

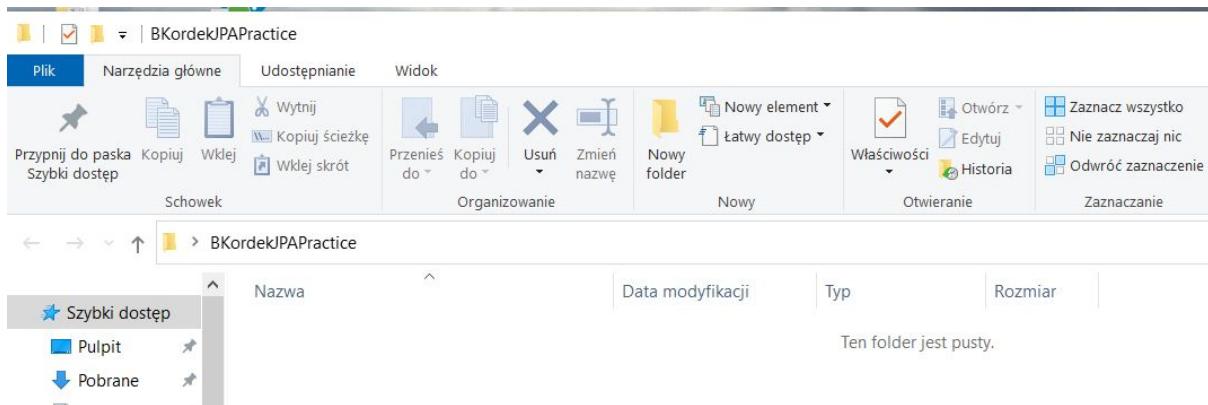
# Bartosz Kordek

## Bazy danych

### Laboratorium - Hibernate/JPA

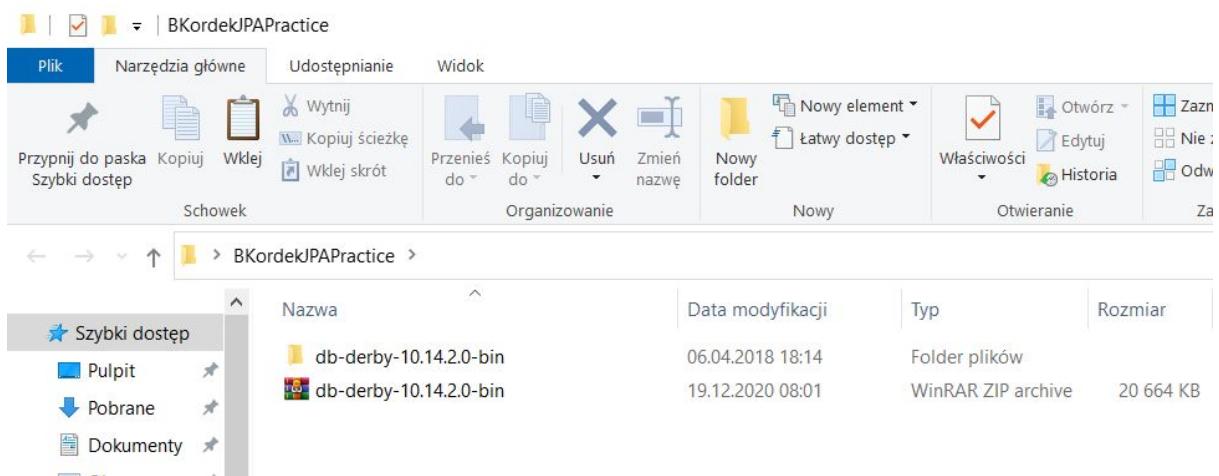
#### I. Basics

##### a. Stwórz folder INazwiskoJPAPractice



##### b. Sciagnij i rozpakuj w stworzonym folderze DBMS Derby

<https://www.eu.apache.org/dist//db/derby/db-derby-10.14.2.0/db-derby-10.14.2.0-bin.zip>

