Hibernate, JPA

Autorzy: Szymon Migas, Dawid Żak

Część I

Sprawdzamy zainstalowaną wersję Javy:

```
java -version
```

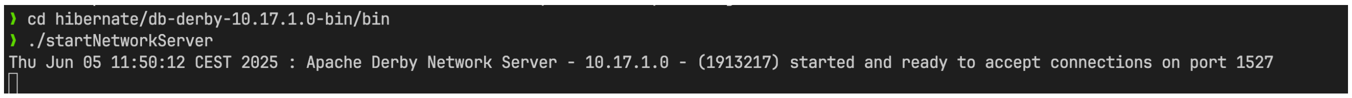
Wynik:

```
java version "21.0.2" 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
```

Po pobraniu i rozpakowaniu serwera Derby, przechodzimy do katalogu `bin` i uruchamiamy serwer:

```
cd hibernate/db-derby-10.17.1.0-bin/bin
./startNetworkServer
```

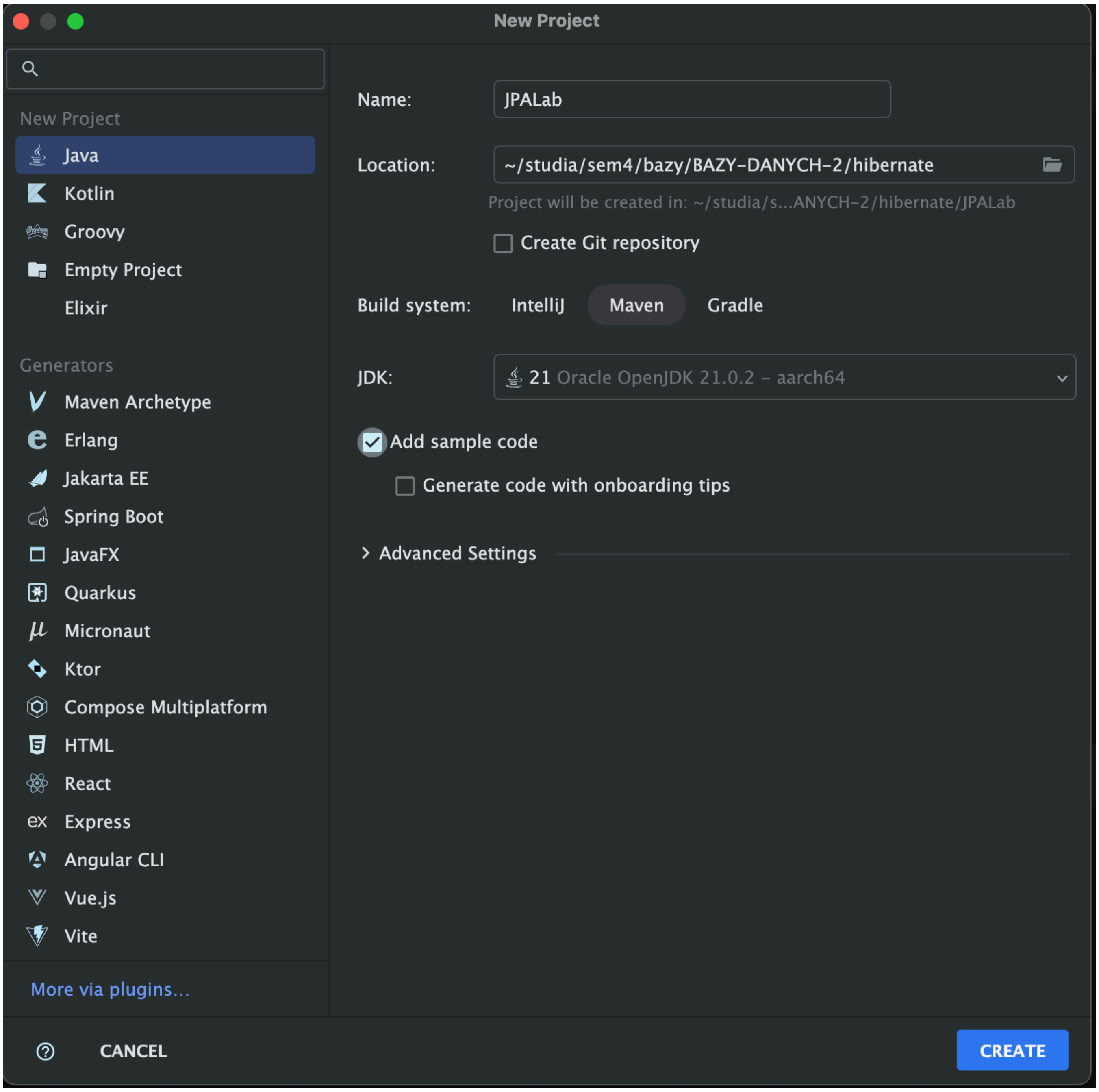
Wynik na konsoli:



```
❯ cd hibernate/db-derby-10.17.1.0-bin/bin
❯ ./startNetworkServer
Thu Jun 05 11:50:12 CEST 2025 : Apache Derby Network Server - 10.17.1.0 - (1913217) started and ready to accept connections on port 1527
```

Następnie w Intelliju tworzymy nowy projekt Mavenowy, w którym dodajemy zależności do Hibernate i Derby.

Tworzenie nowego projektu Mavenowego:



Name: JPALab

Location: ~/studia/sem4/bazy/BAZY-DANYCH-2/hibernate

Project will be created in: ~/studia/s...ANYCH-2/hibernate/JPALab

☐ Create Git repository

Build system: IntelliJ [✓ Maven] Gradle

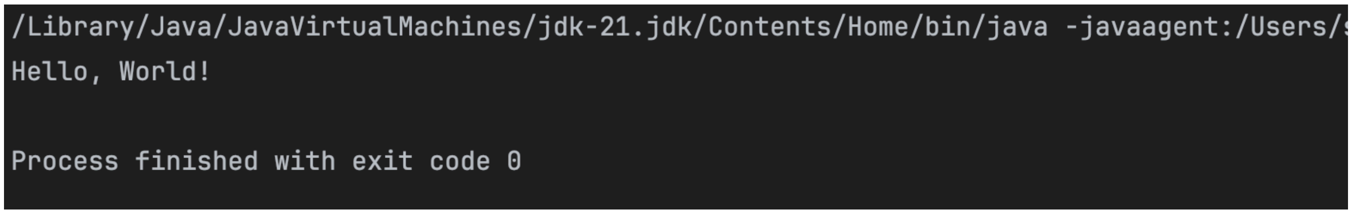
JDK: 21 Oracle OpenJDK 21.0.2 - aarch64

☑ Add sample code

☐ Generate code with onboarding tips

> Advanced Settings

Testowe uruchomienie projektu:

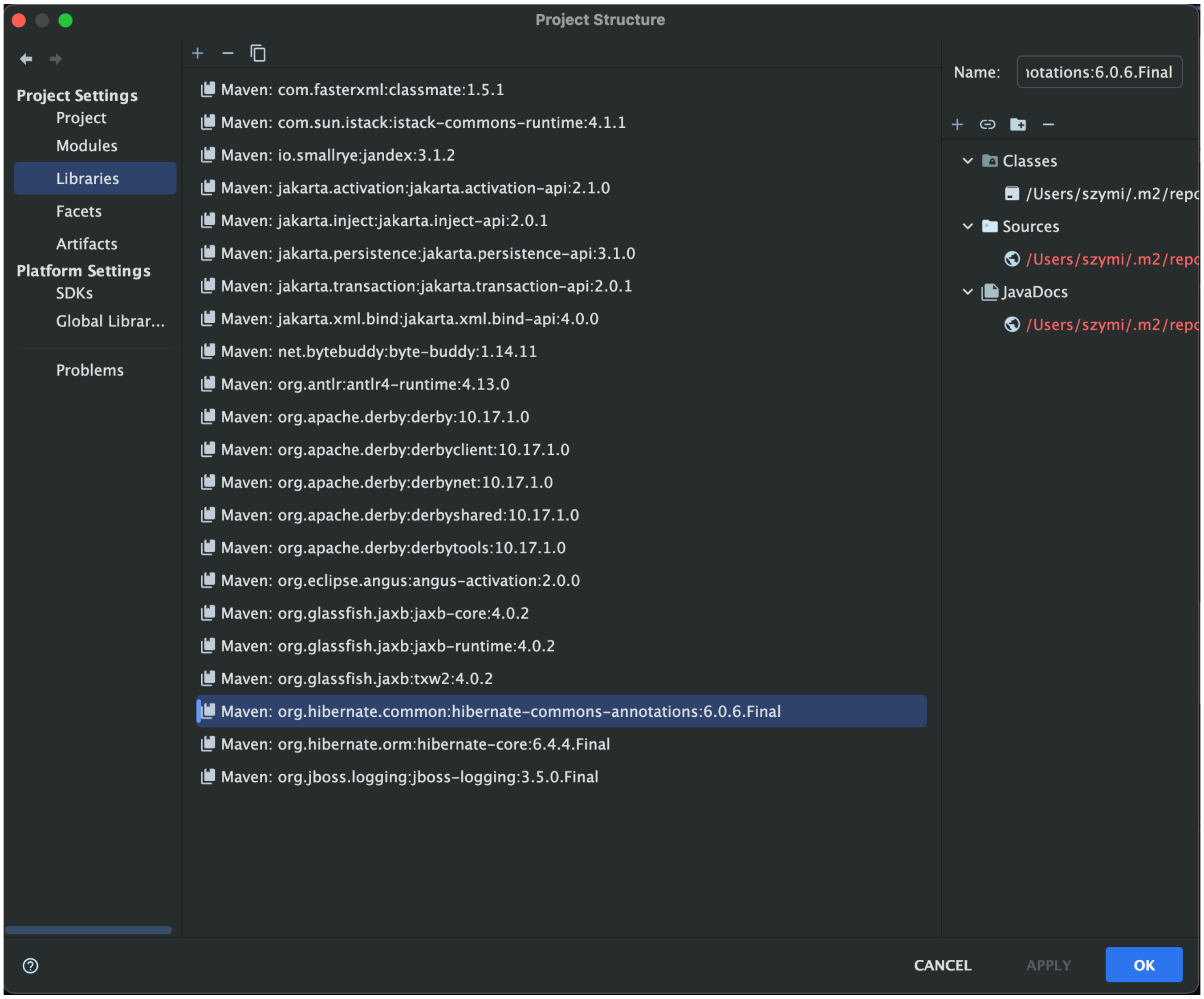


```
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Users/
Hello, World!

Process finished with exit code 0
```

Dodajemy zależności do pliku `pom.xml`:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
...

    <dependencies>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.4.4.Final</version>
        </dependency>

        <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derbyclient</artifactId>
            <version>10.17.1.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.derby</groupId>
            <artifactId>derbynet</artifactId>
            <version>10.17.1.0</version>
        </dependency>
    </dependencies>

</project>
```

Moduły w projekcie po przeładowaniu zależności:


Project Structure

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Librar...
Problems

```
Maven: com.fasterxml:classmate:1.5.1
Maven: com.sun.istack:istack-commons-runtime:4.1.1
Maven: io.smallrye:jandex:3.1.2
Maven: jakarta.activation:jakarta.activation-api:2.1.0
Maven: jakarta.inject:jakarta.inject-api:2.0.1
Maven: jakarta.persistence:jakarta.persistence-api:3.1.0
Maven: jakarta.transaction:jakarta.transaction-api:2.0.1
Maven: jakarta.xml.bind:jakarta.xml.bind-api:4.0.0
Maven: net.bytebuddy:byte-buddy:1.14.11
Maven: org.antlr:antlr4-runtime:4.13.0
Maven: org.apache.derby:derby:10.17.1.0
Maven: org.apache.derby:derbyclient:10.17.1.0
Maven: org.apache.derby:derbynet:10.17.1.0
Maven: org.apache.derby:derbyshared:10.17.1.0
Maven: org.apache.derby:derbytools:10.17.1.0
Maven: org.eclipse.angus:angus-activation:2.0.0
Maven: org.glassfish.jaxb:jaxb-core:4.0.2
Maven: org.glassfish.jaxb:jaxb-runtime:4.0.2
Maven: org.glassfish.jaxb:txw2:4.0.2
Maven: org.hibernate.common:hibernate-commons-annotations:6.0.6.Final
Maven: org.hibernate.orm:hibernate-core:6.4.4.Final
Maven: org.jboss.logging:jboss-logging:3.5.0.Final
```

Name: ıotations:6.0.6.Final

Classes
/Users/szymi/.m2/repo
Sources
/Users/szymi/.m2/repo
JavaDocs
/Users/szymi/.m2/repo

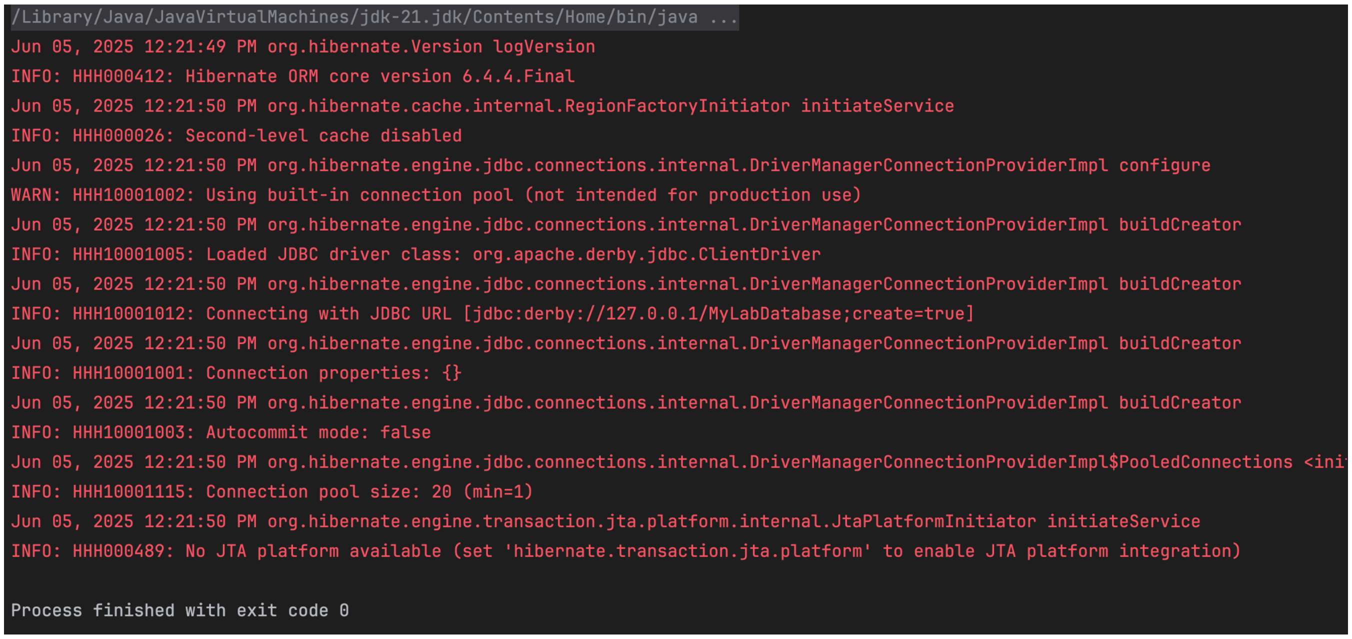
Kofiguracja Hibernate w pliku `hibernate.cfg.xml`:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>

        <property name="connection.driver_class">
            org.apache.derby.jdbc.ClientDriver
        </property>

        <property name="connection.url">
            jdbc:derby://127.0.0.1/MyLabDatabase;create=true
        </property>


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

    </session-factory>
</hibernate-configuration>
```

Uzupełniamy klasę `Main`:

