację
    @ManyToOne
    private Supplier isSuppliedBy;

    public Product() {}

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

    // Dodałem setter do pola isSuppliedBy
    public void setIsSuppliedBy(Supplier isSuppliedBy) {
        this.isSuppliedBy = isSuppliedBy;
    }
}
```

Dodałem do maina kod dodający dostawcę, znajdujący wcześniej dodany produkt i ustawiający jego dostawcę na nowo dodanego

```
tx = session.beginTransaction();
var supplier = new Supplier("Inpost", "Kawiory", "Kraków");
session.save(supplier);
product = session.get(Product.class, 1);
product.setIsSuppliedBy(supplier);
session.save(product);
tx.commit();
```

Wynik działania kodu

```
Hibernate:
    create table Supplier (
        supplierId integer not null,
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        primary key (supplierId)
    )
Hibernate:
    alter table Product
        add constraint FK1etx50i6xp1rj7vm6ggjq5mu1s
        foreign key (isSuppliedBy_supplierId)
        references Supplier
Hibernate:

values
    next value for Product_SEQ
Hibernate:
/* insert for
    org.example.Product */insert
into
    Product (isSuppliedBy_supplierId, productName, unitsInStock, productId)
values
    (?, ?, ?, ?)
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:
/* insert for
    org.example.Supplier */insert
into
    Supplier (city, companyName, street, supplierId)
values
    (?, ?, ?, ?)
Hibernate:
/* update
    for org.example.Product */update Product
set
    isSuppliedBy_supplierId=?,
    productName=?,
    unitsInStock=?
where
    productId=?
```


Schemat i zawartość bazy danych

The screenshot displays a database management interface with two tables: **PRODUCT** and **SUPPLIER**.

PRODUCT Table:

ISSUPPLIEDBY_SUPPLIERID	PRODUCTID	UNITSINSTOCK	PRODUCTNAME
1	1	1	111 Kredki

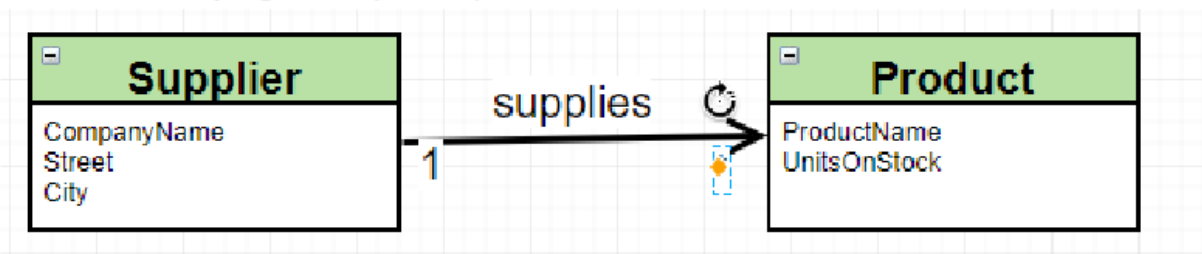
SUPPLIER Table:

SUPPLIERID	CITY	COMPANYNAME	STREET
1	Kraków	Inpost	Kawłory

The bottom part of the screenshot shows the ER diagram for these tables. The **SUPPLIER** table has columns: CITY (varchar(255)), COMPANYNAME (varchar(255)), STREET (varchar(255)), and SUPPLIERID (integer, primary key). The **PRODUCT** table has columns: ISSUPPLIEDBY_SUPPLIERID (integer, foreign key to SUPPLIERID), UNITSINSTOCK (integer), PRODUCTNAME (varchar(255)), and PRODUCTID (integer, primary key). A relationship line connects SUPPLIERID in SUPPLIER to ISSUPPLIEDBY_SUPPLIERID in PRODUCT.

Podpunkt 2

II. Odwróć relacje zgodnie z poniższym schematem



- Zamodeluj powyższe w dwóch wariantach „z” i „bez” tabeli łącznikowej
- Stwórz kilka produktów
- Dodaj je do produktów dostarczanych przez nowo stworzonego dostawcę
- Udokumentuj wykonane kroki oraz uzyskane rezultaty w obu wariantach (logi wywołań słowych, describe table/schemat bazy danych, select * from....)

Wariant z tabelą łącznikową

Modyfikacje modelu danych

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsInStock;
    // Usunąłem pole tworzące relację
    // @ManyToOne
```

```
// private Supplier isSuppliedBy;

public Product() {}

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

// Usunąłem setter isSuppliedBy
// public void setIsSuppliedBy(Supplier isSuppliedBy) {
//     this.isSuppliedBy = isSuppliedBy;
// }
}
```

```
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierId;
    private String companyName;
    private String street;
    private String city;
    // Dodałem pole tworzące relację
    @OneToMany
    private Set<Product> supplies = new HashSet<>();

    public Supplier() {}

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

    // Dodałem metodę do dodawania produktu
    public void addSuppliedProduct(Product product) {
        supplies.add(product);
    }
}
```

Usunąłem kod w mainie z poprzedniego podpunktu i dodałem nowy

```
tx = session.beginTransaction();
var supplier = new Supplier("Inpost", "Kawitory", "Kraków");
product = session.get(Product.class, 1);
supplier.addSuppliedProduct(product);
session.save(supplier);
tx.commit();
```

Wynik działania kodu

```
Hibernate:
    create table Supplier (
        supplierId integer not null,
        city varchar(255),
        companyName varchar(255),
        street varchar(255),
        primary key (supplierId)
    )
Hibernate:
    create table Supplier_Product (
        Supplier_supplierId integer not null,
        supplies_productId integer not null unique,
        primary key (Supplier_supplierId, supplies_productId)
    )
Hibernate:
    alter table Supplier_Product
    add constraint FK4k7vne8ha6r6m4h2rmtxfu75k
    foreign key (supplies_productId)
    references Product
Hibernate:
    alter table Supplier_Product
    add constraint FK24j3kwmoysj1j4x3tdhpane6g
    foreign key (Supplier_supplierId)
    references Supplier
Hibernate:

values
    next value for Product_SEQ
Hibernate:
    /* insert for
        org.example.Product */insert
    into
        Product (productName, unitsInStock, productId)
    values
        (?, ?, ?)
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:
    /* insert for
        org.example.Supplier */insert
    into
        Supplier (city, companyName, street, supplierId)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        org.example.Supplier.supplies */insert
    into
        Supplier_Product (Supplier_supplierId, supplies_productId)
    values
        (?, ?)
```