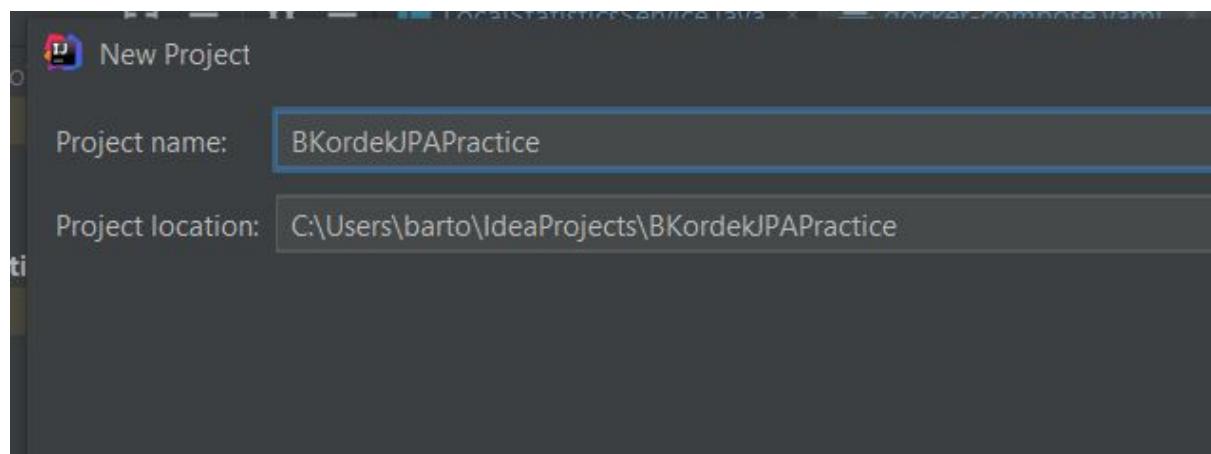
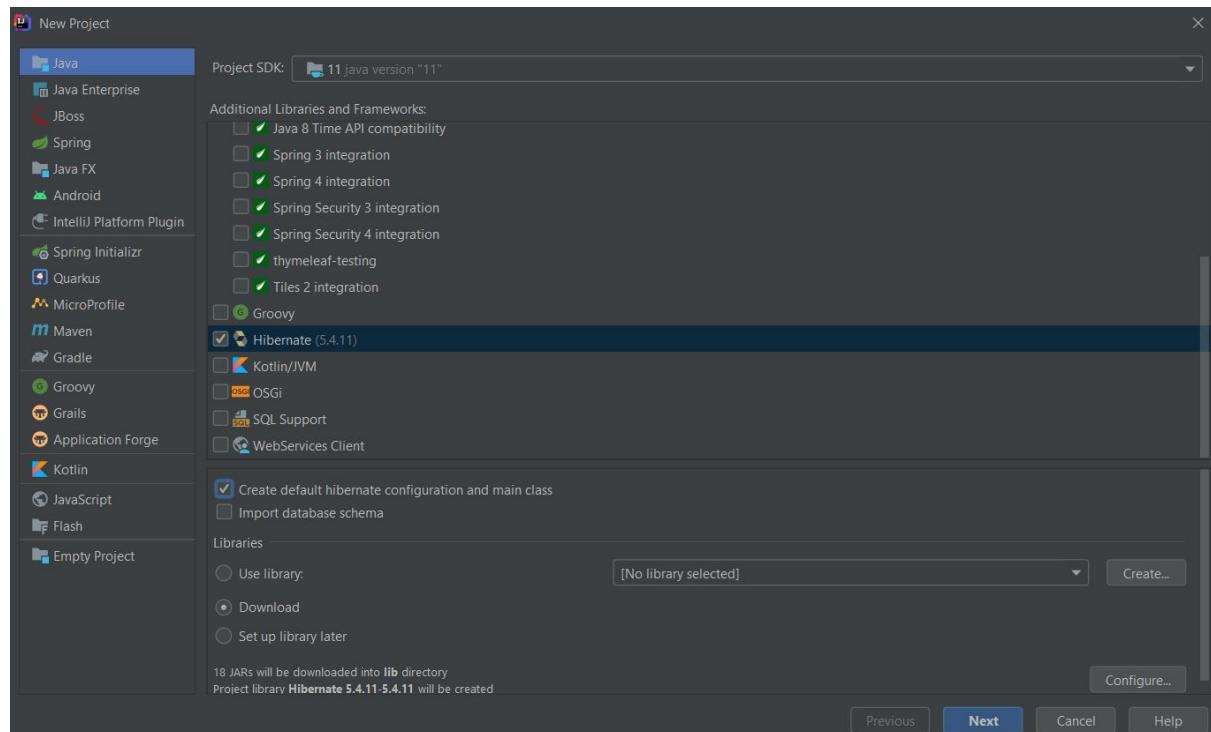


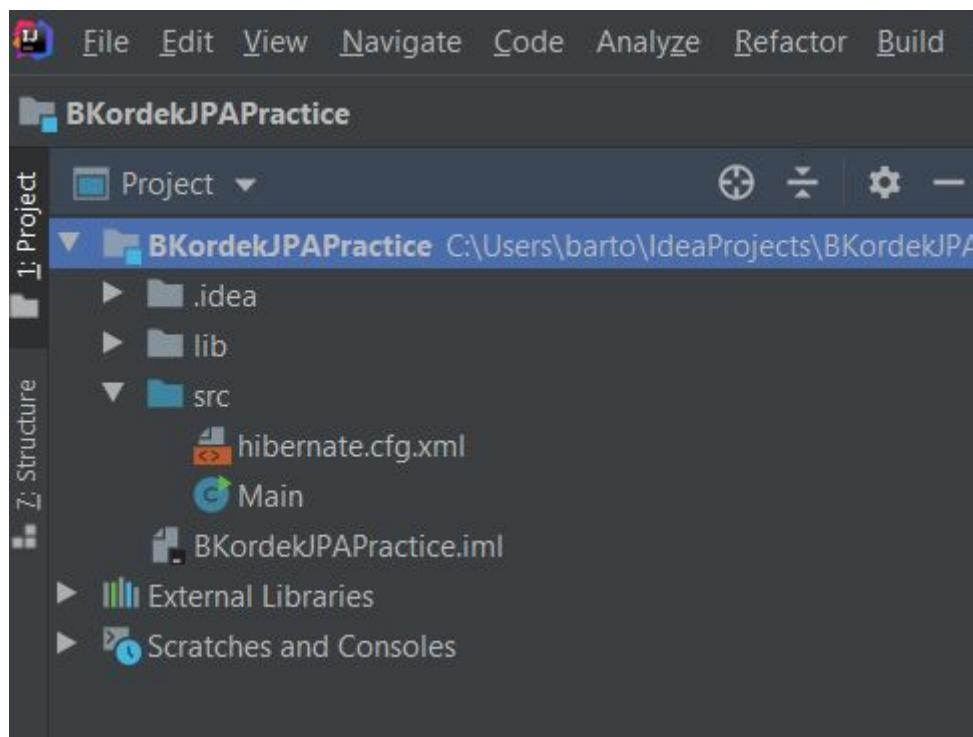
##### c. Uruchom serwer Derby (skrypt startNetworkSerwer z podkatalogu bin)

```
C:\WINDOWS\system32\cmd.exe
Sat Dec 19 08:45:55 CET 2020 : Security manager installed using the Basic server security policy.
Sat Dec 19 08:46:00 CET 2020 : Serwer sieciowy Apache Derby - 10.14.2.0 - (1828579) uruchomiony i gotowy do zaakceptowania po "cze" na porcie 1527 w {3}
```

A screenshot of a Windows Command Prompt window. The title bar says 'C:\WINDOWS\system32\cmd.exe'. The window displays the output of a command, likely 'java -jar derby.jar -startNetworkServer'. It shows the security manager being installed and the Derby server starting up, indicating it's ready to accept connections on port 1527.