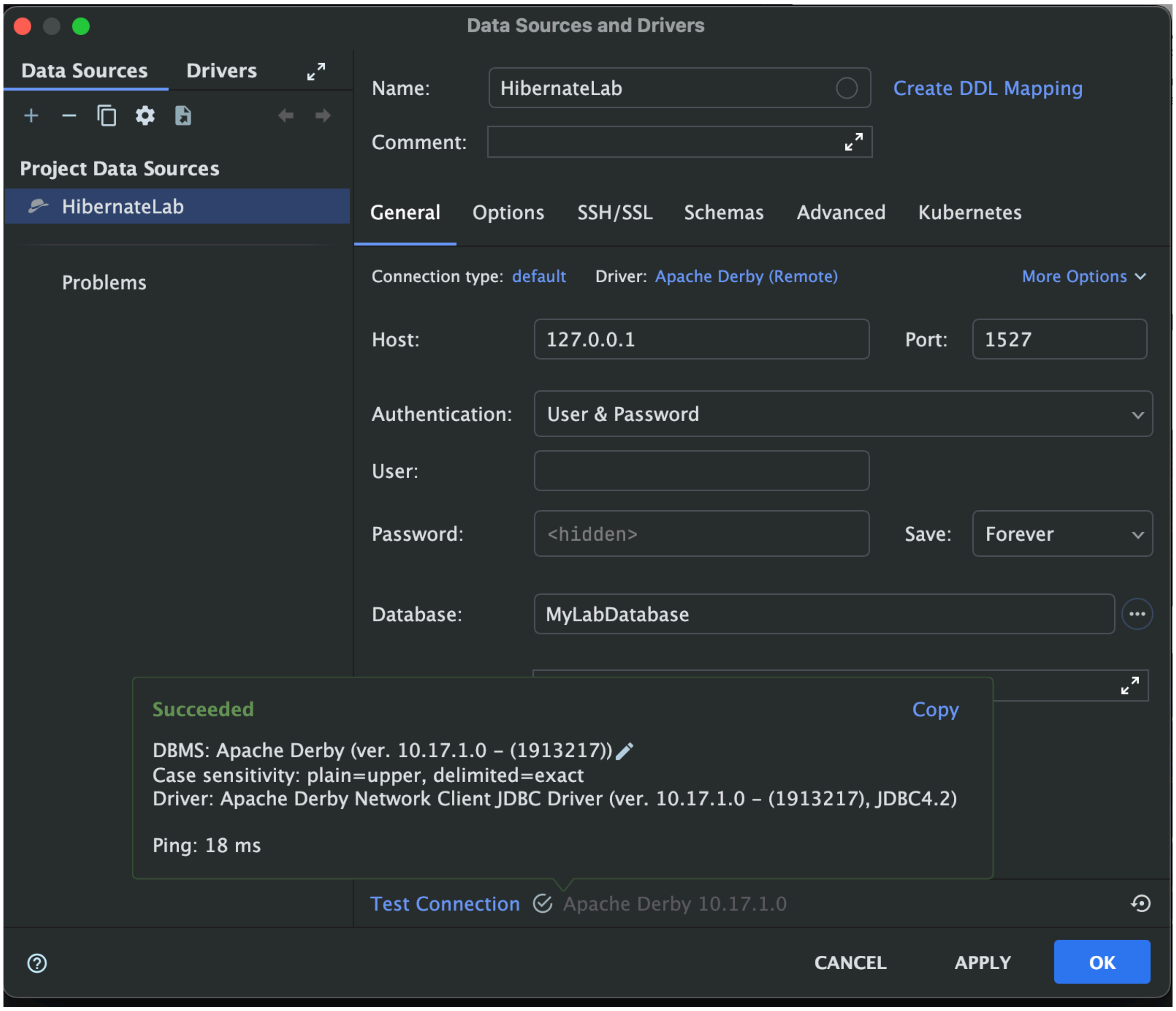
```java
package org.example;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class Main {
    private static SessionFactory sessionFactory = null;

    public static void main(String[] args) {
        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();
        session.close();
    }

    private static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            Configuration configuration = new Configuration();
            sessionFactory = configuration.configure().buildSessionFactory();
        }
        return sessionFactory;
    }
}
```

Uruchamiamy projekt, aby sprawdzić, czy Hibernate poprawnie łączy się z bazą danych Derby:



```
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java ...
Jun 05, 2025 12:21:49 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate ORM core version 6.4.4.Final
Jun 05, 2025 12:21:50 PM org.hibernate.cache.internal.RegionFactoryInitiator initiateService
INFO: HHH000026: Second-level cache disabled
Jun 05, 2025 12:21:50 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using built-in connection pool (not intended for production use)
Jun 05, 2025 12:21:50 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: Loaded JDBC driver class: org.apache.derby.jdbc.ClientDriver
Jun 05, 2025 12:21:50 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001012: Connecting with JDBC URL [jdbc:derby://127.0.0.1/MyLabDatabase;create=true]
Jun 05, 2025 12:21:50 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {}
Jun 05, 2025 12:21:50 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
Jun 05, 2025 12:21:50 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <ini
INFO: HHH10001115: Connection pool size: 20 (min=1)
Jun 05, 2025 12:21:50 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
```

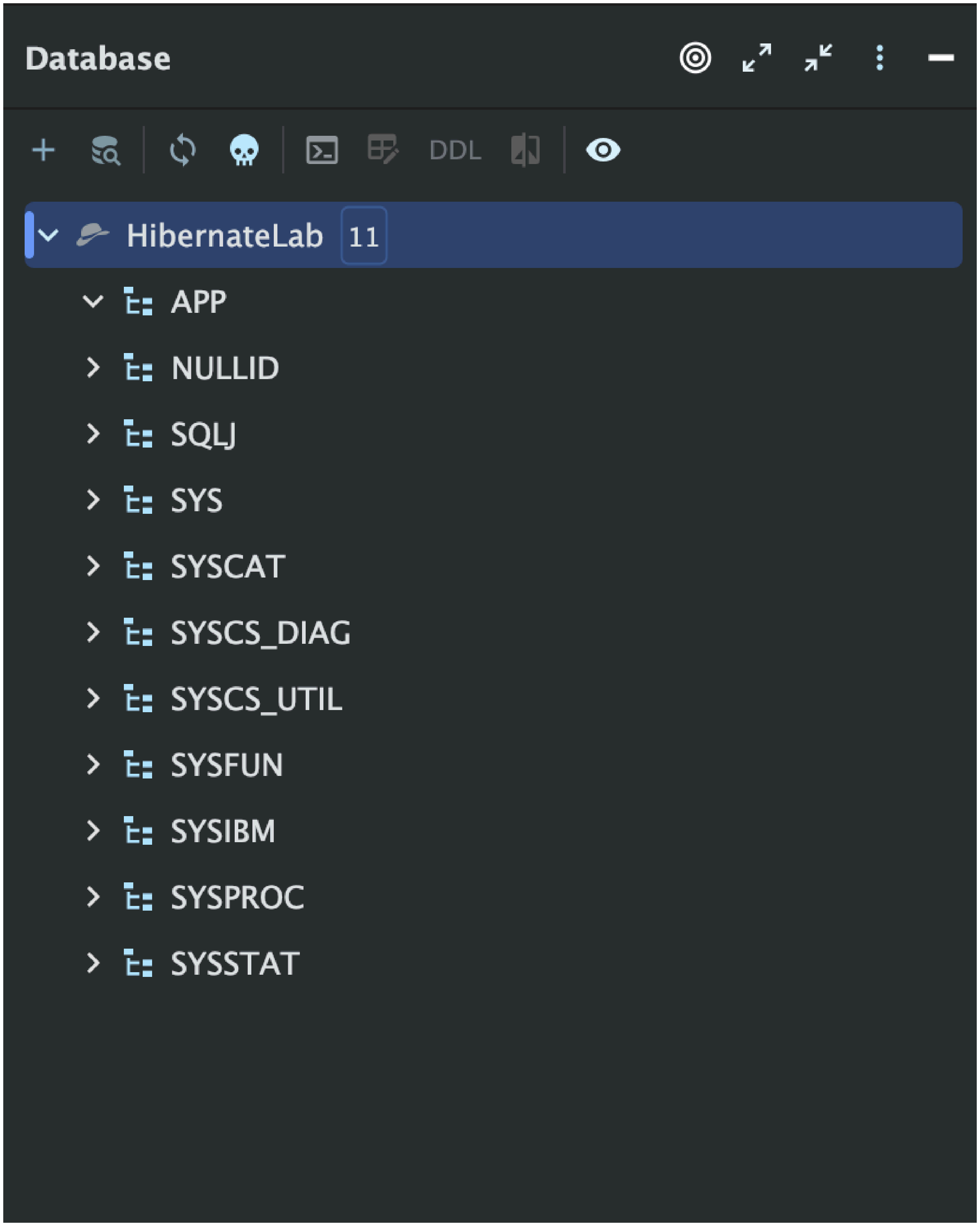
Dodajemy do Intellij bazę danych Derby:


Name: HibernateLab
Comment:
Connection type: default Driver: Apache Derby (Remote)
Host: 127.0.0.1 Port: 1527
Authentication: User & Password
User:
Password: <hidden> Save: Forever
Database: MyLabDatabase

```
Succeeded
DBMS: Apache Derby (ver. 10.17.1.0 - (1913217))
Case sensitivity: plain=upper, delimited=exact
Driver: Apache Derby Network Client JDBC Driver (ver. 10.17.1.0 - (1913217), JDBC4.2)

Ping: 18 ms
```

Test Connection ✓ Apache Derby 10.17.1.0

Widać że połączenie się powiodło, póki co otrzymujemy:



```
HibernateLab 11
  APP
  NULLID
  SQLJ
  SYS
  SYSCAT
  SYSCS_DIAG
  SYSCS_UTIL
  SYSFUN
  SYSIBM
  SYSPROC
  SYSSTAT
```

Następnie w celu dodania do bazy danych nowych danych, tworzymy klasę

`Product`:

```java
package org.example;

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

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

    public Product() {

    }

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

Oraz w klasie `Main` dodajemy sekcję tworzącą nową transakcję i zapisującą do bazy danych nowy produkt:

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

// ********************************************************************
        Product product = new Product("Kredki", 100);
        Transaction tx = session.beginTransaction();
        session.persist(product);
        tx.commit();
 // ********************************************************************

        session.close();
    }

    ...
```

Ponieważ

```
    transaction.save(...);
```

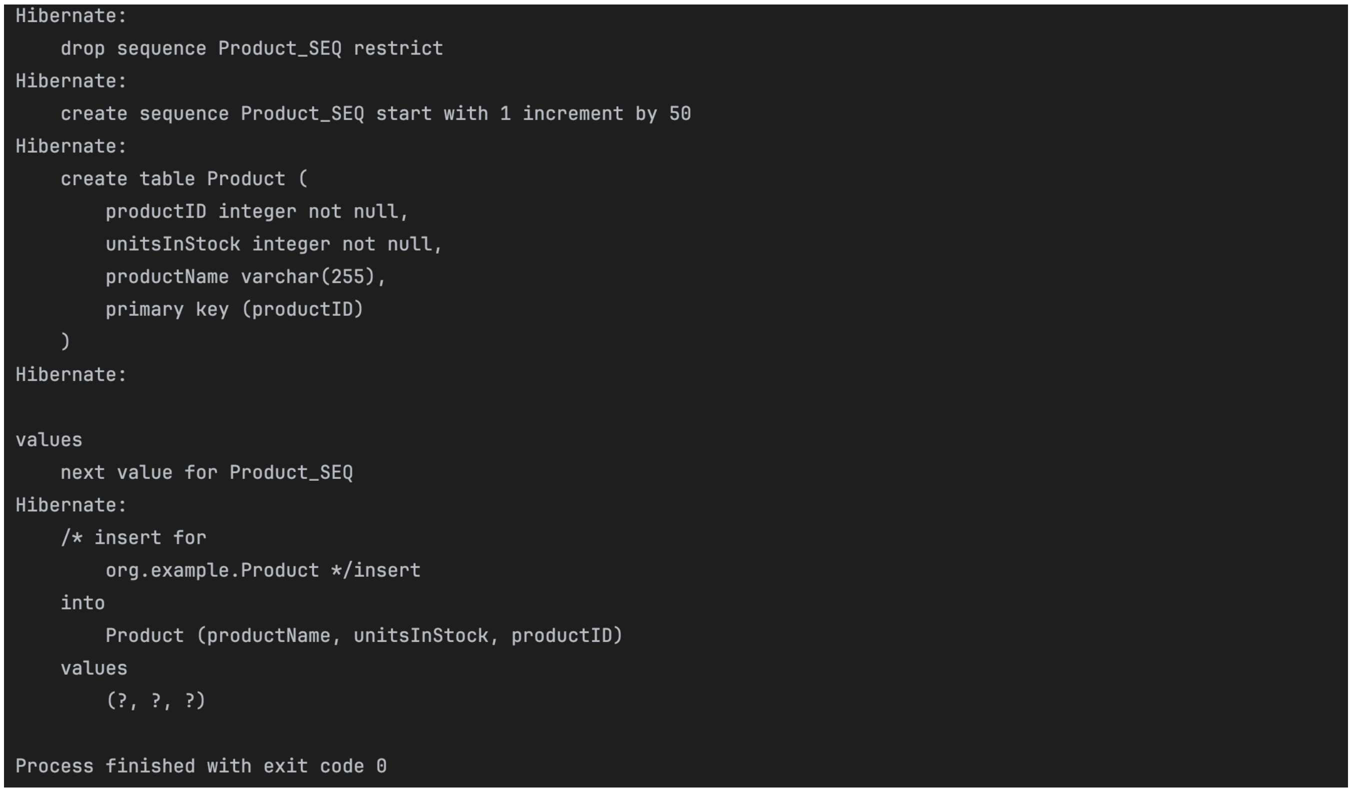
jest przestarzałe od wersji 6.0, używamy:

```
    session.persist(...);
```

Przed uruchomieniem projektu, dodajemy mapping klasy `Product` w pliku `hibernate.cfg.xml`:

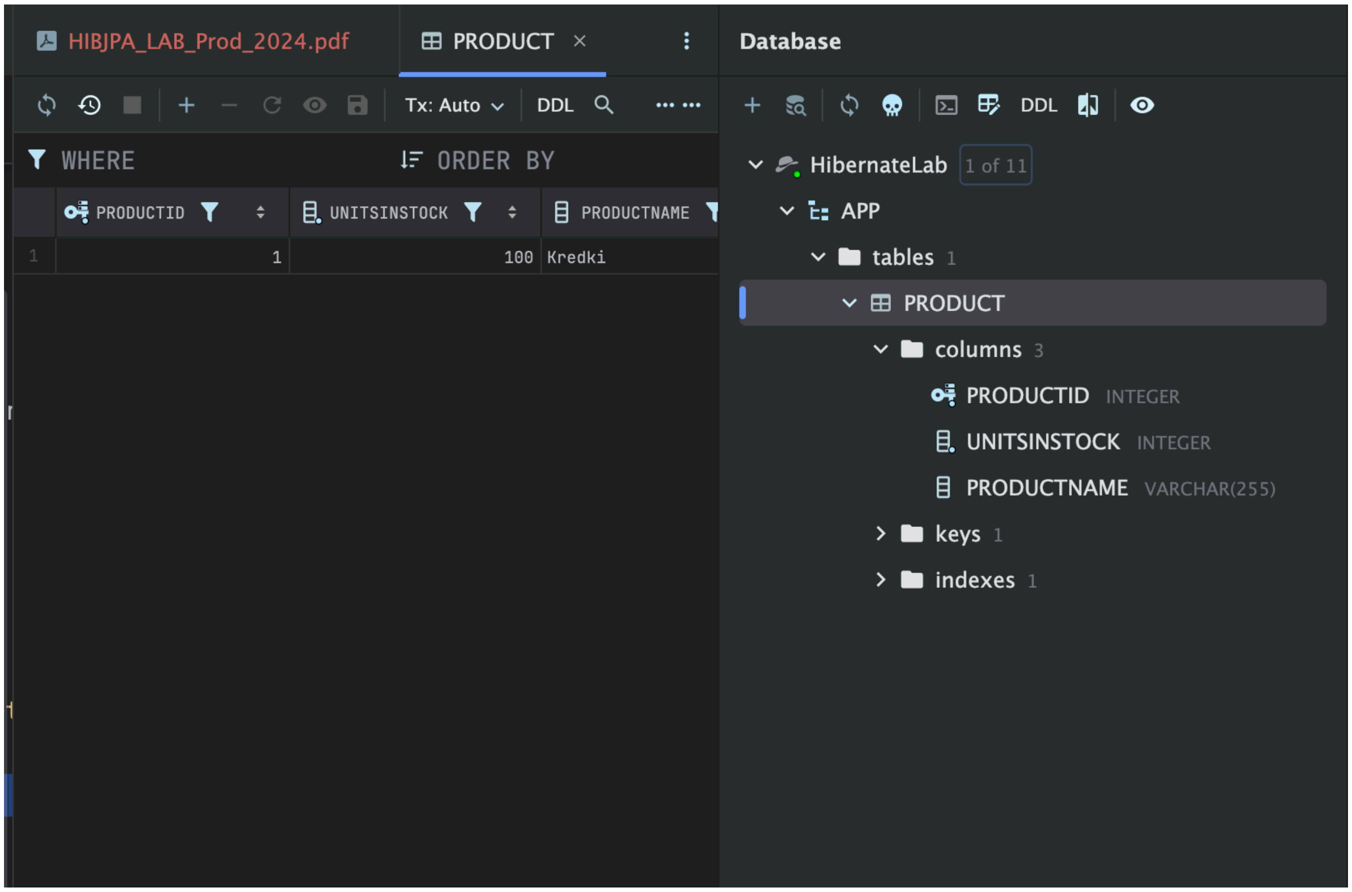
```
...
    <mapping class="org.example.Product"/>
...
```

Wynik po uruchomieniu:



```
Hibernate:
    drop sequence Product_SEQ restrict
Hibernate:
    create sequence Product_SEQ start with 1 increment by 50
Hibernate:
    create table Product (
        productID integer not null,
        unitsInStock integer not null,
        productName varchar(255),
        primary key (productID)
    )
Hibernate:

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

Process finished with exit code 0
```

W bazie danych pojawił się nowy produkt:



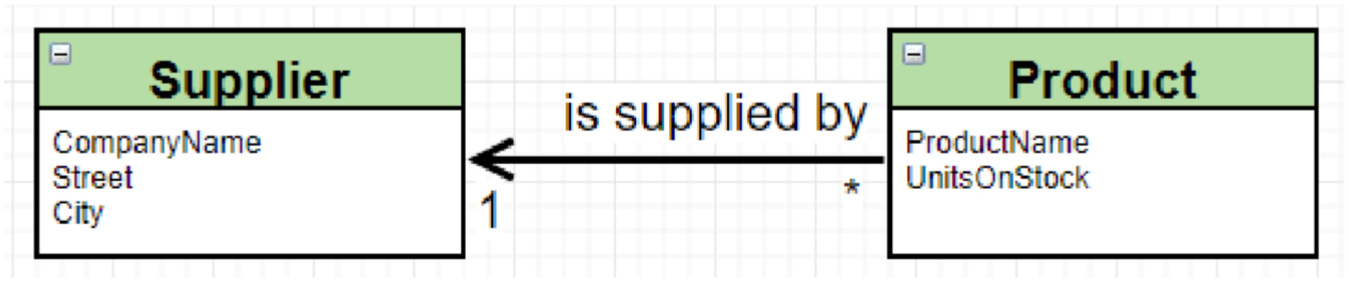
	PRODUCTID	UNITSINSTOCK	PRODUCTNAME
1	1	100	Kredki

```
HibernateLab  1 of 11
  APP
    tables 1
      PRODUCT
        columns 3
          PRODUCTID INTEGER
          UNITSINSTOCK INTEGER
          PRODUCTNAME VARCHAR(255)
        keys 1
        indexes 1
```

Część II

Zadanie I

Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej:



- stwórz nowego dostawcę
- Znajdz poprzednio wprowadzony produkt i ustaw jego dostawce na właśnie dodanego

Zaczynamy od stworzenia klasy `Supplier`:

```java
package org.example;

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

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

Zmieniona klasa `Product`:

```
...
@Entity
public class Product {
    ...
    @ManyToOne
    @JoinColumn(name = "supplierID")
    private Supplier supplier;

    ...