Schemat i zawartość bazy danych

The screenshot shows a database management tool interface. The top panel displays the database structure, including the 'SzymonZukDatabase' and its 'APP' schema. The 'SUPPLIER_PRODUCT' table is highlighted. The middle panel shows the 'Visualization for SUPPLIER_PRODUCT' table, which is a junction table between 'PRODUCT' and 'SUPPLIER'. The bottom panel shows the data for each table:

PRODUCTID	UNITSINSTOCK	PRODUCTNAME
1	1	Kredki

SUPPLIERID	CITY	COMPANYNAME	STREET
1	Kraków	Inpost	Kawłory

SUPPLIER_SUPPLIERID	SUPPLIES_PRODUCTID	SUPPLIER_PRODUCTID
1	1	1

Wariant bez tabeli łącznikowej

Modyfikacje modelu danych

```

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

    @OneToMany
    // Dodałem JoinColumn
    @JoinColumn(name="SUPPLIER_FK")
    private Set<Product> supplies = new HashSet<>();

    public Supplier() {}

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

    // Dodałem metodę do dodawania produktu

```

```
public void addSuppliedProduct(Product product) {  
    supplies.add(product);  
}  
}
```

Kod w mainie zostawiłem taki sam jak w poprzednim wariancie. W wyniku działania kodu pojawił się klucz obcy

Hibernate:

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

Hibernate:

```
alter table Product  
    add constraint FKeury2hxl2j8urlkmw36585tkr  
    foreign key (SUPPLIER_FK)  
    references Supplier
```

Hibernate:

Schemat i zawartość bazy danych

The screenshot displays a database management interface with two tables: PRODUCT and SUPPLIER. The PRODUCT table has columns: SUPPLIER_FK, PRODUCTID, UNITSINSTOCK, and PRODUCTNAME. The SUPPLIER table has columns: SUPPLIERID, CITY, COMPANYNAME, and STREET. The data for the PRODUCT table is as follows:

SUPPLIER_FK	PRODUCTID	UNITSINSTOCK	PRODUCTNAME
1	1	1	111 Kredki

The data for the SUPPLIER table is as follows:

SUPPLIERID	CITY	COMPANYNAME	STREET
1	1 Kraków	Inpost	Kawiony

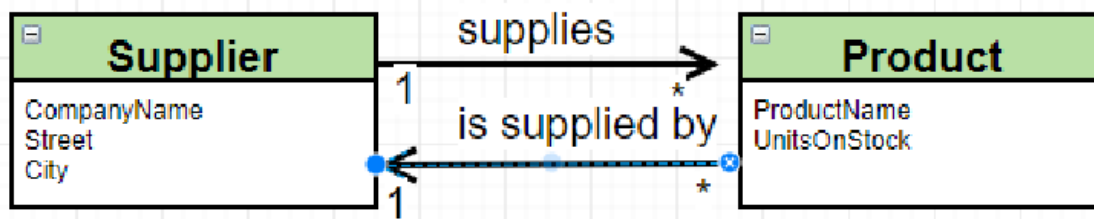
The interface also shows a schema diagram at the bottom with the following details:

- SUPPLIER**
 - CITY: varchar(255)
 - COMPANYNAME: varchar(255)
 - STREET: varchar(255)
 - SUPPLIERID: integer (primary key)
- PRODUCT**
 - UNITSINSTOCK: integer
 - PRODUCTNAME: varchar(255)
 - PRODUCTID: integer (primary key)

The right sidebar shows the database structure: SzymonZukDatabase (1 of 11) > APP > tables 2 > PRODUCT > columns 3, keys 1, foreign keys 1, indexes 2. The SUPPLIER table is also listed with columns 4, keys 1, and indexes 1.

Podpunkt 3

III. Zamodeluj relację dwustronną jak poniżej:



- Tradycyjnie: Stwórz kilka produktów
- Dodaj je do produktów dostarczanych przez nowo stworzonego dostawcę (pamiętaj o poprawnej obsłudze dwustronności relacji)
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select * from....)

Modyfikacje modelu danych

```

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsInStock;
    // Dodałem pole tworzące relację z JoinColumn
    @ManyToOne
    // Nazwa JoinColumn taka sama jak w Supplier
    @JoinColumn(name="SUPPLIER_FK")
    private Supplier isSuppliedBy;

    public Product() {}

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

    // Znów dodałem setter isSuppliedBy
    public void setIsSuppliedBy(Supplier isSuppliedBy) {
        this.isSuppliedBy = isSuppliedBy;
    }
}

```

Usunąłem kod w mainie z poprzedniego punktu i dodałem nowy

```

tx = session.beginTransaction();
var supplier = new Supplier("Inpost", "Kawior", "Kraków");
product = session.get(Product.class, 1);
product.setIsSuppliedBy(supplier);
supplier.addSuppliedProduct(product);
session.save(supplier);
tx.commit();

```

Wynik działania kodu

Hibernate:

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

Hibernate:

```
alter table Product  
    add constraint FKeury2hxl2j8urlkmw36585tkr  
    foreign key (SUPPLIER_FK)  
    references Supplier
```

Hibernate:

values

```
next value for Product_SEQ
```

Hibernate:

```
/* insert for  
    org.example.Product */insert  
into  
    Product (SUPPLIER_FK, productName, unitsInStock, productId)  
values  
    (?, ?, ?, ?)
```

Hibernate:

values

```
next value for Supplier_SEQ
```

Hibernate:

```
/* insert for  
    org.example.Supplier */insert  
into  
    Supplier (city, companyName, street, supplierId)  
values  
    (?, ?, ?, ?)
```

Hibernate:

```
/* update  
    for org.example.Product */update Product  
set  
    SUPPLIER_FK=?,  
    productName=?,  
    unitsInStock=?
```



```

where
    productId=?
Hibernate:
    update
        Product
    set
        SUPPLIER_FK=?
    where
        productId=?

```

Schemat i zawartość bazy danych są takie same jak w podpunkcie 2 z wariantem bez tabeli łącznikowej

The screenshot displays a database management interface for 'SzymonZukDatabase'. It shows two tables: 'PRODUCT' and 'SUPPLIER'. The 'PRODUCT' table has columns: SUPPLIER_FK (integer), PRODUCTID (integer), UNITSINSTOCK (integer), and PRODUCTNAME (varchar(255)). The 'SUPPLIER' table has columns: SUPPLIERID (integer), CITY (varchar(255)), COMPANYNAME (varchar(255)), and STREET (varchar(255)). A foreign key relationship is established between PRODUCT.SUPPLIER_FK and SUPPLIER.SUPPLIERID. The data view shows one row in each table: Product 111 'Kredki' and Supplier 1 'Kraków'.