d. Stwórz projekt Javowy o nazwie InazwiskoJPAPractice

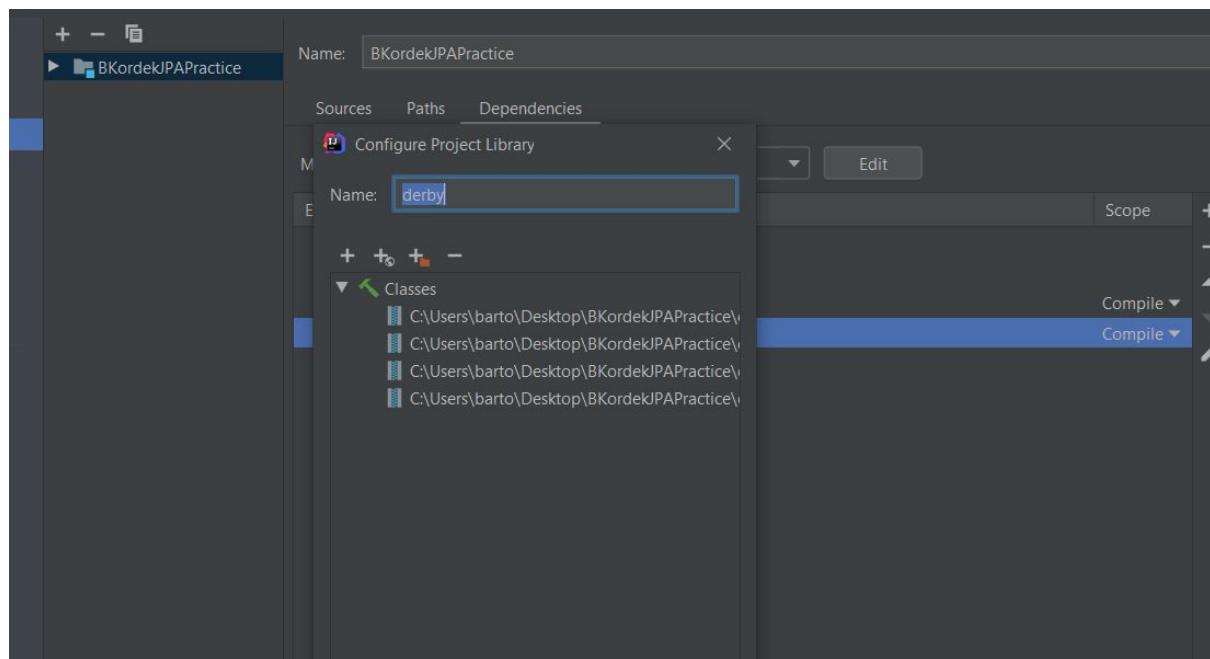
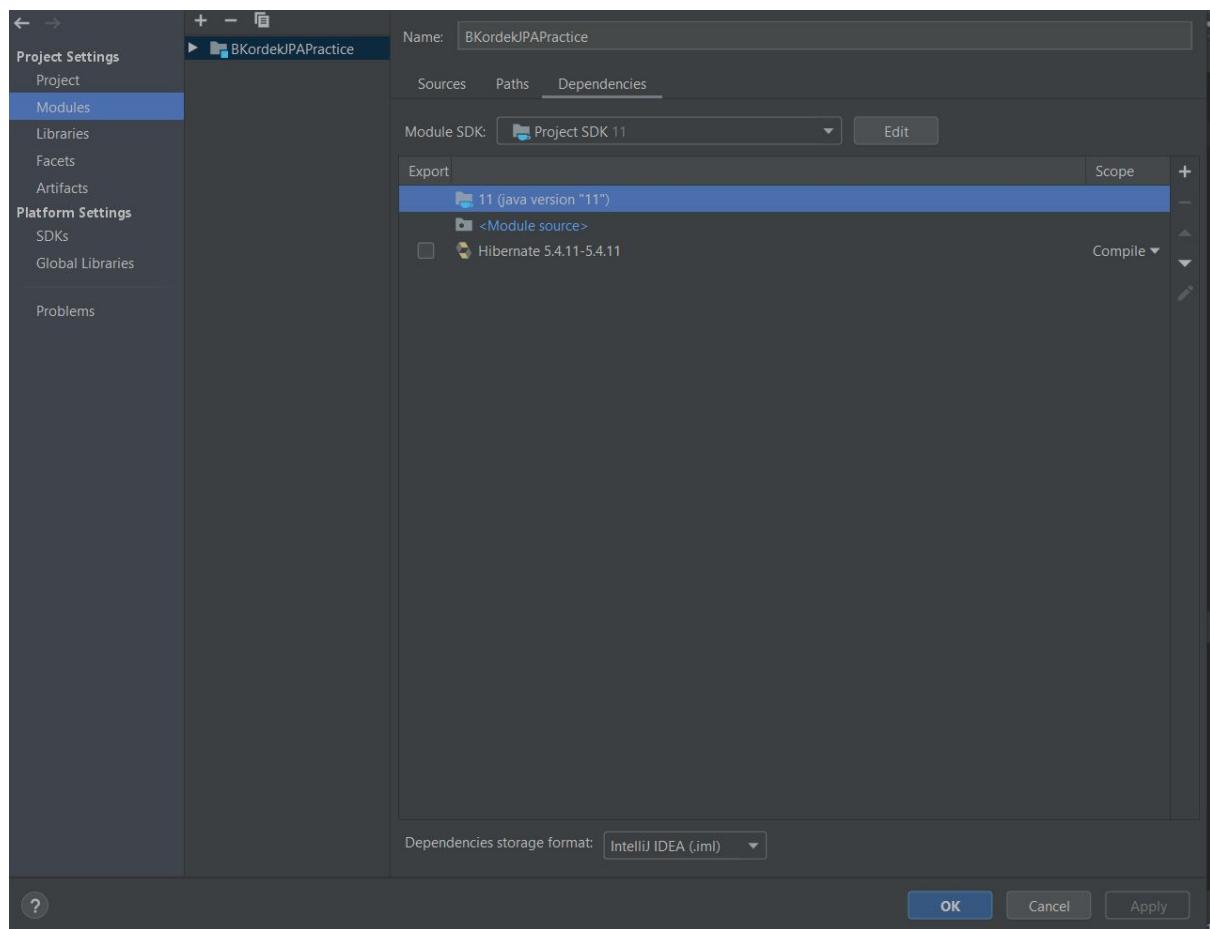