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

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

Aby zmienić zachowanie hibernate, podczas uruchamiania projektu zmieniamy plik `hibernate.cfg.xml`:

```
 <property name="hbm2ddl.auto">update</property>
```

Zmiana ta spowoduje, że Hibernate zaktualizuje schemat bazy danych, zamiast go usuwać i tworzyć od nowa.

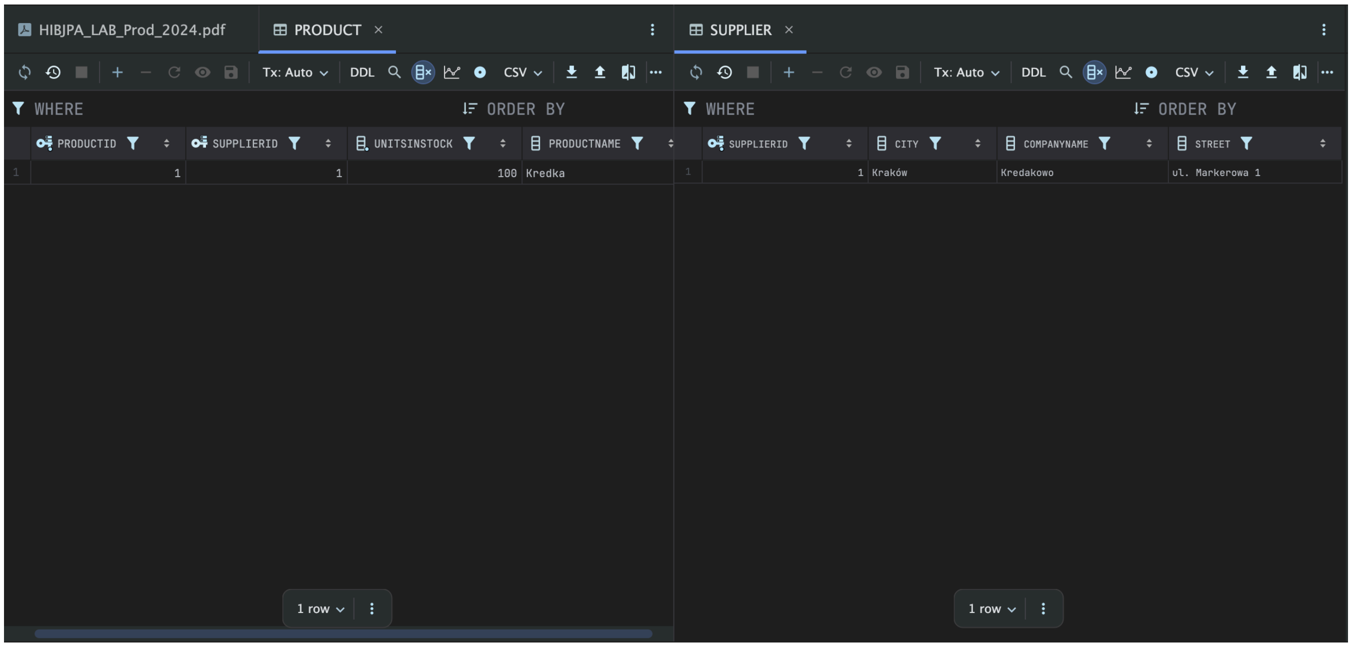
Klasa `main` dodająca nowego dostawcę i przypisująca go do produktu:

```
...

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

        Transaction tx = session.beginTransaction();

        // Create a new supplier
        Supplier supplier = new Supplier("Kredakowo", "ul. Markerowa 1", "Kraków");
        session.persist(supplier);

        // Create a new product supplied by the supplier
        Product product = session.get(Product.class, 1);
        product.setSupplier(supplier);

        session.persist(product);

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

Logi przy uruchomieniu:

```
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:
    select
        p1_0.productID,
        p1_0.productName,
        s1_0.supplierID,
        s1_0.city,
        s1_0.companyName,
        s1_0.street,
        p1_0.unitsInStock
    from
        Product p1_0
    left join
        Supplier s1_0
            on s1_0.supplierID=p1_0.supplierID
    where
        p1_0.productID=?
Hibernate:
    /* insert for
        org.example.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* update
        for org.example.Product */update Product
    set
        productName=?,
        supplierID=?,
        unitsInStock=?
    where
        productID=?

Process finished with exit code 0
```

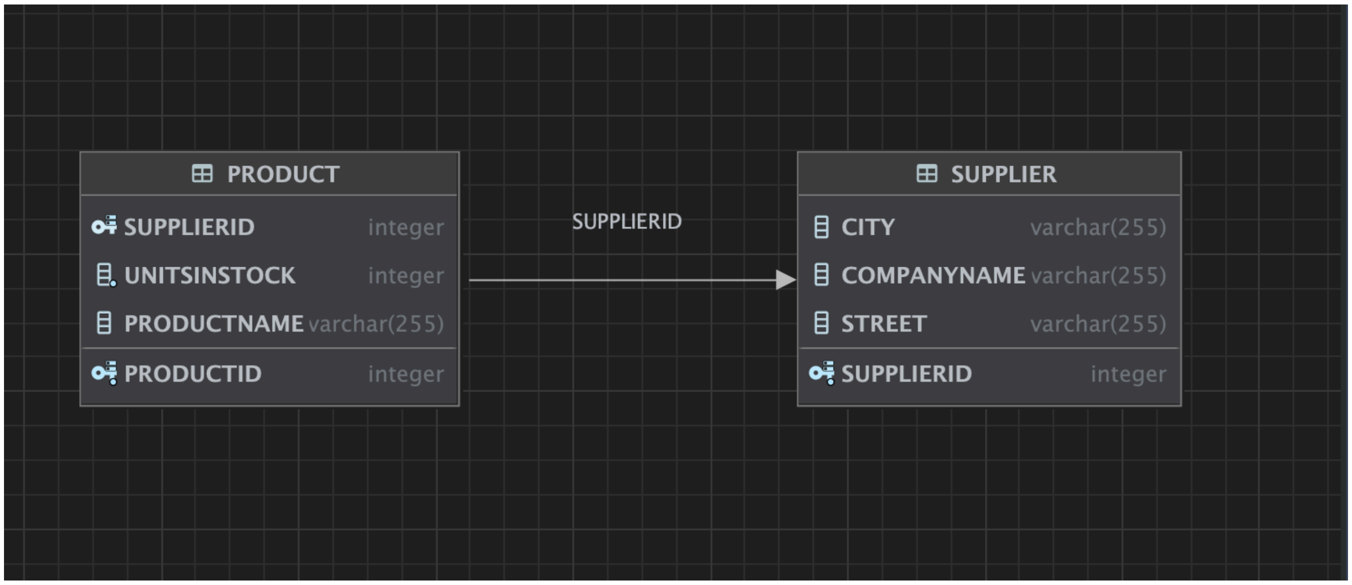
Baza danych po uruchomieniu:



	PRODUCTID	SUPPLIERID	UNITSINSTOCK	PRODUCTNAME
1	1	1	100	Kredka

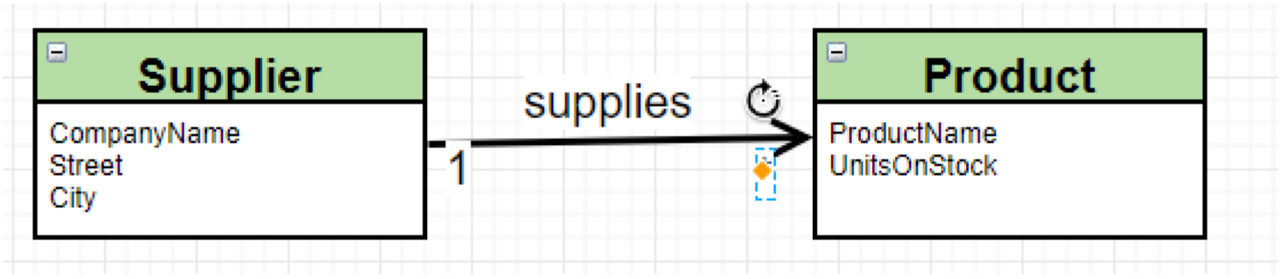
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	Kredakowo	ul. Markerowa 1

Schemat bazy danych:



Zadanie II

Odwróć relację zgodnie z poniższym schematem:



- Zamodeluj powyższe w dwoch wariantach „z” i „bez” tabeli łącznikowej
- Stworz kilka produktow
- Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę

W tym zadaniu tworzymy relację jeden do wielu pomiędzy `Supplier` a `Product`.

Ponieważ schemat bazy danych będzie modyfikowany, wracamy do ustawienia `hbm2ddl.auto` na `create-drop` w pliku `hibernate.cfg.xml`:

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

Wariant bez tabeli łącznikowej

Z klasy `Product` usuwamy dodane w poprzednim zadaniu pola związane z relacją do `Supplier`:

```
...

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


    public Product() {

    }

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

Klasa `Supplier` z relacją jeden do wielu:

```java
...

@Entity
public class Supplier {
    ...

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

    ...

    // Dodawanie produktu
    public void addSuppliedProduct(Product product) {
        suppliedProducts.add(product);
    }
}
```

Zastosowanie adnotacji `@OneToMany` i `@JoinColumn` pozwala na utworzenie relacji jeden do wielu pomiędzy `Supplier` a `Product`, gdzie `SUPPLIER_FK` jest kluczem obcym w tabeli `Product`.

W klasie `Main` tworzymy kilku dostawców i produkty, a następnie przypisujemy produkty do dostawców:

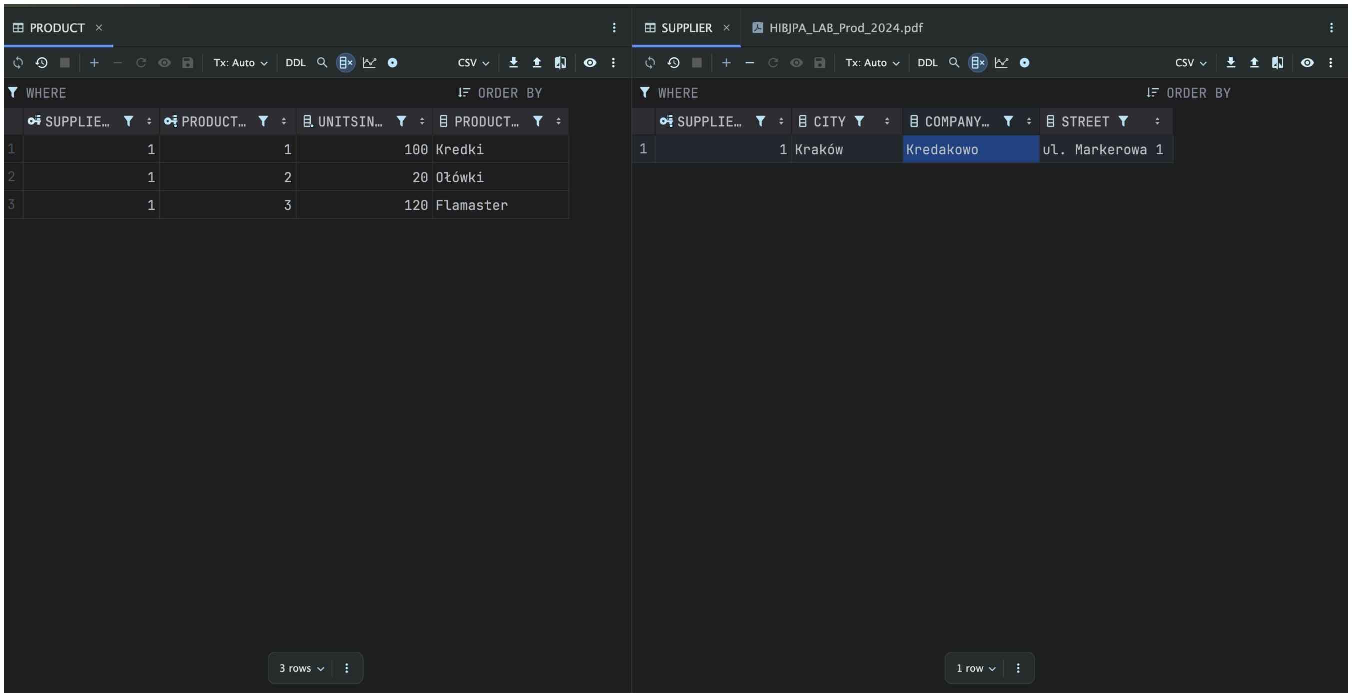
```java
public static void main(String[] args) {
        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        // Create a new supplier
        Supplier supplier = new Supplier("Kredakowo", "ul. Markerowa 1", "Kraków");

        // Create a bunch of products
        Product product1 = new Product("Kredki", 100);
        session.persist(product1);
        supplier.addSuppliedProduct(product1);

        Product product2 = new Product("Ołówki", 20);
        session.persist(product2);
        supplier.addSuppliedProduct(product2);

        Product product3 = new Product("Flamaster", 120);
        session.persist(product3);
        supplier.addSuppliedProduct(product3);

        // Persist the supplier with its products
        session.persist(supplier);

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

Logi przy uruchomieniu:

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

values
    next value for Product_SEQ
Hibernate:

values
    next value for Product_SEQ
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:
    /* insert for
        org.example.Product */insert
    into
        Product (productName, unitsInStock, productID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        org.example.Product */insert
    into
        Product (productName, unitsInStock, productID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        org.example.Product */insert
    into
        Product (productName, unitsInStock, productID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        org.example.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    update
```

```
        Product
    set
        SUPPLIER_FK=?
    where
        productID=?
Hibernate: 
    update
        Product 
    set
        SUPPLIER_FK=? 
    where
        productID=?
Hibernate: 
    update
        Product 
    set
        SUPPLIER_FK=? 
    where
        productID=?

Process finished with exit code 0
```

Baza danych po uruchomieniu:



PRODUCT

	SUPPLIE…	PRODUCT…	UNITSIN…	PRODUCT…
1	1	1	100	Kredki
2	1	2	20	Ołówki
3	1	3	120	Flamaster