Podpunkt 4

- IV. Dodaj klasę Category z property int CategoryID, String Name oraz listą produktów List<Product> Products
- Zmodyfikuj produkty dodając wskazanie na kategorii do której należy.
 - Stwórz kilka produktów i kilka kategorii
 - Dodaj kilka produktów do wybranej kategorii
 - Wydobądź produkty z wybranej kategorii oraz kategorię do której należy wybrany produkt
 - Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select * from....)

Modyfikacje modelu danych

```
// Dodałem klasę kategorii (i jej mapping)
@Entity
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int categoryId;
    private String categoryName;
    @OneToMany
    @JoinColumn(name="CATEGORY_FK")
    private List<Product> products = new ArrayList<>();

    public Category() {}

    public Category(String categoryName, List<Product> products) {
        this.categoryName = categoryName;
        this.products = products;
    }

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

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

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsInStock;
    @ManyToOne
    @JoinColumn(name="SUPPLIER_FK")
    private Supplier isSuppliedBy;
    // Dodałem pole tworzące relację
    @ManyToOne
    @JoinColumn(name="CATEGORY_FK")
    private Category category;

    /* ... */

    // Dodałem setter kategorii
    public void setCategory(Category category) {
        this.category = category;
    }

    // Dodałem getter do wypisania danych w konsoli
    public String getProductName() {
        return productName;
    }
}
```

Usunąłem wszystko co było wcześniej w mainie i dodałem nowy kod, który dodaje produkty i kategorie, a potem wydobywa produkty z wybranej kategorii

```
var sessionFactory = getSessionFactory();
var session = sessionFactory.openSession();
var tx = session.beginTransaction();

var klawiatura = new Product("Klawiatura", 25);
var myszka = new Product("Myszka", 22);
var zasilacz = new Product("Zasilacz", 0);

var tulipan = new Product("Tulipan", 20);
var storczyk = new Product("Storczyk", 3);

var buty = new Product("Buty", 2);
var spodnie = new Product("Spodnie", 15);

var peryferia = new Category(
    "Peryferia",
    List.of(klawiatura, myszka, zasilacz)
);
klawiatura.setCategory(peryferia);
myszka.setCategory(peryferia);
zasilacz.setCategory(peryferia);
session.save(klawiatura);
session.save(myszka);
session.save(zasilacz);
session.save(peryferia);

var kwiaty = new Category(
    "Kwiaty",
    List.of(tulipan, storczyk)
);
tulipan.setCategory(kwiaty);
storczyk.setCategory(kwiaty);
session.save(tulipan);
session.save(storczyk);
session.save(kwiaty);

var ubrania = new Category(
    "Ubrania",
    List.of(buty, spodnie)
);
buty.setCategory(ubrania);
spodnie.setCategory(ubrania);
session.save(buty);
session.save(spodnie);
session.save(ubrania);

tx.commit();

tx = session.beginTransaction();
var query = session.createQuery("from Category as cat where cat.categoryName='Ubrania'");
var kategoria = (Category) query.getSingleResult();
System.out.println("Produkty z kategorii Ubrania:");
for (var p : kategoria.getProducts()) {
    System.out.println(p.getProductName());
}
tx.commit();

session.close();
```

Wynik działania kodu

Hibernate:

```
create table Category (
    categoryId integer not null,
    categoryName varchar(255),
    primary key (categoryId)
)
```

cze 01, 2025 5:57:07 PM org.hibernate.resource

INFO: HHH10001501: Connection obtained from

Hibernate:

```
create table Product (
    CATEGORY_FK integer,
    SUPPLIER_FK integer,
    productId integer not null,
    unitsInStock integer not null,
    productName varchar(255),
    primary key (productId)
)
```

Produkty z kategorii Ubrania:

Buty

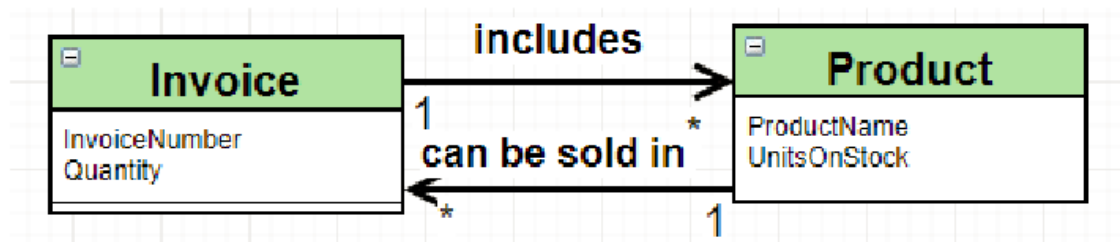
Spodnie

Schemat i zawartość bazy danych

The screenshot shows a database management tool interface. The top panel displays the schema for the 'SzymonZukDatabase'. The 'CATEGORY' table has columns 'categoryId' (primary key) and 'categoryName'. The 'PRODUCT' table has columns 'CATEGORY_FK', 'SUPPLIER_FK', 'productId' (primary key), 'unitsInStock', and 'productName'. The 'SUPPLIER' table has columns 'city', 'companyName', 'street', and 'supplierId' (primary key). The bottom panel shows the data for the 'CATEGORY' table, which contains 3 rows: '1 Peryferia', '2 Kwiaty', and '3 Ubrania'. The 'PRODUCT' table contains 7 rows of data, including products like 'Klawiatura', 'Myszka', 'Zasilacz', 'Tulipan', 'Storczyk', 'Buty', and 'Spodnie'. A diagram view for the 'PRODUCT' table illustrates the relationships between the tables, showing foreign key constraints from 'PRODUCT' to 'SUPPLIER' and 'CATEGORY'.