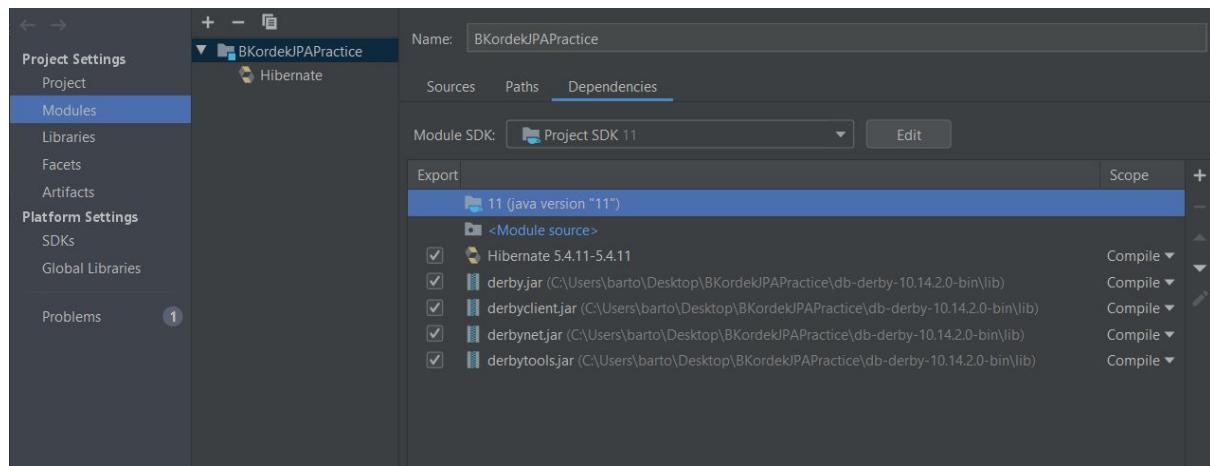


- e. Dołącz do projektu ( w IJ: File→Project Structure → Modules→ Dependencies) Jar-ki Związane z obsługą/komunikacją z Derby (*derby.jar*, *derbyclient.jar*, *derbynnet.jar*, *derbytools.jar*).

> Ten komputer > Pulpit > BKordekJPAPractice > db-derby-10.14.2.0-bin > lib

Nazwa	Data modyfikacji	Typ	Rozmiar
derby	06.04.2018 18:10	Executable Jar File	3 158 KB
derby.war	06.04.2018 18:10	Plik WAR	2 KB
derbyclient	06.04.2018 18:10	Executable Jar File	575 KB
derbyLocale_cs	06.04.2018 18:10	Executable Jar File	93 KB
derbyLocale_de_DE	06.04.2018 18:10	Executable Jar File	110 KB
derbyLocale_es	06.04.2018 18:10	Executable Jar File	104 KB
derbyLocale_fr	06.04.2018 18:10	Executable Jar File	110 KB
derbyLocale_hu	06.04.2018 18:10	Executable Jar File	94 KB
derbyLocale_it	06.04.2018 18:10	Executable Jar File	104 KB
derbyLocale_ja_JP	06.04.2018 18:10	Executable Jar File	121 KB
derbyLocale_ko_KR	06.04.2018 18:10	Executable Jar File	115 KB
derbyLocale_pl	06.04.2018 18:10	Executable Jar File	92 KB
derbyLocale_pt_BR	06.04.2018 18:10	Executable Jar File	89 KB
derbyLocale_ru	06.04.2018 18:10	Executable Jar File	119 KB
derbyLocale_zh_CN	06.04.2018 18:10	Executable Jar File	108 KB





f. Uzupełnij wpisy w *hibernate.cfg.xml*

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="connection.url">jdbc:derby://127.0.0.1/BKordekJPA;create=true</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hbm2ddl.auto">create-drop</property>
    </session-factory>
</hibernate-configuration>
```

## g. Uruchom projekt

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure for 'BKordekJPAPractice' with 'src' selected. Inside 'src', there is a 'Main' package containing 'Main.java' and 'hibernate.cfg.xml'. The right pane shows the code for 'Main.java':

```
import ...  
public class Main {  
    private static final SessionFactory ourSessionFactory;  
    static {  
        try {  
            Configuration configuration = new Configuration();  
            configuration.configure();  
  
            ourSessionFactory = configuration.buildSessionFactory();  
        } catch (Throwable ex) {  
            throw new ExceptionInInitializerError(ex);  
        }  
    }  
  
    public static Session getSession() throws HibernateException {  
        return ourSessionFactory.openSession();  
    }  
  
    public static void main(final String[] args) throws Exception {  
        final Session session = getSession();  
    }  
}
```

The 'Run' tool window at the bottom shows the application output:

```
gru 19, 2020 9:22:12 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>  
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)  
gru 19, 2020 9:22:13 AM org.hibernate.dialect.Dialect <init>  
INFO: HHH000400: Using dialect: org.hibernate.dialect.DerbyTenSevenDialect  
gru 19, 2020 9:22:14 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService  
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]  
querying all the managed entities...  
  
Process finished with exit code 0
```

The screenshot shows a terminal window with the title 'Run: Main'. It displays the same log output as the IntelliJ IDEA run window, indicating the successful execution of the application.

```
"C:\Program Files\OpenJDK11\jdk-11\bin\java.exe" ...  
gru 19, 2020 9:22:08 AM org.hibernate.Version logVersion  
INFO: HHH000412: Hibernate Core {5.4.11.Final}  
gru 19, 2020 9:22:09 AM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>  
INFO: HCANN000001: Hibernate Commons Annotations {5.1.0.Final}  
gru 19, 2020 9:22:12 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure  
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)  
gru 19, 2020 9:22:12 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator  
INFO: HHH10001005: using driver [org.apache.derby.jdbc.ClientDriver] at URL [jdbc:derby://127.0.0.1/BKordekJPA;create=true]  
gru 19, 2020 9:22:12 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator  
INFO: HHH10001001: Connection properties: {}  
gru 19, 2020 9:22:12 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator  
INFO: HHH10001003: Autocommit mode: false  
gru 19, 2020 9:22:12 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>  
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)  
gru 19, 2020 9:22:13 AM org.hibernate.dialect.Dialect <init>  
INFO: HHH000400: Using dialect: org.hibernate.dialect.DerbyTenSevenDialect  
gru 19, 2020 9:22:14 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService  
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]  
querying all the managed entities...  
  