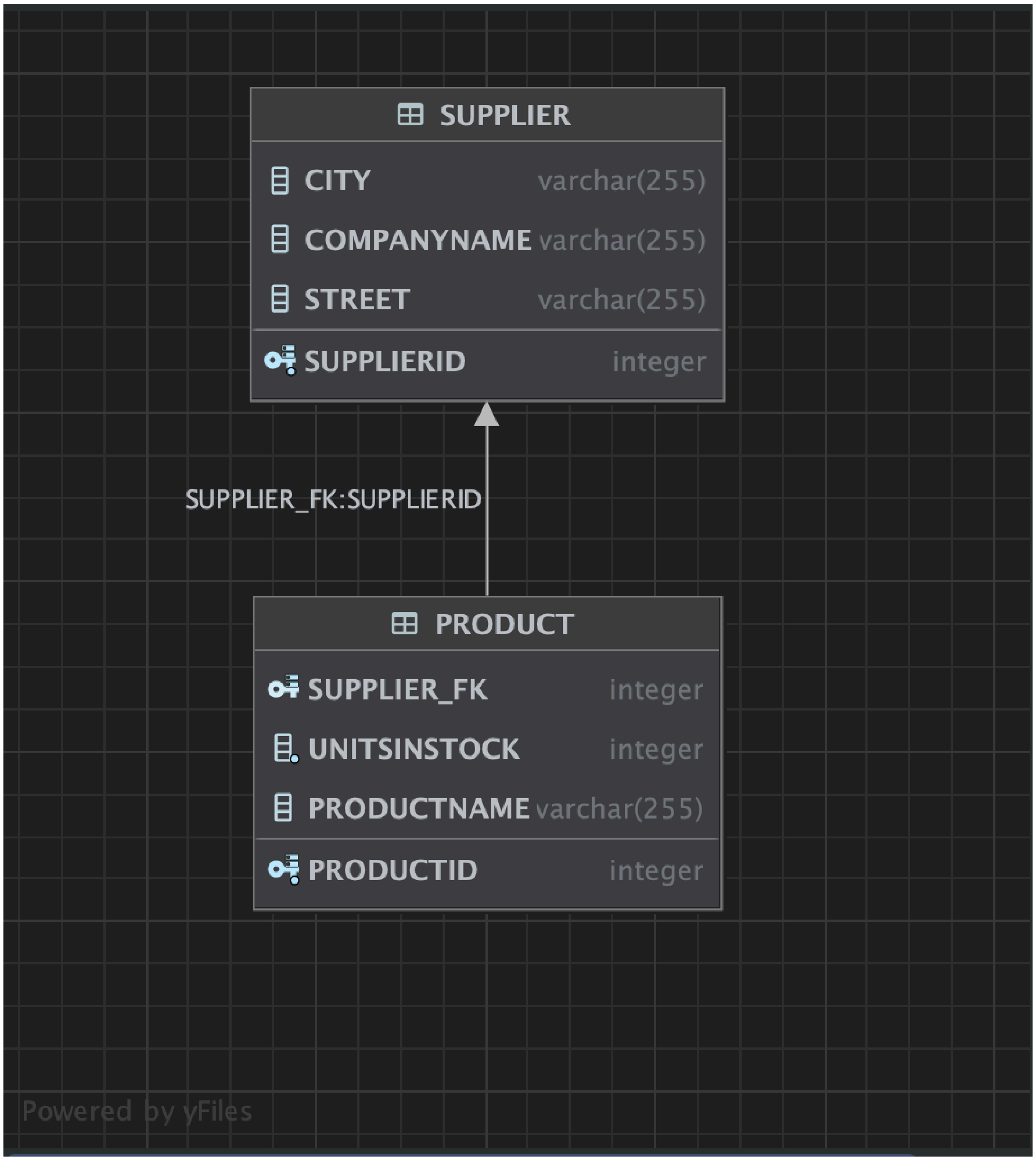
3 rows

SUPPLIER

	SUPPLIE…	CITY	COMPANY…	STREET
1	1	Kraków	Kredakowo	ul. Markerowa 1

1 row

Schemat bazy danych:



Jak widać, produkty są przypisane do dostawcy poprzez klucz obcy `SUPPLIER_FK` w tabeli `Product`.

Wariant z tabelą łącznikową

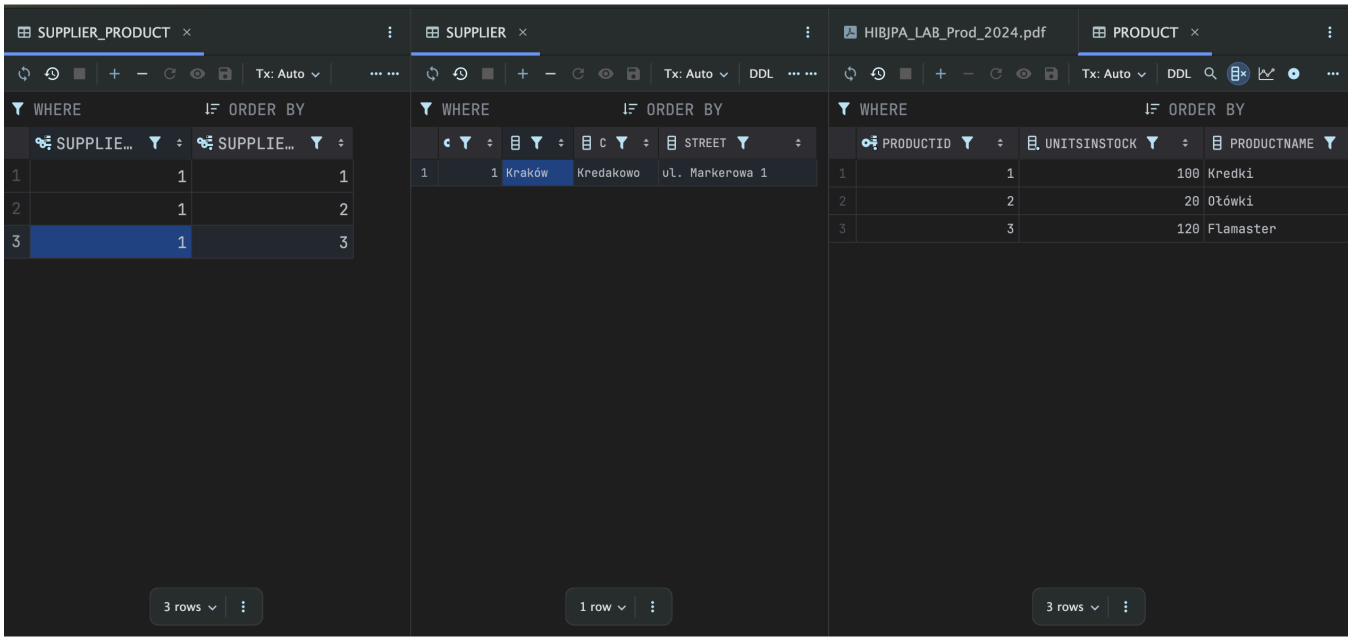
W tym wariancie tworzymy tabelę łącznikową, która będzie przechowywać relację między `Supplier` a `Product`.

Aby zaimplementować ten wariant, wystarczy usunąć adnotację `@JoinColumn` z klasy `Supplier`.

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

    @OneToMany
    private Set<Product> suppliedProducts = new HashSet<>();

    public Supplier() {
    }

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

    // Dodawanie produktu
    public void addSuppliedProduct(Product product) {
        suppliedProducts.add(product);
    }
}
```

W ten sposób Hibernate sam utworzy tabelę łącznikową, o nazwie `Supplier_Product`, która będzie przechowywać relację między `Supplier` a `Product`.

Baza danych po uruchomieniu:



SUPPLIER_PRODUCT

	SUPPLIE…	SUPPLIE…
1	1	1
2	1	2
3	1	3

SUPPLIER

	c		C	STREET
1	1	Kraków	Kredakowo	ul. Markerowa 1

PRODUCT

	PRODUCTID	UNITSINSTOCK	PRODUCTNAME
1	1	100	Kredki
2	2	20	Ołówki
3	3	120	Flamaster

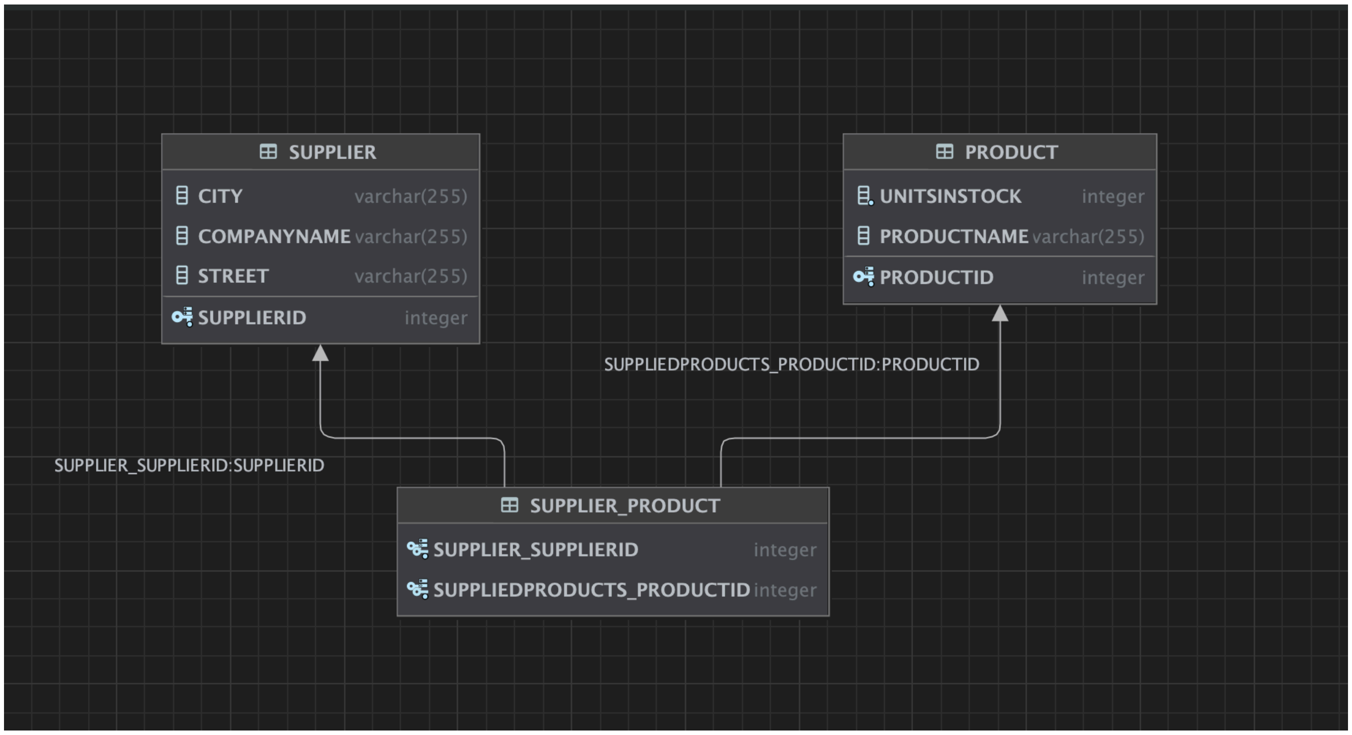
Logi przy uruchomieniu:

```
Hibernate: 
    drop sequence Product_SEQ restrict
Hibernate: 
    drop sequence Supplier_SEQ restrict
Hibernate: 
    create sequence Product_SEQ start with 1 increment by 50
Hibernate: 
    create sequence Supplier_SEQ start with 1 increment by 50
Hibernate: 
    create table Product (
        productID integer not null,
        unitsInStock integer not null,
        productName varchar(255),
        primary key (productID)
    )
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,
        suppliedProducts_productID integer not null unique,
        primary key (Supplier_supplierID, suppliedProducts_productID)
    )
Hibernate: 
    alter table Supplier_Product 
       add constraint FK9lle0r59trgfnn3u0j8xds67a 
       foreign key (suppliedProducts_productID) 
       references Product
Hibernate: 
    alter table Supplier_Product 
       add constraint FKjskj7cplt17tebkn930wt8ke6 
       foreign key (Supplier_supplierID) 
       references Supplier
Hibernate: 

values
    next value for Product_SEQ
Hibernate: 

values
    next value for Product_SEQ
```

```
Hibernate:

values
    next value for Supplier_SEQ
Hibernate:
    /* insert for
        org.example.Product */insert
    into
        Product (productName, unitsInStock, productID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        org.example.Product */insert
    into
        Product (productName, unitsInStock, productID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        org.example.Product */insert
    into
        Product (productName, unitsInStock, productID)
    values
        (?, ?, ?)
Hibernate:
    /* insert for
        org.example.Supplier */insert
    into
        Supplier (city, companyName, street, supplierID)
    values
        (?, ?, ?, ?)
Hibernate:
    /* insert for
        org.example.Supplier.suppliedProducts */insert
    into
        Supplier_Product (Supplier_supplierID, suppliedProducts_productID)
    values
        (?, ?)
Hibernate:
    /* insert for
        org.example.Supplier.suppliedProducts */insert
    into
        Supplier_Product (Supplier_supplierID, suppliedProducts_productID)
    values
        (?, ?)
Hibernate:
    /* insert for
```

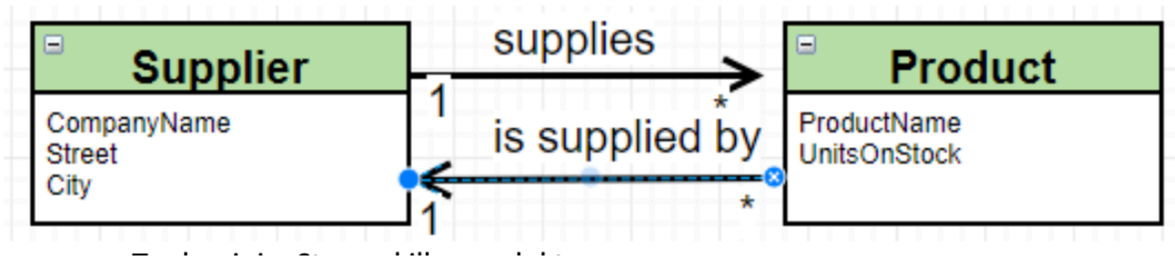
```
        org.example.Supplier.suppliedProducts */insert
    into
        Supplier_Product (Supplier_supplierID, suppliedProducts_productID)
    values
        (?, ?)

Process finished with exit code 0
```

Schemat bazy danych:



Zadanie III

Zamodeluj relację dwustronną jak poniżej:



- Tradycyjnie: Stwórz kilka produktów
- Dodaj je do produktow dostarczanych przez nowo stworzonego dostawcę

(pamiętaj o
poprawnej obsłudze dwustronności relacji)

W tym podpunkcie stworzymy relację bez tabeli łącznikowej.

Zmieniamy klasę `Supplier`:

```
...

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

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

    public Supplier() {
    }

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

    // Dodawanie produktu
    public void addSuppliedProduct(Product product) {
        suppliedProducts.add(product);
    }
}
```

Zmieniamy klasę `Product`:

```java
...