Podpunkt 5

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



- Stórz kilka produktów i “sprzedaj” je na kilku transakcjach.
- Pokaż produkty sprzedane w ramach wybranej faktury/transakcji
- Pokaż faktury w ramach których był sprzedany wybrany produkt
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select * from....)**

Modyfikacje modelu danych

```

// Dodałem klasę faktury
@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int invoiceId;
    private int invoiceNumber;
    private int quantity;
    @ManyToMany
    private Set<Product> includes = new HashSet<>();

    public Invoice() {}

    public Invoice(int invoiceNumber, int quantity, Set<Product> includes) {
        this.invoiceNumber = invoiceNumber;
        this.quantity = quantity;
        this.includes = includes;
    }

    public Set<Product> getIncludes() {
        return includes;
    }

    public int getInvoiceNumber() {
        return invoiceNumber;
    }
}
  
```

```

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
  
```

```

    private int productId;
    private String productName;
    private int unitsInStock;
    @ManyToOne
    @JoinColumn(name="SUPPLIER_FK")
    private Supplier isSuppliedBy;
    @ManyToOne
    @JoinColumn(name="CATEGORY_FK")
    private Category category;
    // Dodałem pole tworzące relację many-to-many
    @ManyToMany(mappedBy="includes")
    private Set<Invoice> canBeSoldIn = new HashSet<>();

    /* ... */

    // Dodałem metodę do dodawania faktury
    public void addCanBeSoldIn(Invoice invoice) {
        canBeSoldIn.add(invoice);
    }

    // Dodałem getter do wypisania faktur
    public Set<Invoice> getCanBeSoldIn() {
        return canBeSoldIn;
    }
}

```

Usunąłem wszystko co było wcześniej w mainie i dodałem nowy kod, który dodaje produkty i faktury, znajduje produkty sprzedane w ramach danej faktury, i faktury, w których sprzedano dany produkt

```

var sessionFactory = getSessionFactory();
var session = sessionFactory.openSession();
var tx = session.beginTransaction();

var tulipan = new Product("Tulipan", 20);
var fiolek = new Product("Fiołek", 5);
var storczyk = new Product("Storczyk", 3);
var roza = new Product("Róża", 3);

var invoice1 = new Invoice(1, 4, Set.of(tulipan, fiolek, storczyk, roza));
tulipan.addCanBeSoldIn(invoice1);
fiolek.addCanBeSoldIn(invoice1);
storczyk.addCanBeSoldIn(invoice1);
roza.addCanBeSoldIn(invoice1);

var invoice2 = new Invoice(2, 3, Set.of(tulipan, fiolek, storczyk));
tulipan.addCanBeSoldIn(invoice2);
fiolek.addCanBeSoldIn(invoice2);
storczyk.addCanBeSoldIn(invoice2);

var invoice3 = new Invoice(3, 1, Set.of(tulipan));
tulipan.addCanBeSoldIn(invoice3);

var invoice4 = new Invoice(4, 1, Set.of(fiolek, roza));
fiolek.addCanBeSoldIn(invoice4);
roza.addCanBeSoldIn(invoice4);

session.save(tulipan);
session.save(fiolek);
session.save(storczyk);
session.save(roza);

session.save(invoice1);
session.save(invoice2);
session.save(invoice3);
session.save(invoice4);

```

```
tx.commit();

// Pokazanie produktów sprzedanych w ramach faktury nr. 1
tx = session.beginTransaction();
var query1 = session.createQuery("from Invoice as inv where inv.invoiceNumber=1");
var faktura = (Invoice) query1.getSingleResult();

// Pokazanie faktur w których sprzedano tulipany
var query2 = session.createQuery("from Product as prod where prod.productName='Tulipan'");
var produkt = (Product) query2.getSingleResult();

System.out.print("Produkty sprzedane w ramach faktury nr 1: ");
for (var p : faktura.getIncludes()) {
    System.out.print(p.getProductName() + " ");
}
System.out.print("\n");

System.out.print("Numery faktur, w których sprzedano tulipany: ");
for (var i : produkt.getCanBeSoldIn()) {
    System.out.print(i.getInvoiceNumber() + " ");
}
tx.commit();

session.close();
```

Wynik działania kodu

Hibernate:

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

Hibernate:

```
create table Invoice_Product (  
    canBeSoldIn_invoiceId integer not null,  
    includes_productId integer not null,  
    primary key (canBeSoldIn_invoiceId, includes_productId)  
)
```

Hibernate:

```
create table Product (  
    CATEGORY_FK integer,  
    SUPPLIER_FK integer,  
    productId integer not null,  
    unitsInStock integer not null,  
    productName varchar(255),  
    primary key (productId)  
)
```

Produkty sprzedane w ramach faktury nr 1: Storczyk Tulipan Róża Fiołek
Numery faktur, w których sprzedano tulipany: 2 3 1

Schemat i zawartość bazy danych

The screenshot displays a database management interface with three tables and a schema diagram.

INVOICE Table:

INVOICEID	INVOICENUMBER	QUANTITY
1	1	4
2	2	3
3	3	1
4	4	1

INVOICE_PRODUCT Table:

CANBESOLDIN_INVOICEID	INCLUDES_PRODUCTID
1	1
2	1
3	1
4	1
5	2
6	2
7	2
8	3
9	2
10	4

PRODUCT Table:

CATEGORY_FK	SUPPLIER_FK	PRODUCTID	UNITSINSTOCK	PRODUCTNAME
<null>	<null>	1	20	Tulipan
<null>	<null>	2	5	Fiołek
<null>	<null>	3	3	Storczyk
<null>	<null>	4	3	Róża