Process finished with exit code 0
```

## h. Schemat APP – na razie pusty

The screenshot shows the 'Data Sources and Drivers' configuration window and the 'Database' browser interface.

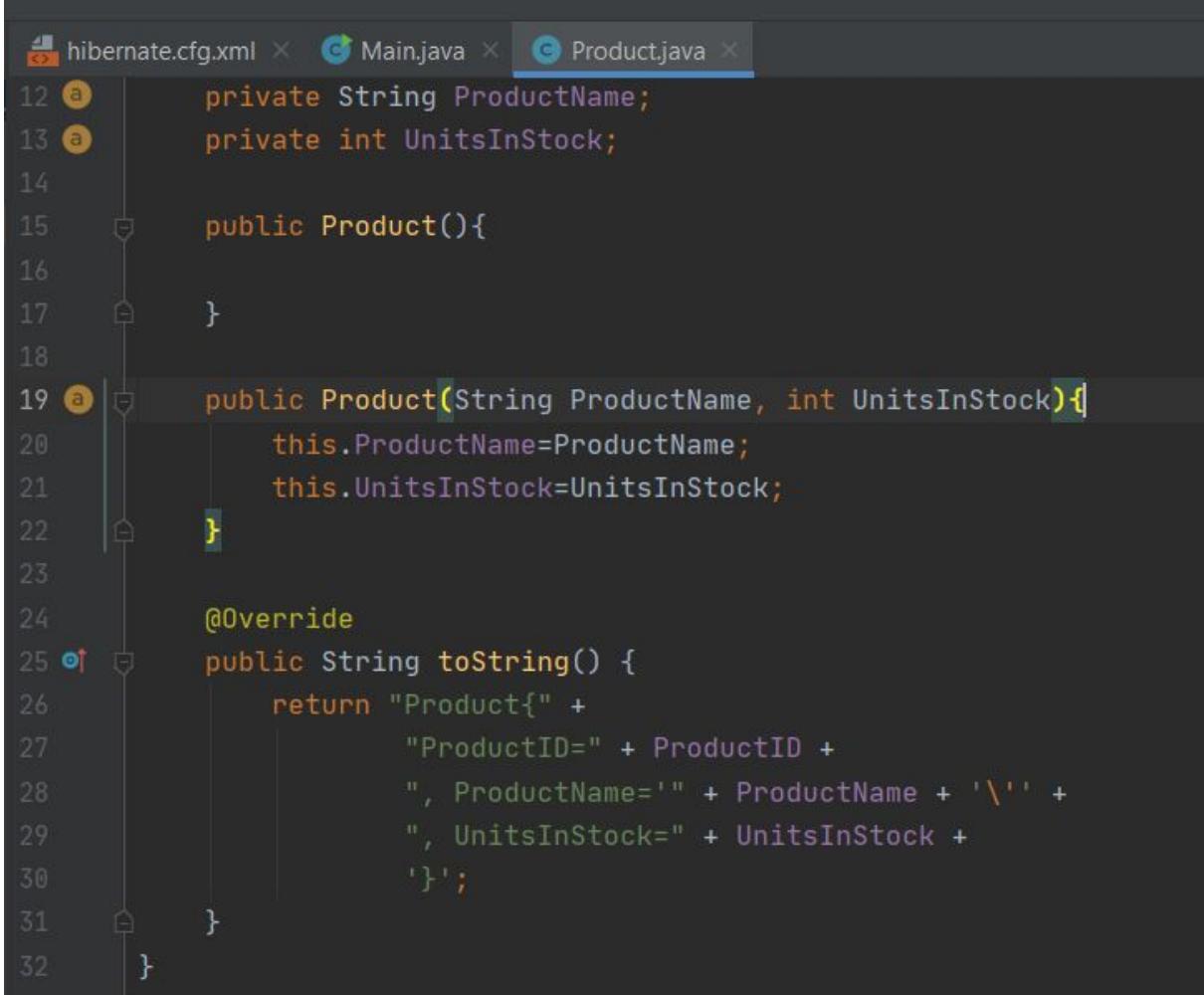
**Data Sources and Drivers Dialog:**

- Name:** jdbc:derby://127.0.0.1/BKordekJPA
- Comment:** (empty)
- General Tab:** Connection type: default, Driver: Apache Derby (Remote). Fields include Host: 127.0.0.1, Port: (empty), User: (empty), Password: <hidden>, Save: Forever, Database: BKordekJPA, URL: jdbc:derby://127.0.0.1/BKordekJPA (highlighted with a blue border).
- Drivers List:** A scrollable list of database drivers including Amazon Redshift, Apache Cassandra, Apache Derby (Embedded), Apache Derby (Remote), Apache Hive, Azure SQL Database, ClickHouse, Exasol, Greenplum, H2, HSQLDB (Local), HSQLDB (Remote), IBM Db2, IBM Db2 (JTOpen), MariaDB, Microsoft SQL Server, Microsoft SQL Server (jTds), MongoDB, MySQL, and others.

**Database Browser:**

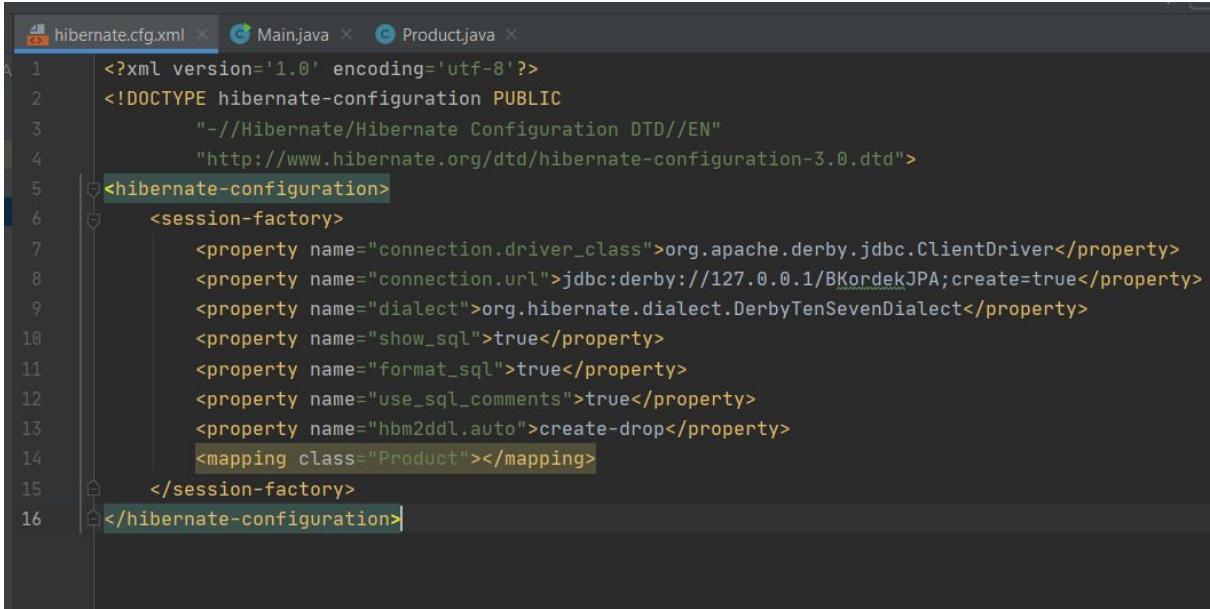
- Shows a tree view of databases:
  - jdbc:derby://127.0.0.1/BKordekJPA (selected, showing 1 of 11 tables)
  - APP (empty)
- Toolbar icons: New, Open, Refresh, Stop, Filter, Sort, Settings, Close.
- Right sidebar: Graph Database.

- i. Stwórz klasę produktu z polami *ProductName*, *UnitsOnStock*. Zapewnij Mapowanie klasy do tabeli *Products*