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

    @ManyToOne
    private Supplier supplier;

    public Product() {

    }

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

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

Klasa `Main`:

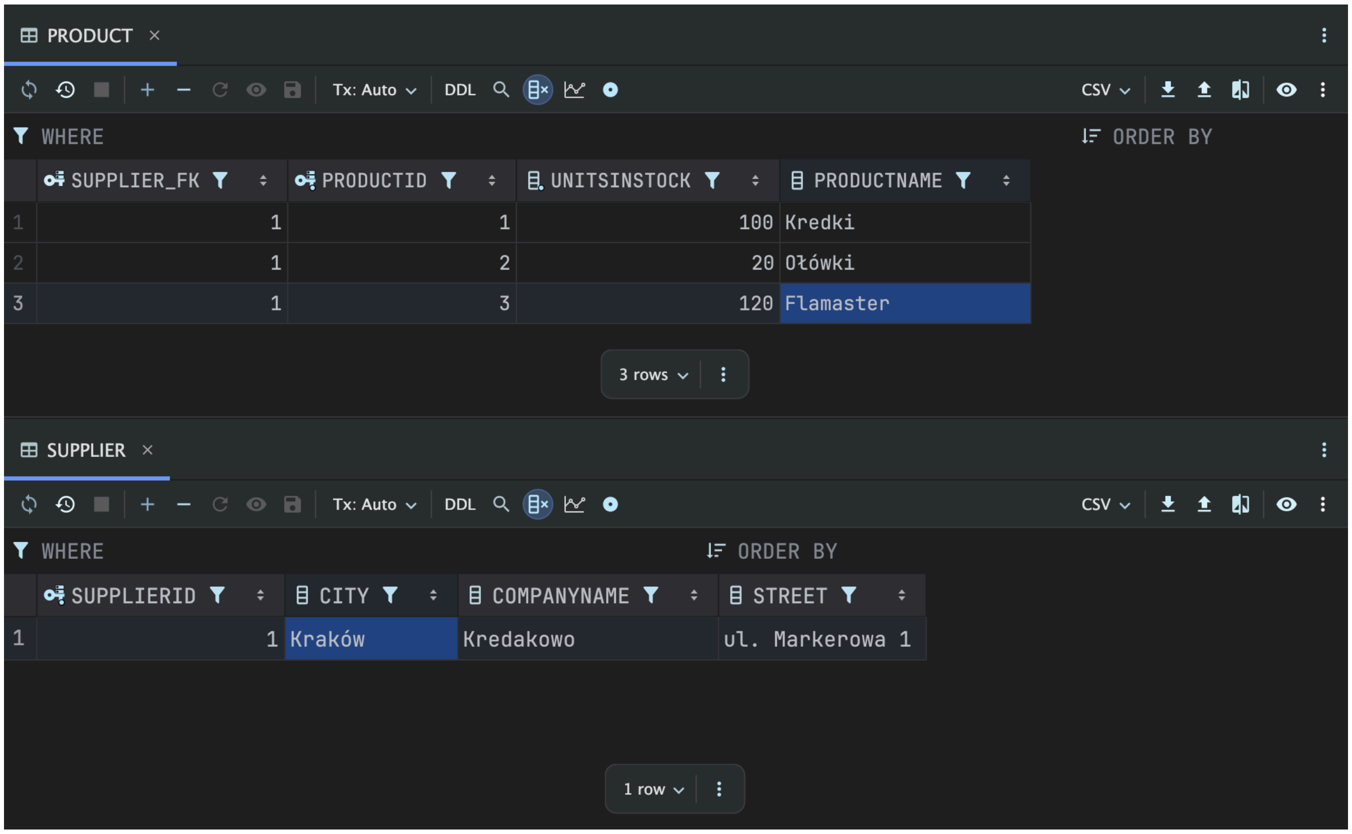
```java
    public static void main(String[] args) {
        sessionFactory = getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();

        // Create a new supplier
        Supplier supplier = new Supplier("Kredakowo", "ul. Markerowa 1", "Kraków");

        // Create a bunch of products
        Product product1 = new Product("Kredki", 100);
        product1.setSupplier(supplier); // Set the supplier for the product
        session.persist(product1);
        supplier.addSuppliedProduct(product1);

        Product product2 = new Product("Ołówki", 20);
        product2.setSupplier(supplier); // Set the supplier for the product
        session.persist(product2);
        supplier.addSuppliedProduct(product2);

        Product product3 = new Product("Flamaster", 120);
        product3.setSupplier(supplier); // Set the supplier for the product
        session.persist(product3);
        supplier.addSuppliedProduct(product3);

        // Persist the supplier with its products
        session.persist(supplier);

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

Logi podczas uruchomienia:

```
values
    next value for Product_SEQ
Hibernate:

values
    next value for Product_SEQ
Hibernate:

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

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

Process finished with exit code 0
```

Wynik po uruchomieniu:



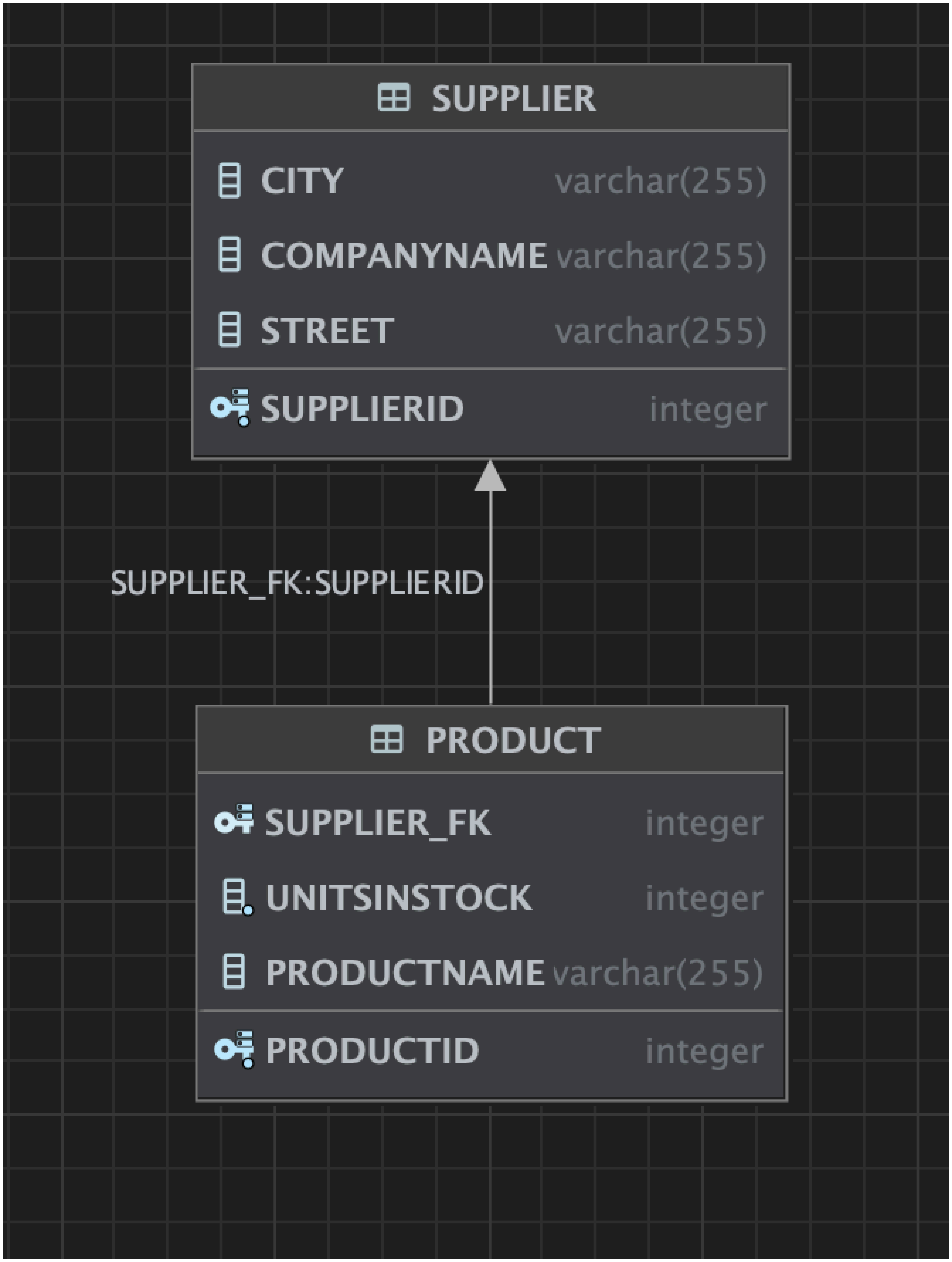
	SUPPLIER_FK	PRODUCTID	UNITSINSTOCK	PRODUCTNAME
1	1	1	100	Kredki
2	1	2	20	Ołówki
3	1	3	120	Flamaster

3 rows

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	Kredakowo	ul. Markerowa 1

1 row

Schemat bazy danych:


SUPPLIER
CITY varchar(255)
COMPANYNAME varchar(255)
STREET varchar(255)
SUPPLIERID integer
SUPPLIER_FK:SUPPLIERID
PRODUCT
SUPPLIER_FK integer
UNITSINSTOCK integer
PRODUCTNAME varchar(255)
PRODUCTID integer

Zadanie IV

Dodaj klase `Category` z property:

```
int CategoryID;
String name
```

oraz listą produktów:

```
List<Product> products;
```

- Zmodyfikuj produkty dodając wskazanie na kategorie do której należy.
- Stworz kilka produktow i kilka kategorii
- Dodaj kilka produktów do wybranej kategorii
- Wydobądź produkty z wybranej kategorii oraz kategorię do której należy wybrany produkt

Zaczynamy od stworzenia klasy `Category`:

```
...

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

    private int categoryID;
    private String name;

    @OneToMany
    @JoinColumn(name = "CATEGORY_FK")
    private Set<Product> products = new HashSet<>();

    public Category() {

    }

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

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

    public Set<Product> getProducts() {
        return this.products;
    }

    public String getName() {
        return this.name;
    }
}
```

Zmodyfikowana klasa `Product`:

```
...

@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 supplier;

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

    public Product() {

    }

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

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

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

    public Category getCategory() {
        return this.category;
    }

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

Klasa `Main`:

```java
...

public class Main {
    ...

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

        // Create a new supplier
        Supplier supplier = new Supplier("Kredakowo", "ul. Markerowa 1", "Kraków");


        // Create new categories
        Category biurowe = new Category("Biurowe");
        Category szkolne = new Category("Szkolne");

        // Create a bunch of products
        Product product1 = new Product("Długopis", 100);
        product1.setSupplier(supplier);
        biurowe.addProduct(product1);
        session.persist(product1);
        supplier.addSuppliedProduct(product1);

        Product product2 = new Product("Zszywacz", 20);
        product2.setSupplier(supplier);
        biurowe.addProduct(product2);
        session.persist(product2);
        supplier.addSuppliedProduct(product2);

        Product product3 = new Product("Zeszyt", 120);
        product3.setSupplier(supplier);
        szkolne.addProduct(product3);
        session.persist(product3);
        supplier.addSuppliedProduct(product3);

        // Persist supplier and categories
        session.persist(supplier);

        session.persist(biurowe);
        session.persist(szkolne);

        tx.commit();
        // Fetch products from Category
        var query = session.createQuery("from Category c  where c.name = 'Biurowe'");
        Category rCat = (Category) query.getSingleResult();
        for (Product p : rCat.getProducts()) {
            System.out.println("Produkt w kategorii " + rCat.getName() + ": " + p.getProductName());
        }

        query = session.createQuery("from Product p where p.productName = 'Zeszyt'");
        Product rProd = (Product) query.getSingleResult();
        System.out.println("Produkt: " + rProd.getProductName() + ", kategoria: " + rProd.getCategory().getN

        session.close();
    }

    private static SessionFactory
}
```

Logi przy uruchomieniu:

```
Hibernate:
    create table Category (
        categoryID integer not null,
        name varchar(255),
        primary key (categoryID)
    )
Hibernate:
    create table Product (
        CATEGORY_FK integer,
        SUPPLIER_FK integer,
        productID integer not null,
        unitsInStock integer not null,
        productName varchar(255),
        primary key (productID)
    )
...
```

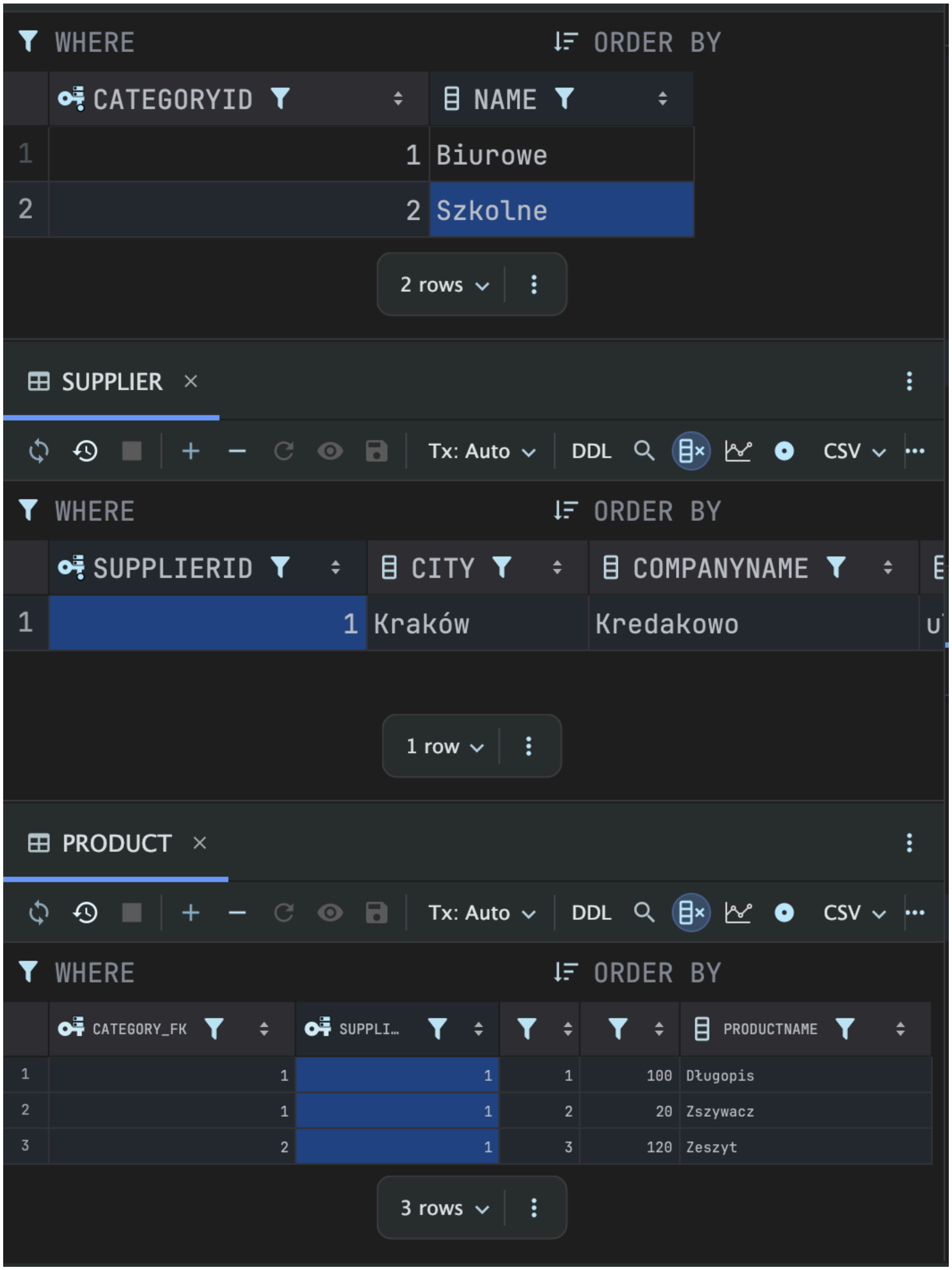
Produkty w kategorii Biurowe wypisane podczas uruchomienia:

```
Produkt w kategorii Biurowe: Zszywacz
Produkt w kategorii Biurowe: Długopis
```

Kategoria wybranego produktu:

```
Produkt: Zeszyt, kategoria: Szkolne
```

Baza danych po uruchomieniu:



WHERE ORDER BY

	CATEGORYID	NAME
1	1	Biurowe
2	2	Szkolne

2 rows

SUPPLIER

Tx: Auto DDL CSV

WHERE ORDER BY

	SUPPLIERID	CITY	COMPANYNAME
1	1	Kraków	Kredakowo

1 row

PRODUCT

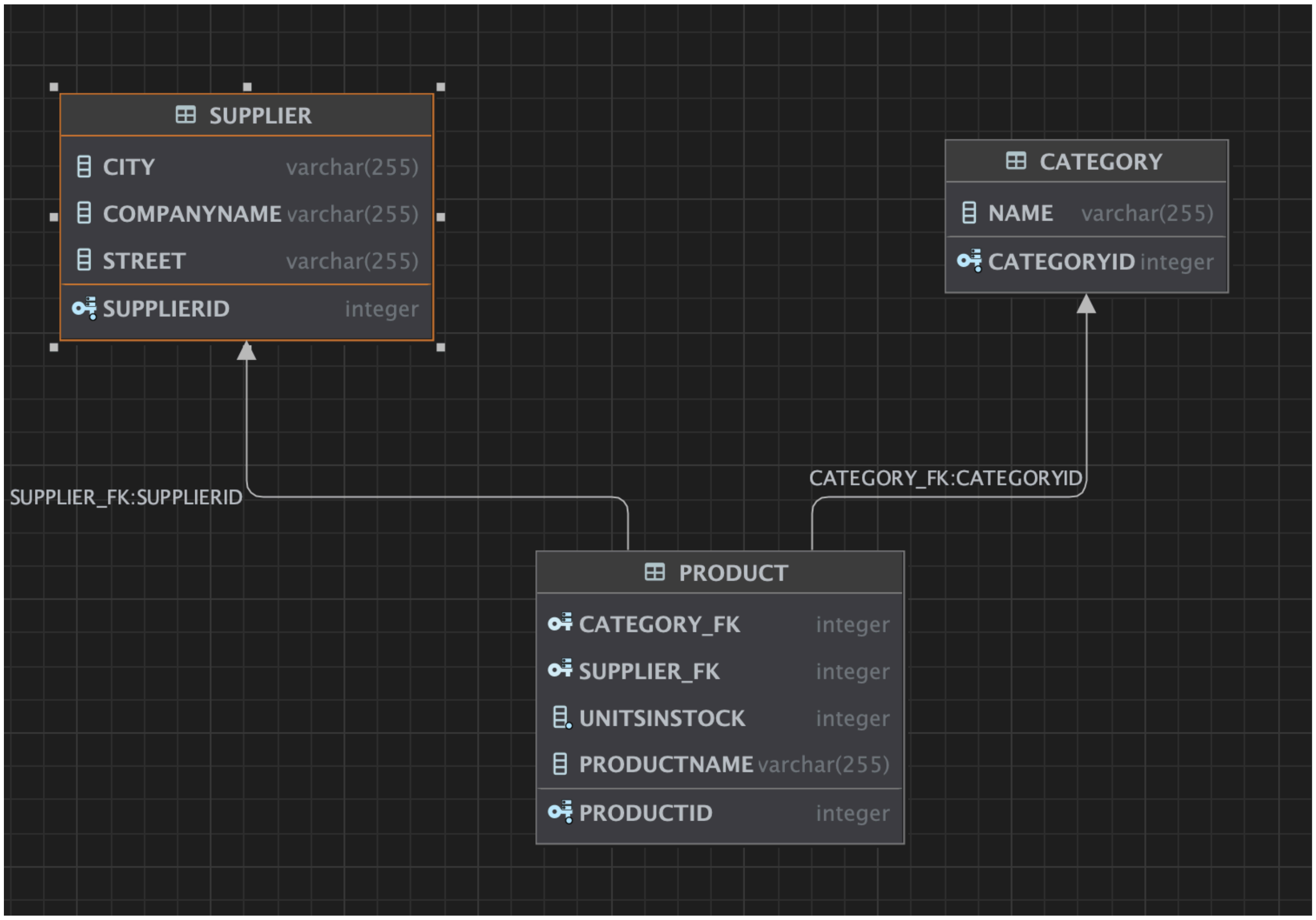
Tx: Auto DDL CSV

WHERE ORDER BY

	CATEGORY_FK	SUPPLI…			PRODUCTNAME
1	1	1	1	100	Długopis
2	1	1	2	20	Zszywacz
3	2	1	3	120	Zeszyt