Database Schema Diagram:

```

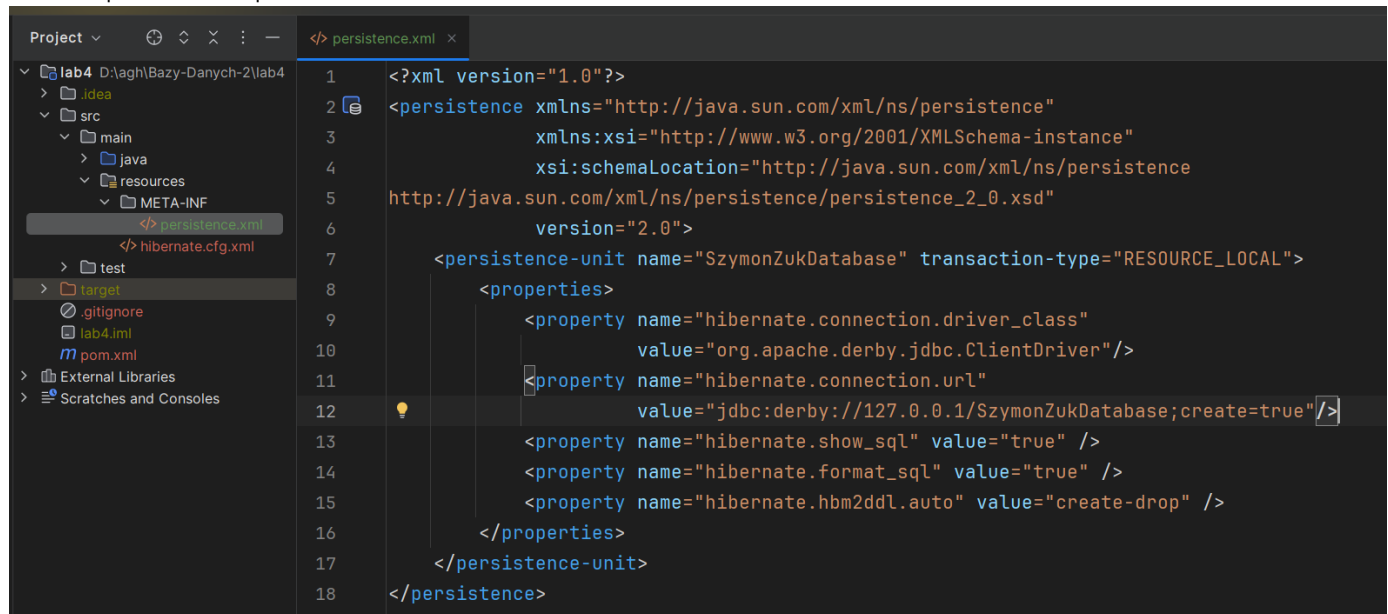
graph TD
    SUPPLIER -- "SUPPLIER_FK: SUPPLIERID" --> PRODUCT
    CATEGORY -- "CATEGORY_FK: CATEGORYID" --> PRODUCT
    INVOICE -- "INVOICEID: INVOICEID" --> INVOICE_PRODUCT
    INVOICE_PRODUCT -- "INCLUDES_PRODUCTID: PRODUCTID" --> PRODUCT
    INVOICE_PRODUCT -- "CANBESOLDIN_INVOICEID: INVOICEID" --> INVOICE
  
```

Podpunkt 6

VI. JPA

- Stwórz nowego maina w którym zrobisz to samo co w poprzednim ale z wykorzystaniem JPA
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select * from....)

Dodałem plik META-INF/persistence.xml



```

1  <?xml version="1.0"?>
2  <persistence xmlns="http://java.sun.com/xml/ns/persistence"
3             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
5             http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
6             version="2.0">
7     <persistence-unit name="SzymonZukDatabase" transaction-type="RESOURCE_LOCAL">
8         <properties>
9             <property name="hibernate.connection.driver_class"
10                    value="org.apache.derby.jdbc.ClientDriver"/>
11             <property name="hibernate.connection.url"
12                    value="jdbc:derby://127.0.0.1/SzymonZukDatabase;create=true"/>
13             <property name="hibernate.show_sql" value="true" />
14             <property name="hibernate.format_sql" value="true" />
15             <property name="hibernate.hbm2ddl.auto" value="create-drop" />
16         </properties>
17     </persistence-unit>
18 </persistence>

```

Napisałem nowego maina, kod ma analogiczne działanie jak w poprzednim podpunkcie

```

public class MainJPA {

    public static void main(String[] args) {
        var emf = Persistence.createEntityManagerFactory("SzymonZukDatabase");
        var em = emf.createEntityManager();
        var etx = em.getTransaction();
        etx.begin();

        var tulipan = new Product("Tulipan", 20);
        var fiolek = new Product("Fiołek", 5);
        var storczyk = new Product("Storczyk", 3);
        var roza = new Product("Róża", 3);

        var invoice1 = new Invoice(1, 4, Set.of(tulipan, fiolek, storczyk, roza));
        tulipan.addCanBeSoldIn(invoice1);
        fiolek.addCanBeSoldIn(invoice1);
        storczyk.addCanBeSoldIn(invoice1);
        roza.addCanBeSoldIn(invoice1);

        var invoice2 = new Invoice(2, 3, Set.of(tulipan, fiolek, storczyk));
        tulipan.addCanBeSoldIn(invoice2);
        fiolek.addCanBeSoldIn(invoice2);
        storczyk.addCanBeSoldIn(invoice2);

        var invoice3 = new Invoice(3, 1, Set.of(tulipan));
        tulipan.addCanBeSoldIn(invoice3);

        var invoice4 = new Invoice(4, 1, Set.of(fiolek, roza));
        fiolek.addCanBeSoldIn(invoice4);
        roza.addCanBeSoldIn(invoice4);

        em.persist(tulipan);
        em.persist(fiolek);
        em.persist(storczyk);
        em.persist(roza);

        em.persist(invoice1);
        em.persist(invoice2);
        em.persist(invoice3);
        em.persist(invoice4);

        etx.commit();
    }
}

```

```
// Pokazanie produktów sprzedanych w ramach faktury nr. 1
etx = em.getTransaction();
etx.begin();

var query1 = em.createQuery("from Invoice as inv where inv.invoiceNumber=1");
var faktura = (Invoice) query1.getSingleResult();

// Pokazanie faktur w których sprzedano tulipany
var query2 = em.createQuery("from Product as prod where prod.productName='Tulipan'");
var produkt = (Product) query2.getSingleResult();

System.out.print("Produkty sprzedane w ramach faktury nr 1: ");
for (var p : faktura.getIncludes()) {
    System.out.print(p.getProductName() + " ");
}
System.out.print("\n");

System.out.print("Numery faktur, w których sprzedano tulipany: ");
for (var i : produkt.getCanBeSoldIn()) {
    System.out.print(i.getInvoiceNumber() + " ");
}
etx.commit();

em.close();
}
```

Wynik jest identyczny jak wcześniej

```
Produkty sprzedane w ramach faktury nr 1: Tulipan Róża Storczyk Fiołek
Numery faktur, w których sprzedano tulipany: 2 3 1
```

Schemat i zawartość bazy danych też jest identyczna

The screenshot shows a database management tool interface. On the left, there are three tables displayed:

- INVOICE**:

INVOICEID	INVOICENUMBER	QUANTITY
1	1	4
2	2	3
3	3	1
4	4	1
- INVOICE_PRODUCT**:

CANBESOLDIN_INVOICEID	INCLUDES_PRODUCTID
1	1
2	1
3	1
4	1
5	2
6	2
7	2
8	3
9	4
10	4
- PRODUCT**:

CATEGORY_FK	SUPPLIER_FK	UNITSINSTOCK	PRODUCTID	PRODUCTNAME
<null>	<null>	1	20	Tulipan
<null>	<null>	2	5	Fiołek
<null>	<null>	3	3	Storczyk
<null>	<null>	4	3	Róża

On the right, there is a 'Database' panel showing the schema structure. It includes a tree view of the database 'SzymonZukDatabase' with tables: CATEGORY, INVOICE, INVOICE_PRODUCT, PRODUCT, and SUPPLIER. Below the tree, there is a 'Visualization for INVOICE_PRODUCT' diagram showing the relationships between the tables. The diagram shows that INVOICE_PRODUCT is linked to INVOICE and PRODUCT. The INVOICE table has columns INVOICENUMBER and INVOICEID. The PRODUCT table has columns CATEGORY_FK, SUPPLIER_FK, UNITSINSTOCK, and PRODUCTID. The INVOICE_PRODUCT table has columns CANBESOLDIN_INVOICEID and INCLUDES_PRODUCTID.

Podpunkt 7

VII. 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/schemat bazy danych, select * from....)**

Modyfikacje modelu danych

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productId;
    private String productName;
    private int unitsInStock;
    @ManyToOne
    @JoinColumn(name="SUPPLIER_FK")
    private Supplier isSuppliedBy;
    @ManyToOne
    @JoinColumn(name="CATEGORY_FK")
    private Category category;
    // Dodałem cascade
    @ManyToMany(mappedBy="includes", cascade = CascadeType.PERSIST)
    private Set<Invoice> canBeSoldIn = new HashSet<>();

    /* ... */
}
```

```
@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int invoiceId;
    private int invoiceNumber;
    private int quantity;
    // Dodałem cascade
    @ManyToMany(cascade = CascadeType.PERSIST)
    private Set<Product> includes = new HashSet<>();

    /* ... */
}
```

W mainie napisałem kod tworzący faktury wraz z nowymi produktami (bez zapisywania explicite) i odwrotnie

```
public static void main(String[] args) {
    var emf = Persistence.createEntityManagerFactory("SzymonZukDatabase");
    var em = emf.createEntityManager();
    var etx = em.getTransaction();
    etx.begin();

    var tulipan = new Product("Tulipan", 20);
    var fiolek = new Product("Fiołek", 5);
```

```

var invoice1 = new Invoice(1, 2, Set.of(tulipan, fiolek));
tulipan.addCanBeSoldIn(invoice1);
fiolek.addCanBeSoldIn(invoice1);

em.persist(invoice1);

var storczyk = new Product("Storczyk", 3);

var invoice2 = new Invoice(2, 1, Set.of(storczyk));
var invoice3 = new Invoice(3, 1, Set.of(storczyk));
storczyk.addCanBeSoldIn(invoice2);
storczyk.addCanBeSoldIn(invoice3);

em.persist(storczyk);

etx.commit();

em.close();
}

```

Po uruchomieniu aplikacji wszystkie produkty i faktury zostały poprawnie zapisane w bazie danych

The screenshot displays a database management interface with three tables open:

- INVOICE**: 3 rows. Columns: INVOICEID, INVOICENUMBER, QUANTITY. Data: (1, 1, 2), (2, 2, 1), (3, 3, 1).
- INVOICE_PRODUCT**: 4 rows. Columns: CANBESOLDIN_INVOICEID, INCLUDES_PRODUCTID. Data: (1, 1), (2, 1), (3, 2), (4, 3).
- PRODUCT**: 3 rows. Columns: CATEGORY_FK, SUPPLIER_FK, PRODUCTID, UNITSINSTOCK, PRODUCTNAME. Data: (null, null, 1, 5, Fiolek), (null, null, 2, 20, Tulipan), (null, null, 3, 3, Storczyk).

The right sidebar shows the database structure for 'SzymonZukDatabase' with tables: CATEGORY, INVOICE, INVOICE_PRODUCT, PRODUCT, and SUPPLIER, each with its respective columns, keys, and indexes.

Podpunkt 8

VIII. Embedded class

- Dodaj do modelu klasę adres. „Wbuduj” ją do tabeli Dostawców.
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat, select * from....)**
- Zmodyfikuj model w taki sposób, że dane adresowe znajdują się w klasie dostawców. Zmapuj to do dwóch osobnych tabel.
- Udokumentuj wykonane kroki oraz uzyskane rezultaty (logi wywołań sqlowych, describe table/schemat bazy danych, select * from....)**

Najpierw zrobiłem wariant z mapowaniem do osobnych tabel, bo wymagał mniejszej ilości modyfikacji

Wariant z mapowaniem do osobnych tabel

Modyfikacje modelu danych

```
@Entity
// Dodałem tabelę adresów
@SecondaryTable(name="SUPPLIER_ADDRESS")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierId;
    private String companyName;
    // Zmapowałem street i city na tabelę adresów
    @Column(table="SUPPLIER_ADDRESS")
    private String street;
    @Column(table="SUPPLIER_ADDRESS")
    private String city;
    @OneToMany
    @JoinColumn(name="SUPPLIER_FK")
    private Set<Product> supplies = new HashSet<>();

    /* ... */
}
```

Napisałem kod w mainie tworzący dwóch dostawców

```
var emf = Persistence.createEntityManagerFactory("SzymonZukDatabase");
var em = emf.createEntityManager();
var etx = em.getTransaction();
etx.begin();

var supplier1 = new Supplier("Inpost", "Kawiory", "Kraków");
var supplier2 = new Supplier("DHL", "Myślenicka", "Warszawa");

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

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

W wyniku działania kodu została utworzona tabela adresów

```

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

```

Schemat i zawartość bazy danych

SUPPLIER_ADDRESS x

WHERE

SUPPLIERID

CITY

STREET

1

1

Kraków

Kawitory

2

2

Warszawa

Myslenicka

ORDER BY

SUPPLIER x

WHERE

SUPPLIERID

COMPANYNAME

1

1

Inpost

2

2

DHL

ORDER BY

Visualization for SUPPLIER_ADDRESS

SUPPLIER

COMPANYNAME

varchar(255)

SUPPLIERID

integer

SUPPLIERID

SUPPLIER_ADDRESS

CITY

varchar(255)

STREET

varchar(255)