```
hibernate.cfg.xml × Main.java × Product.java ×
12     private String ProductName;
13     private int UnitsInStock;
14
15     public Product(){
16
17     }
18
19     public Product(String ProductName, int UnitsInStock){
20         this.ProductName=ProductName;
21         this.UnitsInStock=UnitsInStock;
22     }
23
24     @Override
25     public String toString() {
26         return "Product{" +
27             "ProductID=" + ProductID +
28             ", ProductName='" + ProductName + '\'' +
29             ", UnitsInStock=" + UnitsInStock +
30             '}';
31     }
32 }
```

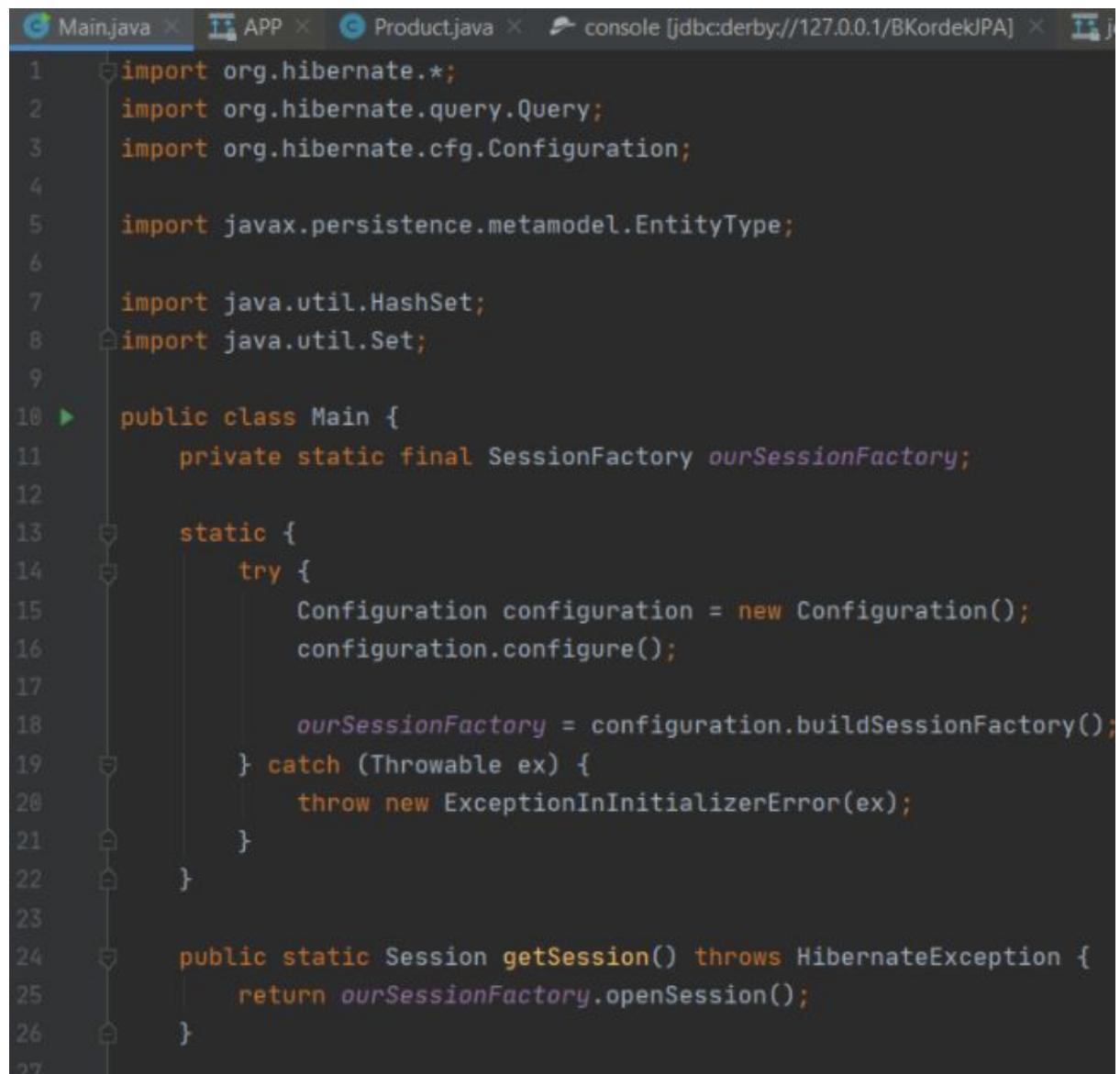
- j. Uzupełnij w projekcie elementy potrzebne do zmapowania klasy do bazy danych