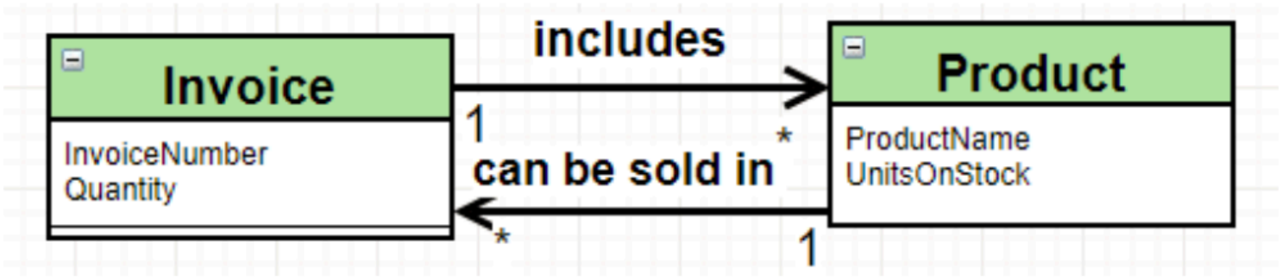
3 rows

Schemat bazy po uruchomieniu:



Zadanie V

Zamodeluj relację wiele do wielu jak poniżej:



- Stwórz kilka produktów i "sprzedaj" je w kilku tranksakcjach
- Pokaż produkty sprzedane w ramach wybranej transakcji
- Pokaz faktury w ramach których był sprzedany wybrany produkt

Stworzona klasa `Invoice`:

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

    private int number;
    private int quantity;

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

    public Invoice() {

    }

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

    public void setNumber(int number) {
        this.number = number;
    }

    public int getNumber() {
        return number;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Invoice ID: ").append(invoiceID).append("\n");
        sb.append("Number: ").append(number).append("\n");
        sb.append("Quantity: ").append(quantity).append("\n");
        sb.append("Products:\n");
        for (Product product : products) {
            sb.append("- ").append(product.getProductName()).append("\n");
        }
        return sb.toString();
    }
}
```

Zmieniona klasa `Product`:

```java
package org.example;

import jakarta.persistence.*;

import java.util.HashSet;
import java.util.Set;

@Entity
public class Product {
    ...

    @ManyToMany
    private Set<Invoice> invoices = new HashSet<>();

    ...

    public String getProductName() {
        return this.productName;
    }

    public void addToInvoice(Invoice invoice, int quantity) {
        invoices.add(invoice);
    }

    public Set<Invoice> getInvoices() {
        return this.invoices;
    }
}
```

Zmieniona klasa `Main`:

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

        // Create a new supplier
        Supplier supplier = new Supplier("Kredakowo", "ul. Markerowa 1", "Kraków");


        // Create new categories
        Category biurowe = new Category("Biurowe");
        Category szkolne = new Category("Szkolne");

        // Create a bunch of products
        Product product1 = new Product("Długopis", 100);
        product1.setSupplier(supplier);
        biurowe.addProduct(product1);
        session.persist(product1);
        supplier.addSuppliedProduct(product1);

        Product product2 = new Product("Zszywacz", 20);
        product2.setSupplier(supplier);
        biurowe.addProduct(product2);
        session.persist(product2);
        supplier.addSuppliedProduct(product2);

        Product product3 = new Product("Zeszyt", 120);
        product3.setSupplier(supplier);
        szkolne.addProduct(product3);
        session.persist(product3);
        supplier.addSuppliedProduct(product3);

        // Persist supplier and categories
        session.persist(supplier);

        session.persist(biurowe);
        session.persist(szkolne);

        // Create new invoices
        Invoice invoice = new Invoice();
        invoice.setNumber(1);

        invoice.addProduct(product1, 2);
        invoice.addProduct(product3, 30);


        Invoice invoice2 = new Invoice();
        invoice2.setNumber(2);

        invoice2.addProduct(product1, 2);
        invoice2.addProduct(product2, 1);
        invoice2.addProduct(product3, 5);

        session.persist(invoice);
        session.persist(invoice2);

        tx.commit();

        var query = session.createQuery("from Invoice i where i.number = 1");
        Invoice rInvoice = (Invoice) query.getSingleResult();
        System.out.println(rInvoice);

        query = session.createQuery("from Product p where p.productName = 'Zeszyt'");
        Product rProd = (Product) query.getSingleResult();
        System.out.println("Produkt: " + rProd.getProductName() + " znajduje się na fakturach:");
        for(Invoice inv: rProd.getInvoices()) {
            System.out.println("- " + inv.getNumber());
        }
    }
```

Wyniki w konsoli:

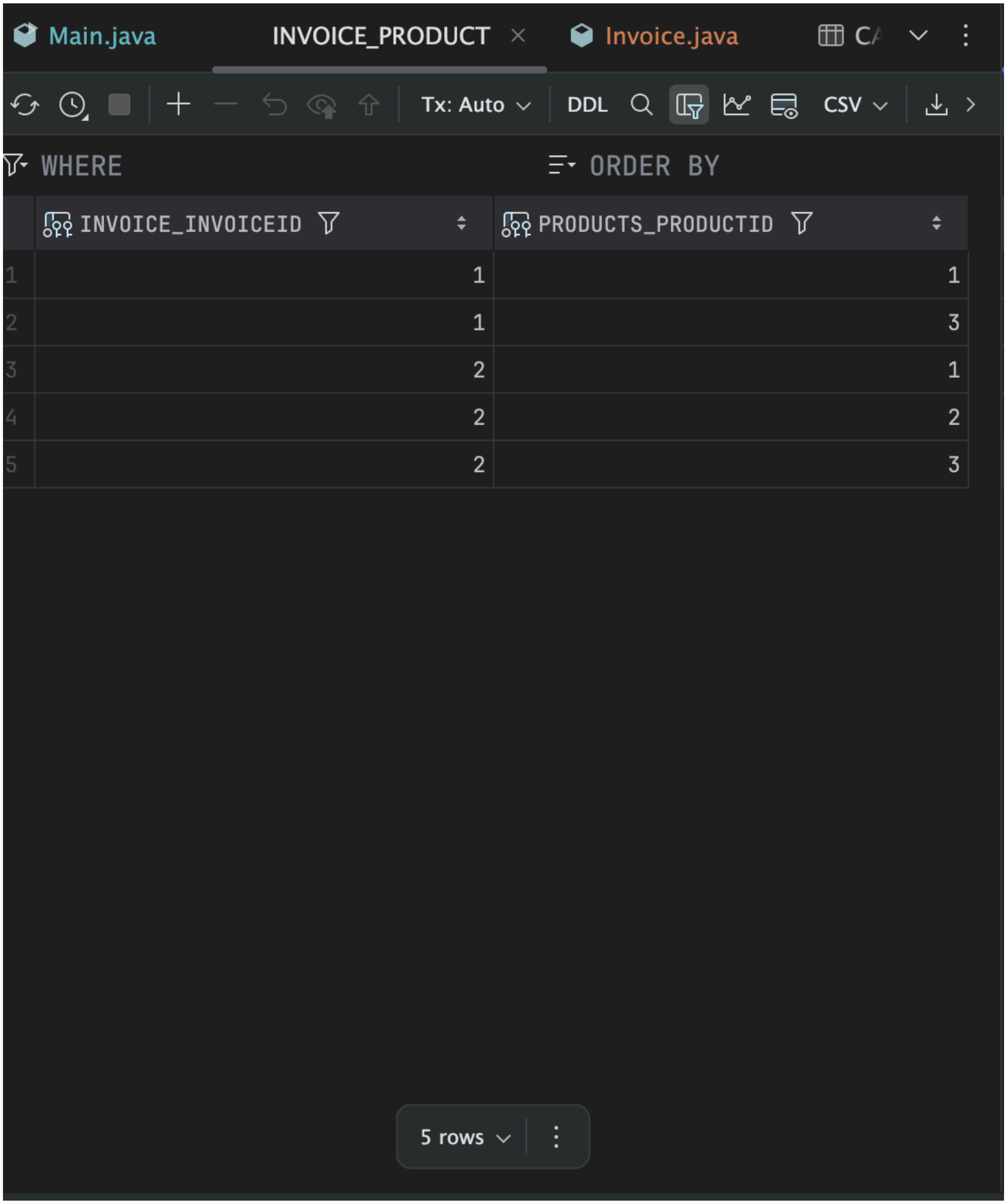
```
Invoice ID: 1
Number: 1
Quantity: 32
Products:
- Zeszyt
- Długopis
```

```
Produkt: Zeszyt znajduje się na fakturach:
- 1
- 2
```

Logi:

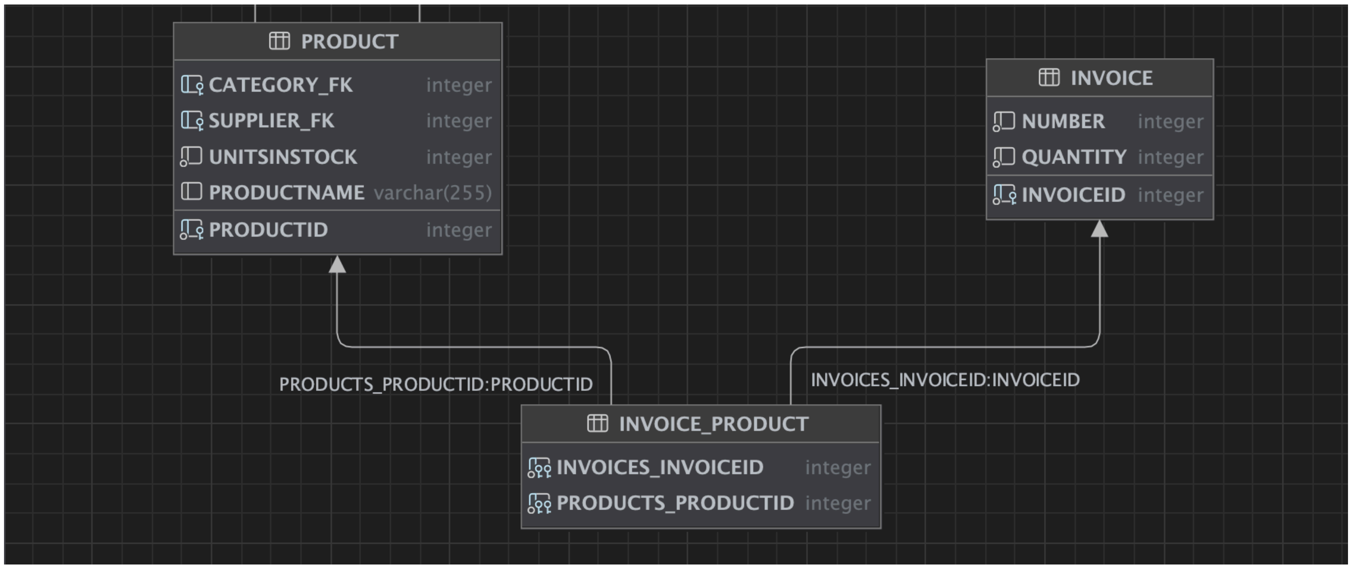
```
Hibernate:
    create table Invoice (
        invoiceID integer not null,
        number integer not null,
        quantity integer not null,
        primary key (invoiceID)
    )
Hibernate:
    create table Invoice_Product (
        Invoice_invoiceID integer not null,
        products_productID integer not null,
        primary key (Invoice_invoiceID, products_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)
    )
```

Baza danych po uruchomieniu:



	INVOICE_INVOICEID	PRODUCTS_PRODUCTID
1	1	1
2	1	3
3	2	1
4	2	2
5	2	3

Schemat bazy danych:



Zadanie VI

JPA:

- Stwórz nowego maina w którym zrobisz to samo co w poprzednim ale z wykorzystaniem JPA

Na początku tworzymy nowy plik `persistence.xml` w katalogu `src/main/resources/META-INF/` :

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

Zmiany w klasie `Main` dla JPA:

Usuwamy wszystkie importy Hibernate i dodajemy odpowiednie dla JPA:

```
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Persistence;
```

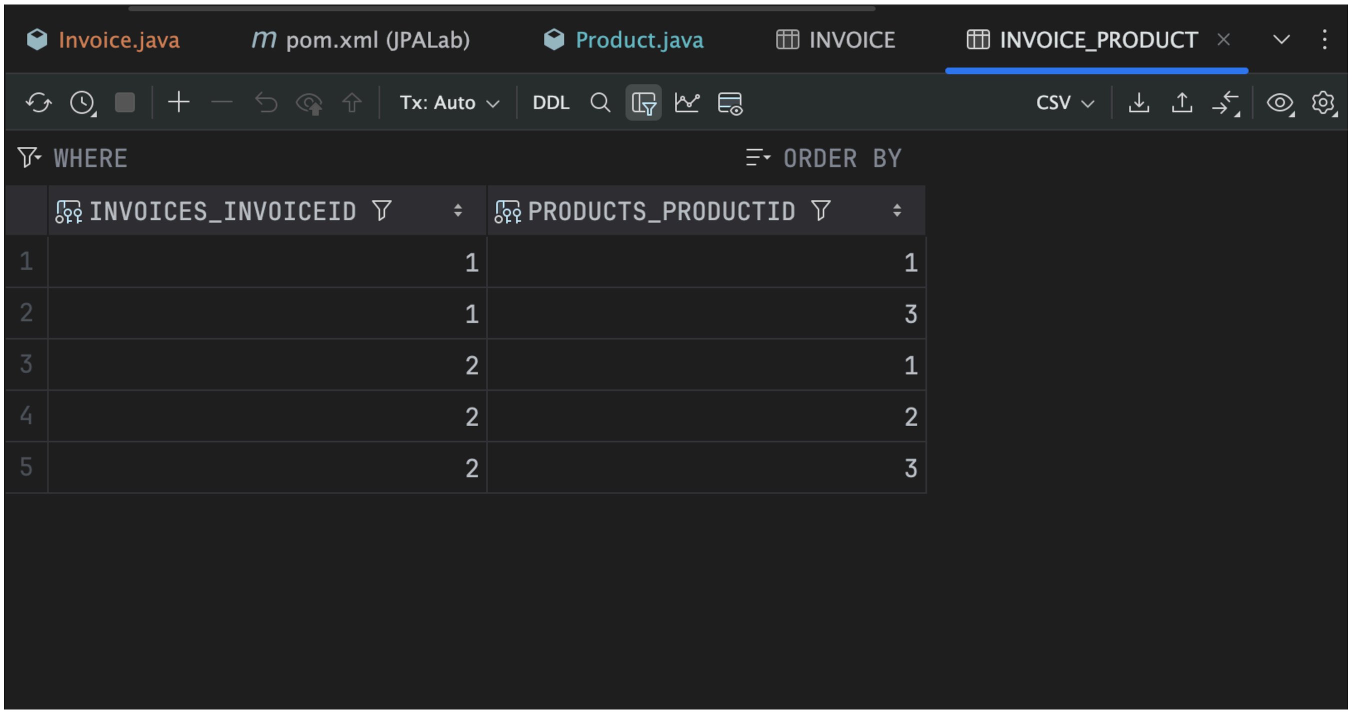
Następnie zmieniamy klasę `Main` tak, aby używała JPA zamiast Hibernate:

```java
public static void main(String[] args) {
        // Create an EntityManagerFactory and EntityManager for JPA
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MyLabDatabase");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();

        // tx is now etx
        etx.begin();

        // Create a new supplier
        Supplier supplier = new Supplier("Kredakowo", "ul. Markerowa 1", "Kraków");


        // Create new categories
        Category biurowe = new Category("Biurowe");
        Category szkolne = new Category("Szkolne");

        // Create a bunch of products
        Product product1 = new Product("Długopis", 100);
        product1.setSupplier(supplier);
        biurowe.addProduct(product1);
        em.persist(product1);
        supplier.addSuppliedProduct(product1);

        Product product2 = new Product("Zszywacz", 20);
        product2.setSupplier(supplier);
        biurowe.addProduct(product2);
        em.persist(product2);
        supplier.addSuppliedProduct(product2);

        Product product3 = new Product("Zeszyt", 120);
        product3.setSupplier(supplier);
        szkolne.addProduct(product3);
        em.persist(product3);
        supplier.addSuppliedProduct(product3);

        // Persist supplier and categories
        em.persist(supplier);

        em.persist(biurowe);
        em.persist(szkolne);

        // Create new invoices
        Invoice invoice = new Invoice();
        invoice.setNumber(1);

        invoice.addProduct(product1, 2);
        invoice.addProduct(product3, 30);


        Invoice invoice2 = new Invoice();
        invoice2.setNumber(2);

        invoice2.addProduct(product1, 2);
        invoice2.addProduct(product2, 1);
        invoice2.addProduct(product3, 5);

        em.persist(invoice);
        em.persist(invoice2);

        etx.commit();

        var query = em.createQuery("from Invoice i where i.number = 1");
        Invoice rInvoice = (Invoice) query.getSingleResult();
        System.out.println(rInvoice);

        query = em.createQuery("from Product p where p.productName = 'Zeszyt'");
        Product rProd = (Product) query.getSingleResult();
        System.out.println("Produkt: " + rProd.getProductName() + " znajduje się na fakturach:");
        for(Invoice inv: rProd.getInvoices()) {
            System.out.println("- " + inv.getNumber());
        }
    }
```

Wyniki:

```
Invoice ID: 1
Number: 1
Quantity: 32
Products:
- Zeszyt
- Długopis
```

```
Produkt: Zeszyt znajduje się na fakturach:
- 1
- 2
```



	INVOICES_INVOICEID	PRODUCTS_PRODUCTID
1	1	1
2	1	3
3	2	1
4	2	2
5	2	3

Zadanie VII - Embedded class

- Dodaj do medelu klase adres. "Wbuduj ją do tabeli Dostawców"
- Zmodyfikuj model w taki sposób, że dane adresowe znajdują się w klasie dostawców. Zmapuj to do dwóch tabel.