SUPPLIERID

integer

Database

SzymonZukDatabase 1 of 11

APP

tables 6

CATEGORY

columns 2

keys 1

indexes 1

INVOICE

columns 3

keys 1

indexes 1

INVOICE_PRODUCT

columns 2

keys 1

foreign keys 2

indexes 3

PRODUCT

columns 5

keys 1

foreign keys 2

indexes 3

SUPPLIER

columns 2

keys 1

indexes 1

SUPPLIER_ADDRESS

columns 3

keys 1

foreign keys 1

indexes 2

Wariant z wbudowaną klasą

Modyfikacje modelu danych

```
// Dodałem klasę Address (i mapping do niej)
@Embeddable
public class Address {
    private String city;
    private String street;

    public Address() {}

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

```
@Entity
// Usunąłem tabelę adresów
// @SecondaryTable(name="SUPPLIER_ADDRESS")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierId;
    private String companyName;
    // Usunąłem pola związane z adresem
    // @Column(table="SUPPLIER_ADDRESS")
    // private String street;
    // @Column(table="SUPPLIER_ADDRESS")
    // private String city;
    @OneToMany
    @JoinColumn(name="SUPPLIER_FK")
    private Set<Product> supplies = new HashSet<>();
    // Dodałem wbudowane pole adresu
    @Embedded
    private Address address;

    public Supplier() {}

    // Zmodyfikowałem konstruktor, tak aby przyjmował klasę adresu
    public Supplier(String companyName, Address address) {
        this.companyName = companyName;
        this.address = address;
    }

    public void addSuppliedProduct(Product product) {
        supplies.add(product);
    }
}
```

Zmodyfikowałem kod w mainie tworzący dwóch dostawców

```
var emf = Persistence.createEntityManagerFactory("SzymonZukDatabase");
var em = emf.createEntityManager();
var etx = em.getTransaction();
etx.begin();

var address1 = new Address("Kawior", "Kraków");
var address2 = new Address("Myślenicka", "Warszawa");

var supplier1 = new Supplier("Inpost", address1);
var supplier2 = new Supplier("DHL", address2);

em.persist(supplier1);
em.persist(supplier2);
```



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

Po uruchomieniu aplikacji stworzyła się tabela dostawców zawierająca dane adresowe (czyli odwrotnie jak w poprzednim wariantcie)

Hibernate:

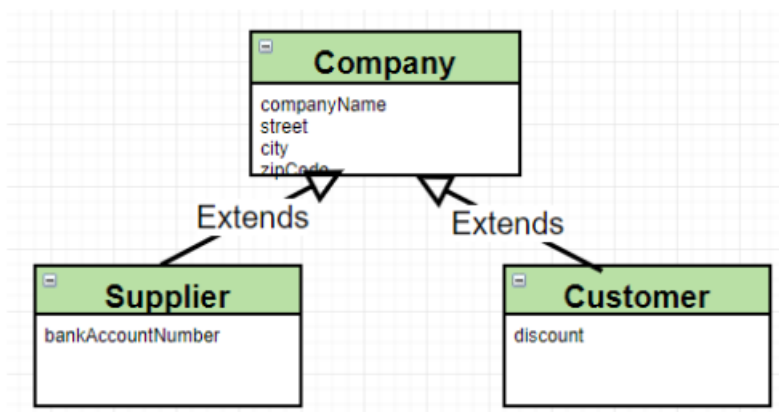
```
create table Supplier (
    supplierId integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    primary key (supplierId)
)
```

Zawartość tabeli suppliers

SUPPLIER					Database				
<div> <div>2 rows</div> <div> <div>WHERE</div> <div>ORDER BY</div> </div> </div>					<div> <div>SzymonZukDatabase 1 of 11</div> <div> <div>APP</div> <div> <div>tables 5</div> <div> <div>CATEGORY</div> <div>INVOICE</div> <div>INVOICE_PRODUCT</div> <div>PRODUCT</div> <div>SUPPLIER</div> </div> </div> </div> </div>				
SUPPLIERID	CITY	COMPANYNAME	STREET						
1	Kawior	Inpost	Kraków						
2	Myślenicka	DHL	Warszawa						

Podpunkt 9

- 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 słowowych, describe table/schemat bazy danych, select * from....)

Wariant z jedną tabelą

Modyfikacje modelu danych

```
// Dodałem klasę firmy
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
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() {}

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

    @Override
    public String toString() {
        return "Company(CompanyId: " + companyId + ", CompanyName: " +
            companyName + ", City: " + city + ", Street: "
            + street + ", ZipCode: " + zipCode + ")";
    }
}
```

```
// Zmodyfikowałem klasę dostawcy wprowadzając dziedziczenie z firmy
@Entity
public class Supplier extends Company {
    private String bankAccountNumber;

    public Supplier() {}

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

    @Override
    public String toString() {
        return "Supplier(CompanyId: " + companyId + ", CompanyName: " +
            companyName + ", City: " + city + ", Street: "
            + street + ", ZipCode: " + zipCode + ", BankAccountNumber: " + bankAccountNumber + ")";
    }
}
```

```
// Dodałem klasę klienta
@Entity
public class Customer extends Company {
    private double discount;

    public Customer() {}

    public Customer(String companyName, String city, String street, String zipCode, double discount) {
```

```

        super(companyName, city, street, zipCode);
        this.discount = discount;
    }

    @Override
    public String toString() {
        return "Customer(CompanyId: " + companyId + ", CompanyName: " +
            companyName + ", City: " + city + ", Street: " +
            street + ", ZipCode: " + zipCode + ", Discount: " + discount + ")";
    }
}

```

Napisałem kod w mainie dodający dostawców i klientów, a następnie wydobywający ich z bazy danych z dziedziczących agregatów i z agregatu Company

```

public static void main(String[] args) {
    var emf = Persistence.createEntityManagerFactory("SzymonZukDatabase");
    var em = emf.createEntityManager();
    var etx = em.getTransaction();
    etx.begin();

    var supplier1 = new Supplier("Inpost", "Kraków", "Kawiory", "00-123",
        "PL37 5269 6062 3118 6527 8335 6401");
    var supplier2 = new Supplier("DHL", "Warszawa", "Myślenicka", "13-125",
        "PL91 1378 1462 6908 8595 1014 0748");

    var customer1 = new Customer("Wedel", "Radom", "Łużycka", "72-244", 0.05);
    var customer2 = new Customer("Milka", "Kraków", "Basztowa", "82-214", 0.05);