```
hibernate.cfg.xml × Main.java × Product.java ×
1  <?xml version='1.0' encoding='utf-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3      "-//Hibernate/Hibernate Configuration DTD//EN"
4      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5  <hibernate-configuration>
6      <session-factory>
7          <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
8          <property name="connection.url">jdbc:derby://127.0.0.1/BKordekJPA;create=true</property>
9          <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
10         <property name="show_sql">true</property>
11         <property name="format_sql">true</property>
12         <property name="use_sql_comments">true</property>
13         <property name="hbm2ddl.auto">create-drop</property>
14         <mapping class="Product"></mapping>
15     </session-factory>
16 </hibernate-configuration>
```

- k. Jeżeli nie wygenerowałeś maina z tworzeniem sesji na etapie zakładania projektu w pliku głównym zapewnij tworzenie sesji i jej otwieranie.

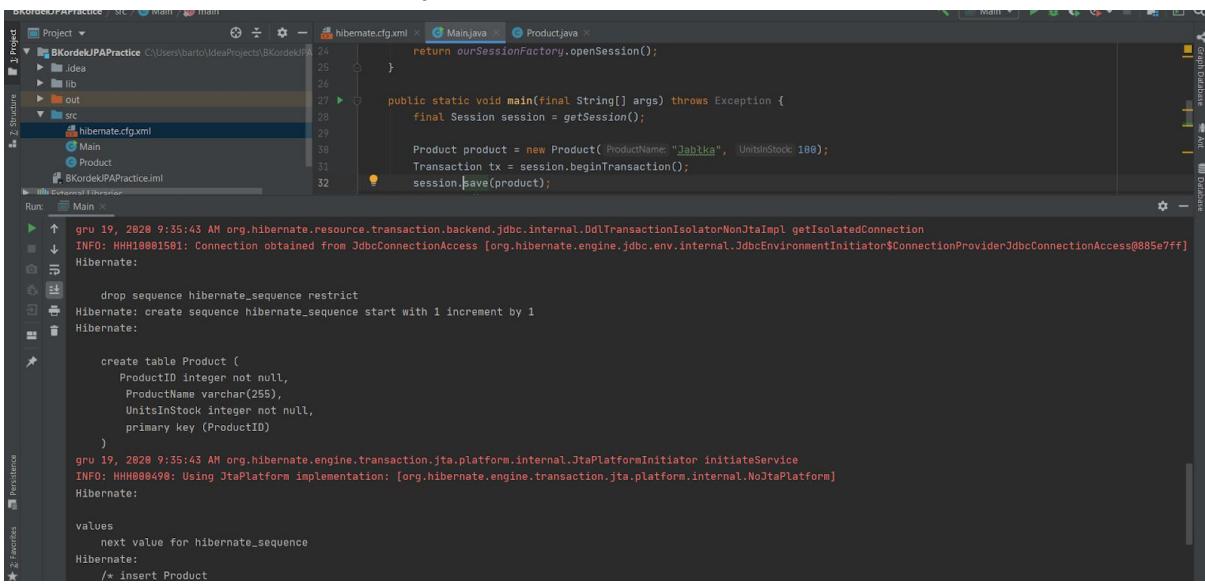
l.



The screenshot shows a Java code editor with the file 'Main.java' open. The code implements a static factory pattern for a SessionFactory. It imports various Hibernate and Java util classes, defines a static block to build the factory, and provides a public static method to get a session. The code is annotated with line numbers from 1 to 27.

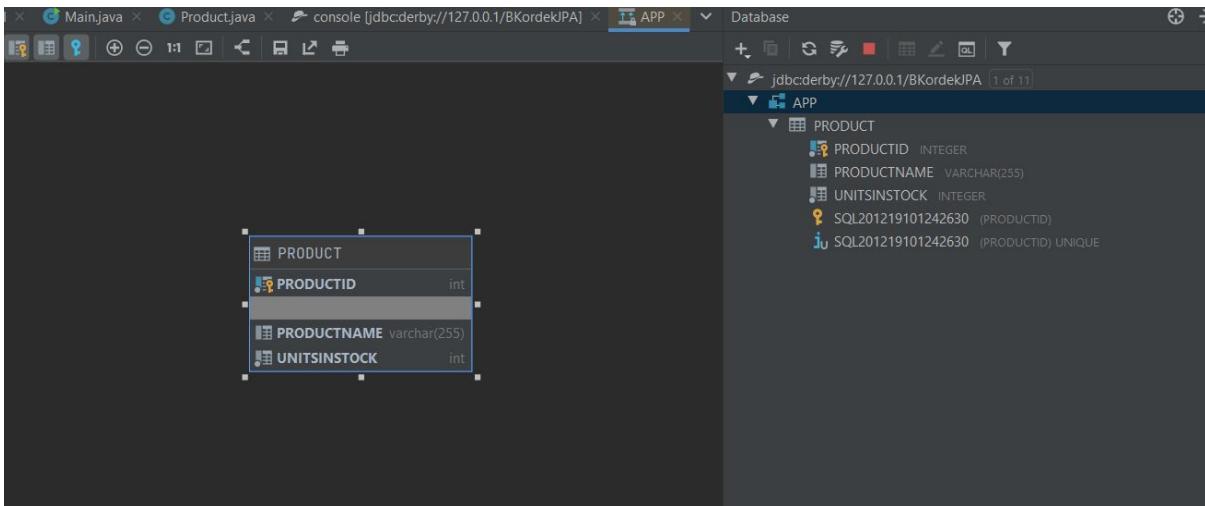
```
1 import org.hibernate.*;
2 import org.hibernate.query.Query;
3 import org.hibernate.cfg.Configuration;
4
5 import javax.persistence.metamodel.EntityType;
6
7 import java.util.HashSet;
8 import java.util.Set;
9
10 public class Main {
11     private static final SessionFactory ourSessionFactory;
12
13     static {
14         try {
15             Configuration configuration = new Configuration();
16             configuration.configure();
17
18             ourSessionFactory = configuration.buildSessionFactory();
19         } catch (Throwable ex) {
20             throw new ExceptionInInitializerError(ex);
21         }
22     }
23
24     public static Session getSession() throws HibernateException {
25         return ourSessionFactory.openSession();
26     }
27 }
```

### m. Uruchomienie projektu

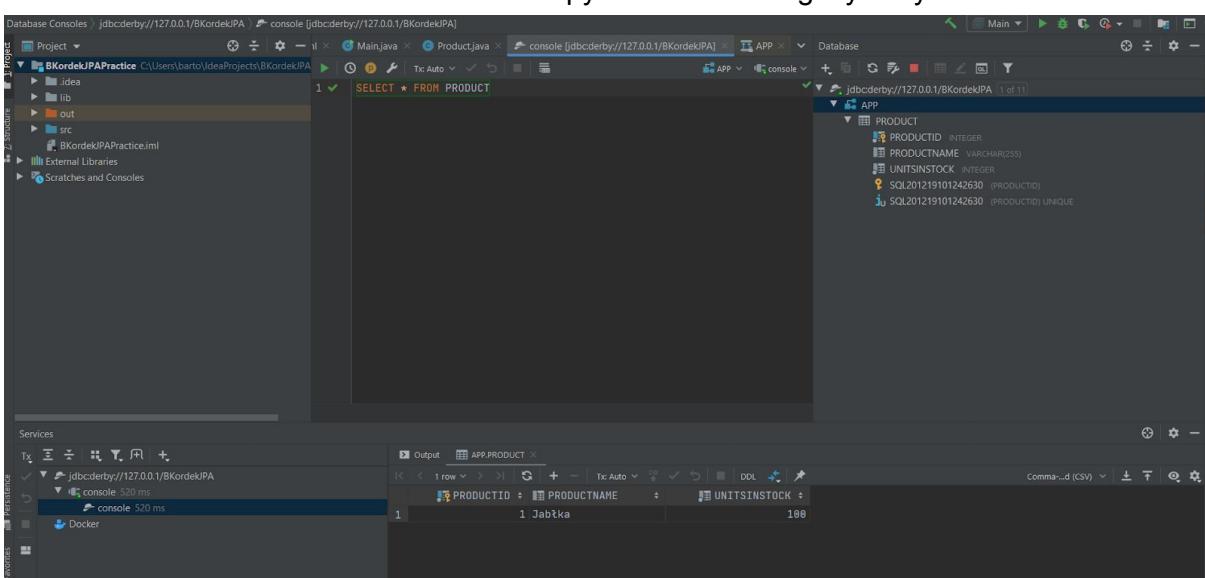