Wersja z wbudowaną klasą

Zaczynamy od stworzenia klasy `Address` :

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

    public Address() {

    }

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

Następnie zmieniamy klasę `Supplier`, aby używała klasy `Address`:

```
...
@Entity
public class Supplier {
    ...

    @Embedded
    private Address address;

    public Supplier() {
    }

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

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

}
```

Klasa `Main`:

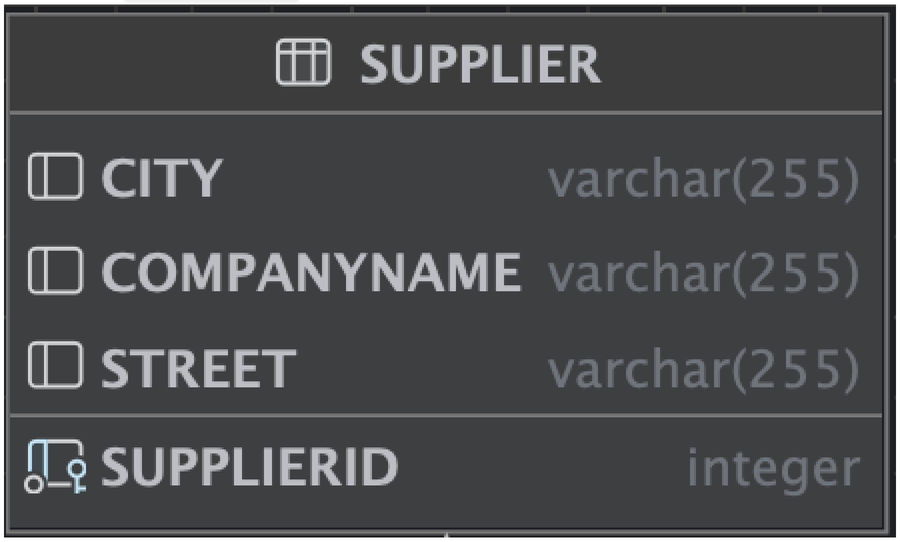
```
public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MyLabDatabase");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();

        etx.begin();

        Address address = new Address("Barszczowa", "Burakowo");
        Supplier supplier1 = new Supplier("Buraki Sp. z o.o.", address);

        Supplier supplier2 = new Supplier("Ziemniaki Sp. z o.o.", "Ziemniaczana", "Ziemniakowo");

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

        var query = em.createQuery("SELECT s FROM Supplier s", Supplier.class);
        List<Supplier> suppliers = query.getResultList();

        for (Supplier supplier : suppliers) {
            System.out.println(supplier);
        }

        etx.commit();
    }
```

Wyniki w konsoli:

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

```
Supplier ID: 1
Company Name: Buraki Sp. z o.o.
Address: Barszczowa, Burakowo
Supplied Products:

Supplier ID: 2
Company Name: Ziemniaki Sp. z o.o.
Address: Ziemniaczana, Ziemniakowo
Supplied Products:
```

Tabela `Supplier` po uruchomieniu:



SUPPLIER	
CITY	varchar(255)
COMPANYNAME	varchar(255)
STREET	varchar(255)
SUPPLIERID	integer

Wersja z dwoma tabelami

Klasa `Supplier`:

```
...

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

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

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

    ...

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Supplier ID: ").append(supplierID).append("\n");
        sb.append("Company Name: ").append(companyName).append("\n");
        sb.append("Address: ").append(street + ", " + city).append("\n");
        sb.append("Supplied Products:\n");
        for (Product product : suppliedProducts) {
            sb.append("- ").append(product.getProductName()).append("\n");
        }
        return sb.toString();
    }
}
```

Klasa `Main`:

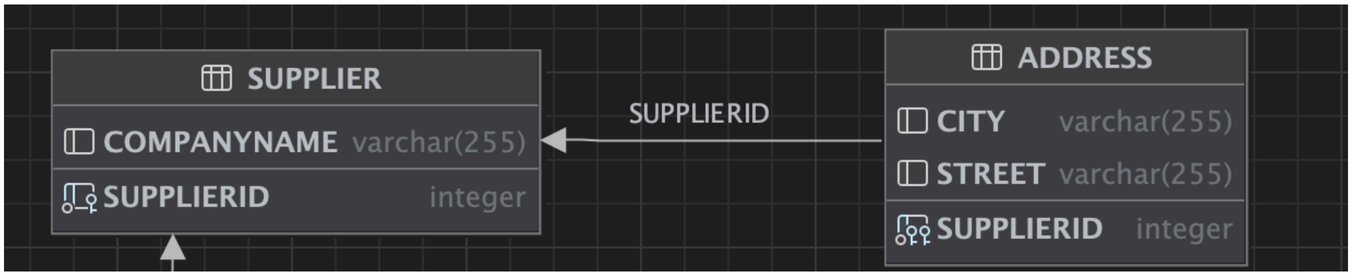
```java
public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("MyLabDatabase");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();

        etx.begin();

        Supplier supplier1 = new Supplier("Buraki Sp. z o.o.", "Barszczowa", "Burakowo");

        Supplier supplier2 = new Supplier("Ziemniaki Sp. z o.o.", "Ziemniaczana", "Ziemniakowo");

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

        var query = em.createQuery("SELECT s FROM Supplier s", Supplier.class);
        List<Supplier> suppliers = query.getResultList();

        for (Supplier supplier : suppliers) {
            System.out.println(supplier);
        }

        etx.commit();
    }
}
```

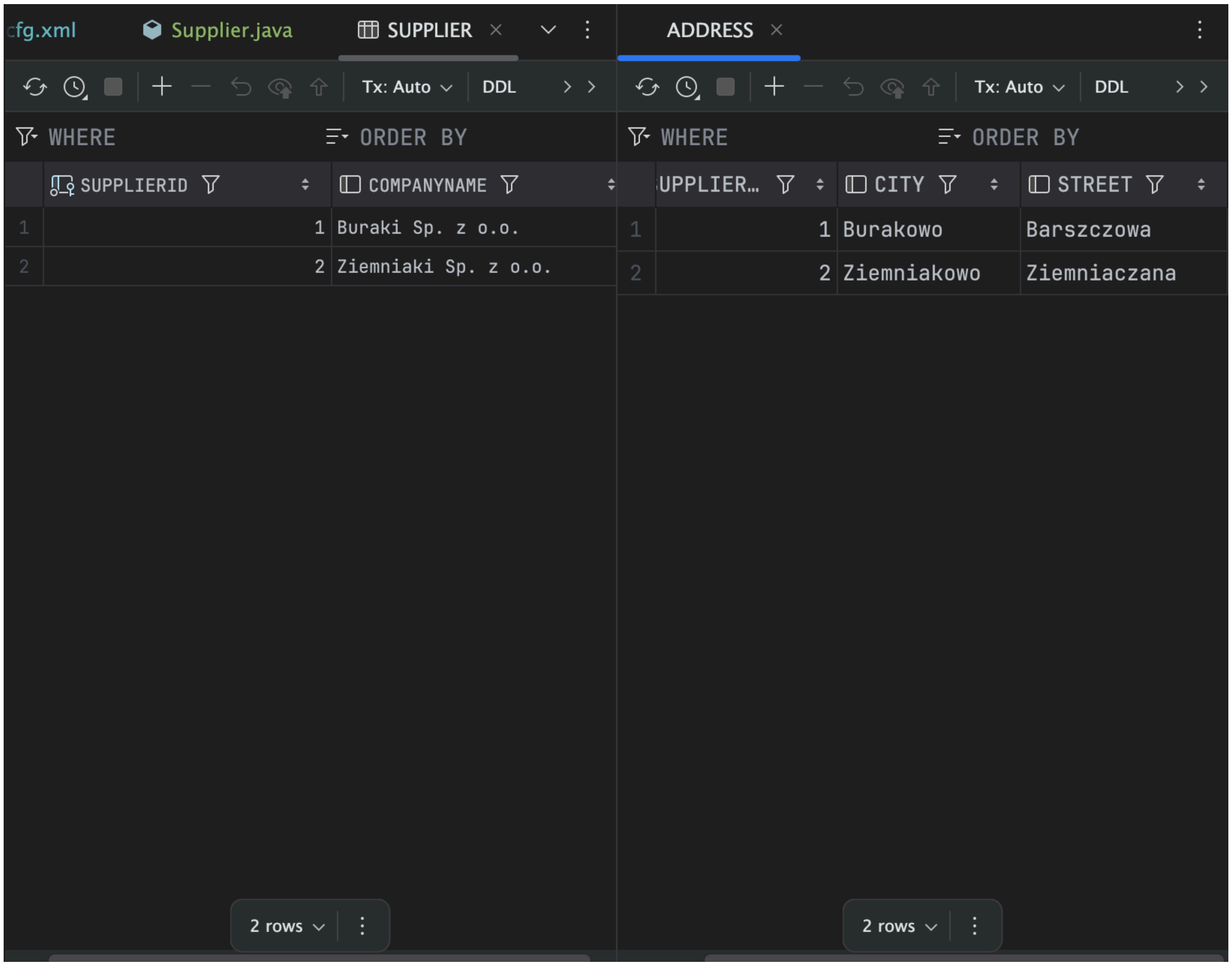
Wyniki w kosoli:

```
Hibernate:
    create table Supplier (
        supplierID integer not null,
        companyName varchar(255),
        primary key (supplierID)
    )
Hibernate:
    alter table Address
       add constraint FKsg53al8nvbanq59s3pd6axyit
       foreign key (supplierID)
       references Supplier
```

```
Supplier ID: 1
Company Name: Buraki Sp. z o.o.
Address: Barszczowa, Burakowo
Supplied Products:

Supplier ID: 2
Company Name: Ziemniaki Sp. z o.o.
Address: Ziemniaczana, Ziemniakowo
Supplied Products:
```

Tabele po uruchomieniu:



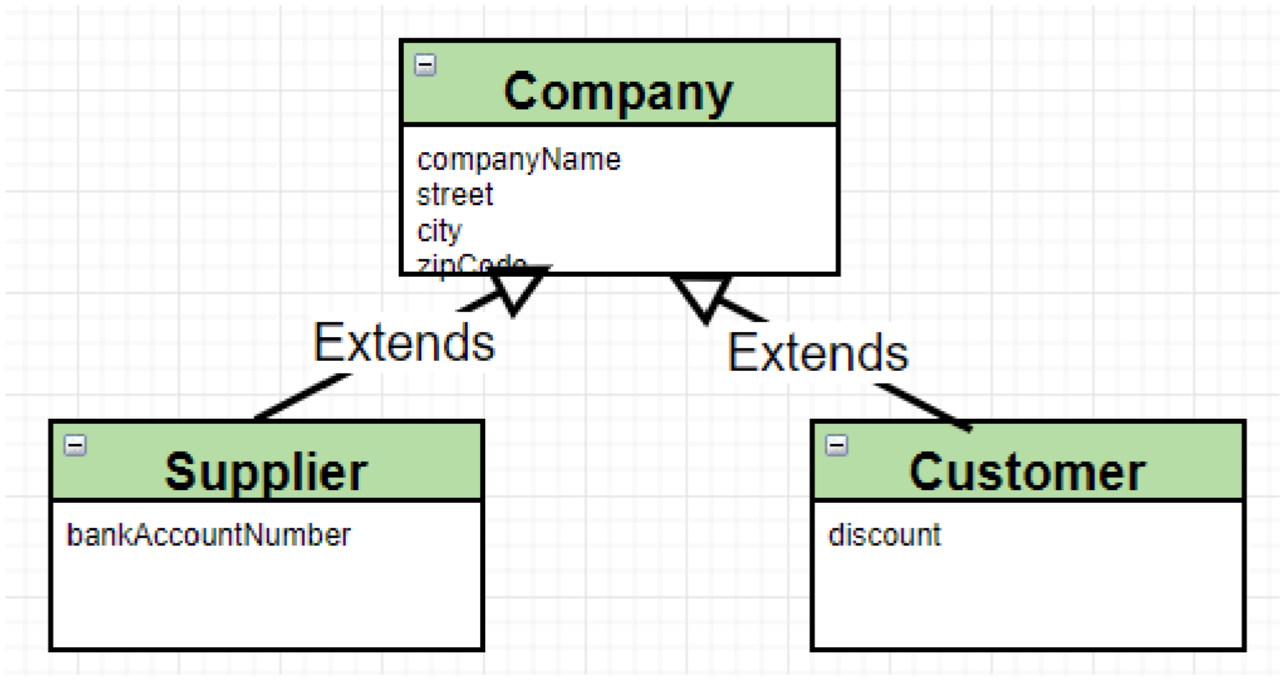


	SUPPLIERID	COMPANYNAME
1	1	Buraki Sp. z o.o.
2	2	Ziemniaki Sp. z o.o.