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

    var query1 = em.createQuery("from Supplier as s " +
        "where s.bankAccountNumber='PL91 1378 1462 6908 8595 1014 0748'", Supplier.class);

    var query2 = em.createQuery("from Customer as c where c.zipCode='72-244'", Customer.class);

    var query3 = em.createQuery("from Company as c where c.city='Kraków'", Company.class);

    var supplier = query1.getSingleResult();
    var customer = query2.getSingleResult();
    var companies = query3.getResultList();

    System.out.println("Dostawca z numerem konta w banku PL91 1378 1462 6908 8595 1014 0748:");
    System.out.println(supplier);
    System.out.println("Klient z kodem pocztowym 13-125:");
    System.out.println(customer);
    System.out.println("Firmy z Krakowa:");
    for (var c : companies) {
        System.out.println(c);
    }

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

```

Wynik działania kodu

Hibernate:

```
create table Company (
    companyId integer not null,
    discount float(52),
    DTYPE varchar(31) not null,
    bankAccountNumber varchar(255),
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    zipCode varchar(255),
    primary key (companyId)
)
```

Dostawca z numerem konta w banku PL91 1378 1462 6908 8595 1014 0748:

Supplier(CompanyId: 2, CompanyName: DHL, City: Warszawa, Street: Myślenicka, ZipCode: 13-125, BankAccountNumber: PL91 1378 1462 6908 8595 1014 0748)

Klient z kodem pocztowym 13-125:

Customer(CompanyId: 3, CompanyName: Wedel, City: Radom, Street: Łużycka, ZipCode: 72-244, Discount: 0.05)

Firmy z Krakowa:

Supplier(CompanyId: 1, CompanyName: Inpost, City: Kraków, Street: Kawiorzy, ZipCode: 00-123, BankAccountNumber: PL37 5269 6062 3118 6527 8335 6401)

Customer(CompanyId: 4, CompanyName: Milka, City: Kraków, Street: Basztowa, ZipCode: 82-214, Discount: 0.05)

Struktura tabeli i jej zawartość w bazie danych

	COMPANYID	DISCOUNT	DTYPE	BANKACCOUNTNUMBER	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	<null>	Supplier	PL37 5269 6062 3118 6527 8335 6401	Kraków	Inpost	Kawiorzy	00-123
2	2	<null>	Supplier	PL91 1378 1462 6908 8595 1014 0748	Warszawa	DHL	Myślenicka	13-125
3	3	0.05	Customer	<null>	Radom	Wedel	Łużycka	72-244
4	4	0.05	Customer	<null>	Kraków	Milka	Basztowa	82-214

Wariant z tabelami łączonymi**Modyfikacje modelu danych**

```
@Entity
// Zmieniłem InheritanceType na JOINED
@Inheritance(strategy=InheritanceType.JOINED)
public class Company {

    /* ... */

}
```

Nie zmieniałem kodu w mainie. Otrzymałem ten sam wynik w zapytaniach wydobywających obiekt z bazy danych co wcześniej. Zmieniła się struktura tworzonych tabel

Hibernate:

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

Hibernate:

```
create table Customer (
    companyId integer not null,
    discount float(52) not null,
    primary key (companyId)
)
```

Hibernate:

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

Schemat bazy danych i jej zawartość

The screenshot shows a database management tool interface. The main window displays the 'COMPANY' table with 4 rows of data. The 'SUPPLIER' table has 2 rows, and the 'CUSTOMER' table has 2 rows. On the right, a 'Database' pane shows the schema structure, including tables, columns, keys, and indexes. At the bottom, a 'Visualization for CATEGORY' pane shows a diagram of the database schema, highlighting the 'COMPANY' table and its relationships with 'SUPPLIER' and 'CUSTOMER'.

COMPANY Table Data:

COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	Kraków	Inpost	Kawiony	00-123
2	Warszawa	DHL	Myślenicka	13-125
3	Radom	Wedeł	Łużycka	72-244
4	Kraków	Milka	Basztowa	82-214

SUPPLIER Table Data:

COMPANYID	BANKACCOUNTNUMBER
1	PL37 5269 6062 3118 6527 8335 6401
2	PL91 1378 1462 6908 8595 1014 0748

CUSTOMER Table Data:

COMPANYID	DISCOUNT
1	3 0.05
2	4 0.05

Schema Diagram:

```

graph TD
    COMPANY[COMPANY] --> SUPPLIER[SUPPLIER]
    COMPANY --> CUSTOMER[CUSTOMER]
    style COMPANY fill:#fff,stroke:#f00,stroke-width:2px
    style SUPPLIER fill:#fff,stroke:#f00,stroke-width:2px
    style CUSTOMER fill:#fff,stroke:#f00,stroke-width:2px
  
```

Wariant z jedną tabelą na każdą klasę

Modyfikacje modelu danych

```
@Entity
// Zmieniłem InheritanceType na TABLE_PER_CLASS
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Company {

    /* ... */

}
```

Znów nie zmieniałem kodu w mainie. Otrzymałem ten sam wynik w zapytaniach wydobywających obiekt z bazy danych co wcześniej. Zmieniła się struktura tworzonych tabel

Hibernate:

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

Hibernate:

```
create table Customer (
    companyId integer not null,
    discount float(52) not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    zipCode varchar(255),
    primary key (companyId)
)
```

Hibernate:

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

Schemat bazy danych i jej zawartość. Jak widać tabela Company jest pusta, bo każda stworzona firma była dostawcą lub klientem

The screenshot displays a database management interface for 'SzymonZukDatabase'. The main window shows three tables: COMPANY, SUPPLIER, and CUSTOMER. The COMPANY table is empty. The SUPPLIER table contains two rows of data. The CUSTOMER table contains two rows of data. The right sidebar shows the database structure, including tables, columns, keys, and indexes.

COMPANYID	BANKACCOUNTNUMBER	CITY	COMPANYNAME	STREET	ZIPCODE
1	PL37 5269 6062 3118 6527 8335 6401	Kraków	Inpost	Kawiorzy	00-123
2	PL91 1378 1462 6908 8595 1014 0748	Warszawa	DHL	Myślenicka	13-125

COMPANYID	DISCOUNT	CITY	COMPANYNAME	STREET	ZIPCODE
1	3	0.05 Radom	Wedel	Łużycka	72-244
2	4	0.05 Kraków	Milka	Basztowa	82-214