	UPPLIER…	CITY	STREET
1	1	Burakowo	Barszczowa
2	2	Ziemniakowo	Ziemniaczana

Zadanie VIII

Wprowadź do modelu następującą hierarchię:



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

Na początku tworzymy nową klasę `Company` po której dziedziczyć będą klasy `Supplier` oraz `Customer`:

```java
...

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

    public Company() {

    }

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

    @Override
    public String toString() {
        return "Company ID: " + companyID + "\n" +
                "Company Name: " + companyName + "\n" +
                "Address: " + street + ", " + city + ", " + zipCode + "\n";
    }
}
```

Klasa `Supplier`:

```
...

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


    public Supplier() {
    }

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


    @Override
    public String toString() {
        return "Supplier ID: " + super.getCompanyID() + "\n" +
                "Bank Account Number: " + bankAccountNumber + "\n" +
                super.toString() +
                "\n";
    }
}
```

Klasa `Customer`:

```java
...

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

    public Customer() {
    }

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

    @Override
    public String toString() {
        return "Customer ID: " + super.getCompanyID() + "\n" +
               "Discount: " + discount + "\n" +
               super.toString() +
               "\n";
    }

}
```

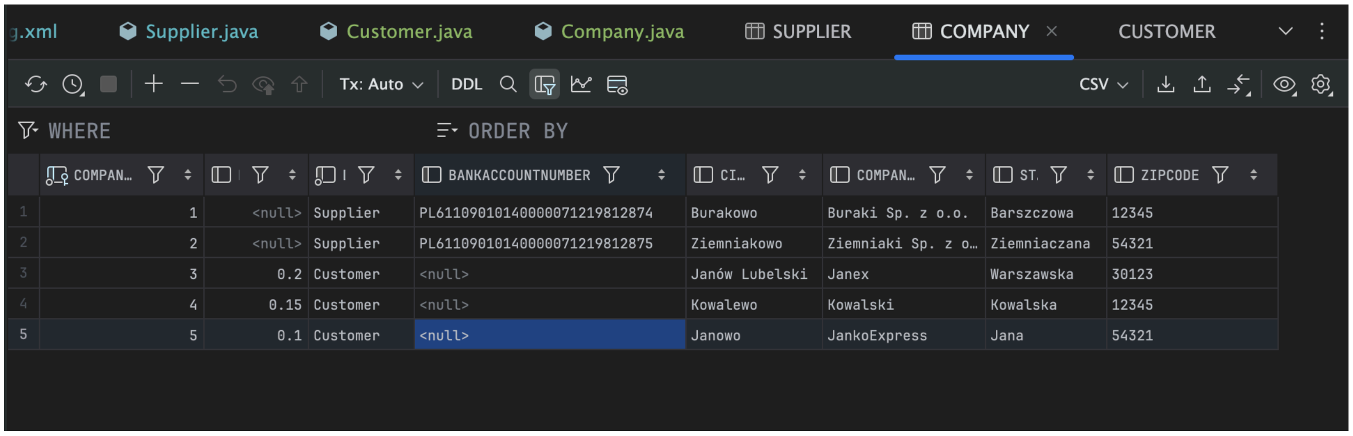
Następnie będziemy wprowadzać różne strategie mapowania dziedziczenia:

- `SINGLE_TABLE`
- `TABLE_PER_CLASS`
- `JOINED`

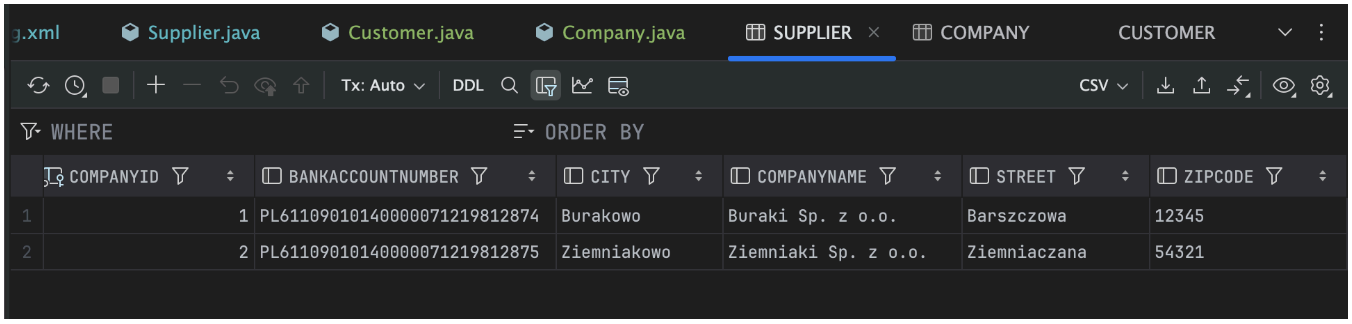
Strategia Table per Class

```
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)
    )
```

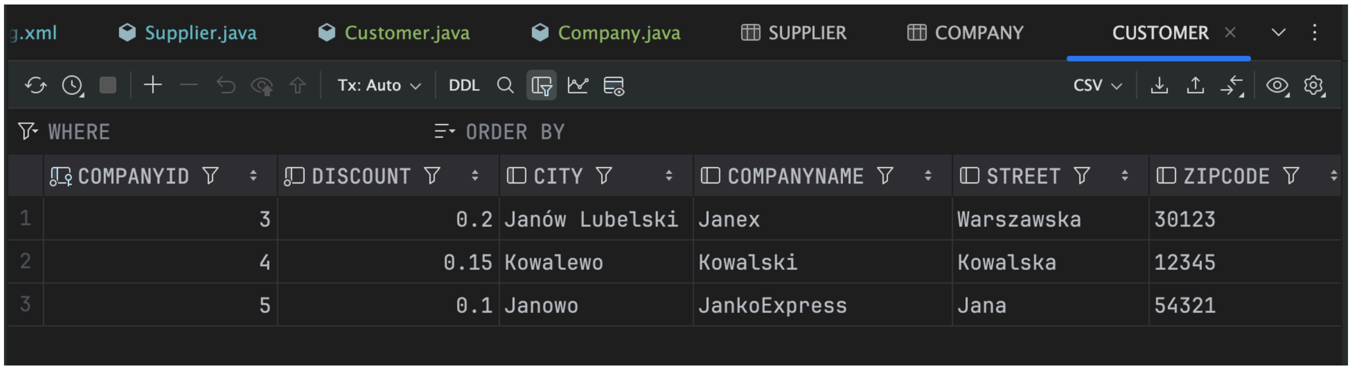
Baza danych po uruchomieniu:



	COMPAN…		I	BANKACCOUNTNUMBER	CI…	COMPAN…	ST.	ZIPCODE
1	1	<null>	Supplier	PL61109010140000071219812874	Burakowo	Buraki Sp. z o.o.	Barszczowa	12345
2	2	<null>	Supplier	PL61109010140000071219812875	Ziemniakowo	Ziemniaki Sp. z o…	Ziemniaczana	54321
3	3	0.2	Customer	<null>	Janów Lubelski	Janex	Warszawska	30123
4	4	0.15	Customer	<null>	Kowalewo	Kowalski	Kowalska	12345
5	5	0.1	Customer	<null>	Janowo	JankoExpress	Jana	54321



	COMPANYID	BANKACCOUNTNUMBER	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	PL61109010140000071219812874	Burakowo	Buraki Sp. z o.o.	Barszczowa	12345
2	2	PL61109010140000071219812875	Ziemniakowo	Ziemniaki Sp. z o.o.	Ziemniaczana	54321



	COMPANYID	DISCOUNT	CITY	COMPANYNAME	STREET	ZIPCODE
1	3	0.2	Janów Lubelski	Janex	Warszawska	30123
2	4	0.15	Kowalewo	Kowalski	Kowalska	12345
3	5	0.1	Janowo	JankoExpress	Jana	54321

```
Supplier found: Supplier ID: 1
Bank Account Number: PL61109010140000071219812874
Company ID: 1
Company Name: Buraki Sp. z o.o.
Address: Barszczowa, Burakowo, 12345
```

```
Customer found: Customer ID: 3
Discount: 0.2
Company ID: 3
Company Name: Janex
Address: Warszawska, Janów Lubelski, 30123
```

Strategia Single Table

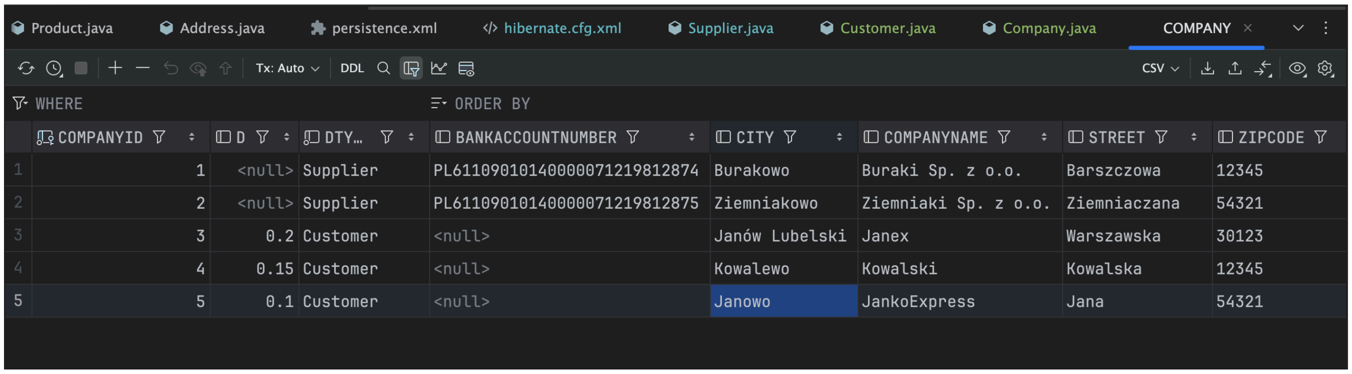
Zmiana strategii dziedziczenia w klasie `Company`:

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

Klasa `Main` pozostaje bez zmian.

```
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)
    )
```

Baza danych po uruchomieniu:



	COMPANYID	D	DTY…	BANKACCOUNTNUMBER	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	<null>	Supplier	PL61109010140000071219812874	Burakowo	Buraki Sp. z o.o.	Barszczowa	12345
2	2	<null>	Supplier	PL61109010140000071219812875	Ziemniakowo	Ziemniaki Sp. z o.o.	Ziemniaczana	54321
3	3	0.2	Customer	<null>	Janów Lubelski	Janex	Warszawska	30123
4	4	0.15	Customer	<null>	Kowalewo	Kowalski	Kowalska	12345
5	5	0.1	Customer	<null>	Janowo	JankoExpress	Jana	54321

```
Supplier found: Supplier ID: 1
Bank Account Number: PL61109010140000071219812874
Company ID: 1
Company Name: Buraki Sp. z o.o.
Address: Barszczowa, Burakowo, 12345
```

```
Customer found: Customer ID: 3
Discount: 0.2
Company ID: 3
Company Name: Janex
Address: Warszawska, Janów Lubelski, 30123
```

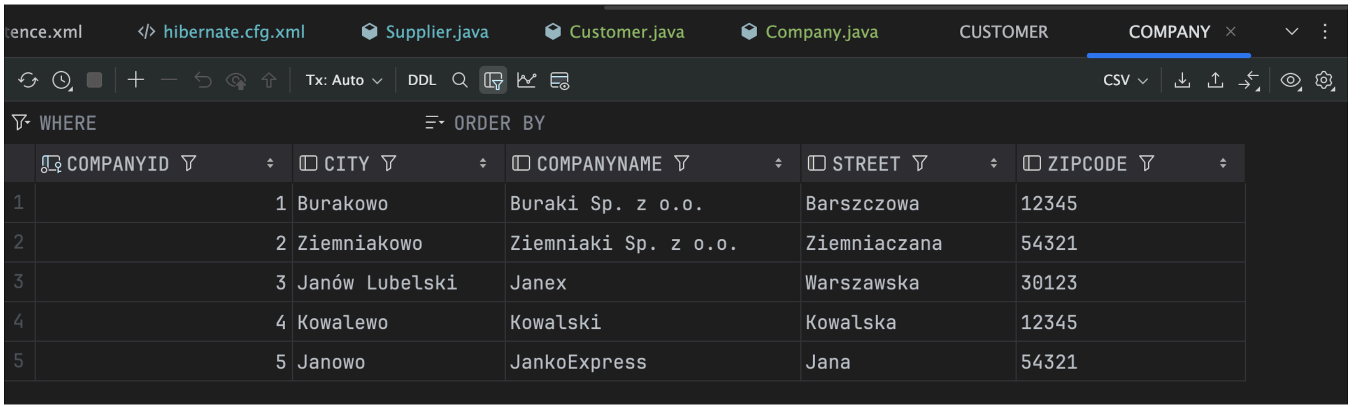
Strategia Joined

Zmiana strategii dziedziczenia w klasie `Company`:

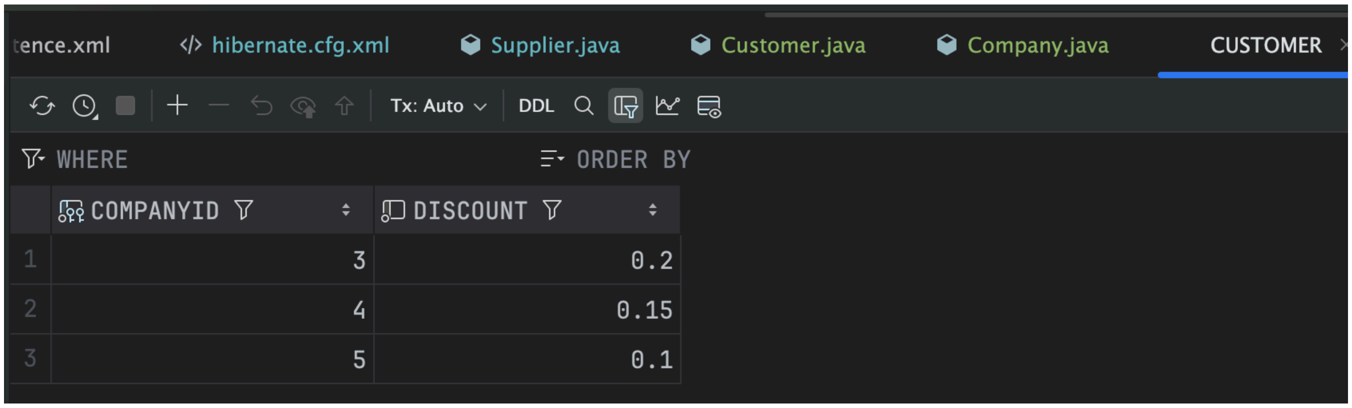
```
@Inheritance(strategy = InheritanceType.JOINED)
```

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

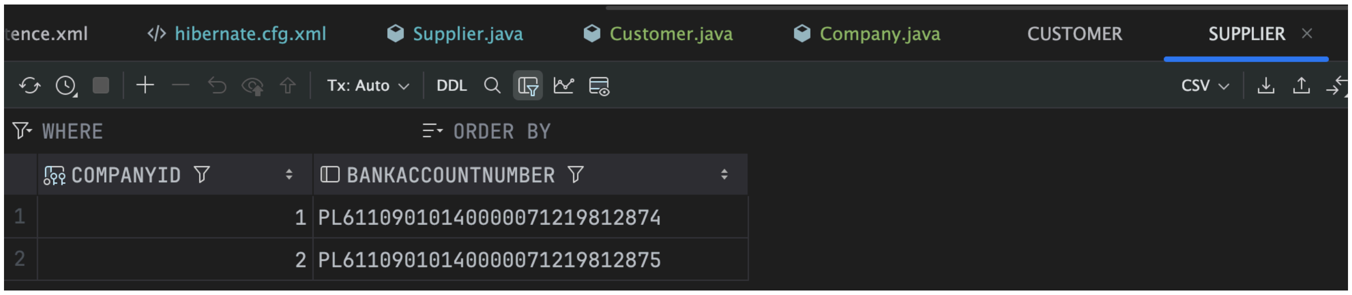
Baza danych po uruchomieniu:



	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	Burakowo	Buraki Sp. z o.o.	Barszczowa	12345
2	2	Ziemniakowo	Ziemniaki Sp. z o.o.	Ziemniaczana	54321
3	3	Janów Lubelski	Janex	Warszawska	30123
4	4	Kowalewo	Kowalski	Kowalska	12345
5	5	Janowo	JankoExpress	Jana	54321



	COMPANYID	DISCOUNT
1	3	0.2
2	4	0.15
3	5	0.1



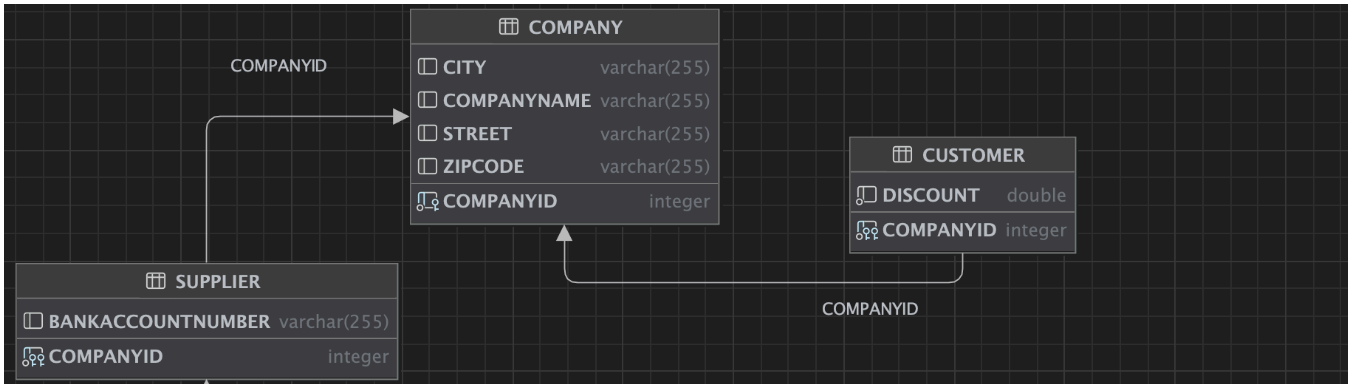
ence.xml | hibernate.cfg.xml | Supplier.java | Customer.java | Company.java | CUSTOMER | SUPPLIER

Tx: Auto | DDL | CSV

WHERE | ORDER BY

	COMPANYID	BANKACCOUNTNUMBER
1	1	PL61109010140000071219812874
2	2	PL61109010140000071219812875

Schemat bazy danych:



```
Supplier found: Supplier ID: 1
Bank Account Number: PL61109010140000071219812874
Company ID: 1
Company Name: Buraki Sp. z o.o.
Address: Barszczowa, Burakowo, 12345
```

```
Customer found: Customer ID: 3
Discount: 0.2
Company ID: 3
Company Name: Janex
Address: Warszawska, Janów Lubelski, 30123
```