```
gru 19, 2020 9:35:43 AM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@8085e7ff]
Hibernate:
    drop sequence hibernate_sequence restrict
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:
    create table Product (
        ProductID integer not null,
        ProductName varchar(255),
        UnitsInStock integer not null,
        primary key (ProductID)
    )
gru 19, 2020 9:35:43 AM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate:
    values
    next value for hibernate_sequence
Hibernate:
    /* insert Product
```

### n. Widok schematu bazy danych z wykorzystaniem IntelliJ



### o. Zawartość tabeli *Product*. Zapytanie o stan magazynowy



```
Database Consoles > jdbcderby://127.0.0.1/BKordekJPA > console [jdbcderby://127.0.0.1/BKordekJPA]
1 ✓ SELECT * FROM PRODUCT
APP
  PRODUCT
    PRODUCTID INTEGER
    PRODUCTNAME VARCHAR(255)
    UNITSINSTOCK INTEGER
    SQL201219101242630 (PRODUCTID)
    SQL201219101242630 (PRODUCTNAME) UNIQUE
```

PRODUCTID	PRODUCTNAME	UNITSINSTOCK
1	Jabłka	100

p. Widok tabel w bazie danych z wykorzystaniem shell'a serwera derby

en komputer > Pulpit > BKordekJPAPractice > db-derby-10.14.2.0-bin > bin

Nazwa	^	Data modyfikacji	Typ	Rozmiar
BKordekJPA		19.12.2020 09:22	Folder plików	
dblock		06.04.2018 18:09	Plik	6 KB
dblock		10.03.2018 08:31	Plik wsadowy Win...	2 KB
derby		19.12.2020 09:22	Dokument tekstowy	6 KB
derby_common		10.03.2018 08:31	Plik wsadowy Win...	3 KB
ij		06.04.2018 18:09	Plik	6 KB
ij		10.03.2018 08:31	Plik wsadowy Win...	2 KB
NetworkServerControl		06.04.2018 18:09	Plik	6 KB
NetworkServerControl		10.03.2018 08:31	Plik wsadowy Win...	2 KB
setEmbeddedCP		10.03.2018 08:31	Plik	2 KB
setEmbeddedCP		10.03.2018 08:31	Plik wsadowy Win...	2 KB
setNetworkClientCP		10.03.2018 08:31	Plik	2 KB
setNetworkClientCP		10.03.2018 08:31	Plik wsadowy Win...	2 KB
setNetworkServerCP		10.03.2018 08:31	Plik	2 KB
setNetworkServerCP		10.03.2018 08:31	Plik wsadowy Win...	2 KB

```
C:\WINDOWS\system32\cmd.exe
ij> show tables;
TABLE_SCHEM | TABLE_NAME | REMARKS
-----+-----+-----+
SYS      | SYSALIASES
SYS      | SYSCHECKS
SYS      | SYSCOLPERMS
SYS      | SYSCOLUMNS
SYS      | SYSCONGLOMERATES
SYS      | SYSCONSTRAINTS
SYS      | SYSDEPENDS
SYS      | SYSFILES
SYS      | SYSFOREIGNKEYS
SYS      | SYSKEYS
SYS      | SYSPERMS
SYS      | SYSROLES
SYS      | SYSROUTINEPERMS
SYS      | SYSSCHEMAS
SYS      | SYSSEQUENCES
SYS      | SYSSTATEMENTS
SYS      | SYSSTATISTICS
SYS      | SYSTABLEPERMS
SYS      | SYSTABLES
SYS      | SYSTRIGGERS
SYS      | SYSUSERS
SYS      | SYSVIEWS
SYSIBM   | SYSUMMY1
APP     | PRODUCT

24 wierszy wybranych
ij>
```

q. Dodanie produktu do bazy z wykorzystaniem klasy Scanner

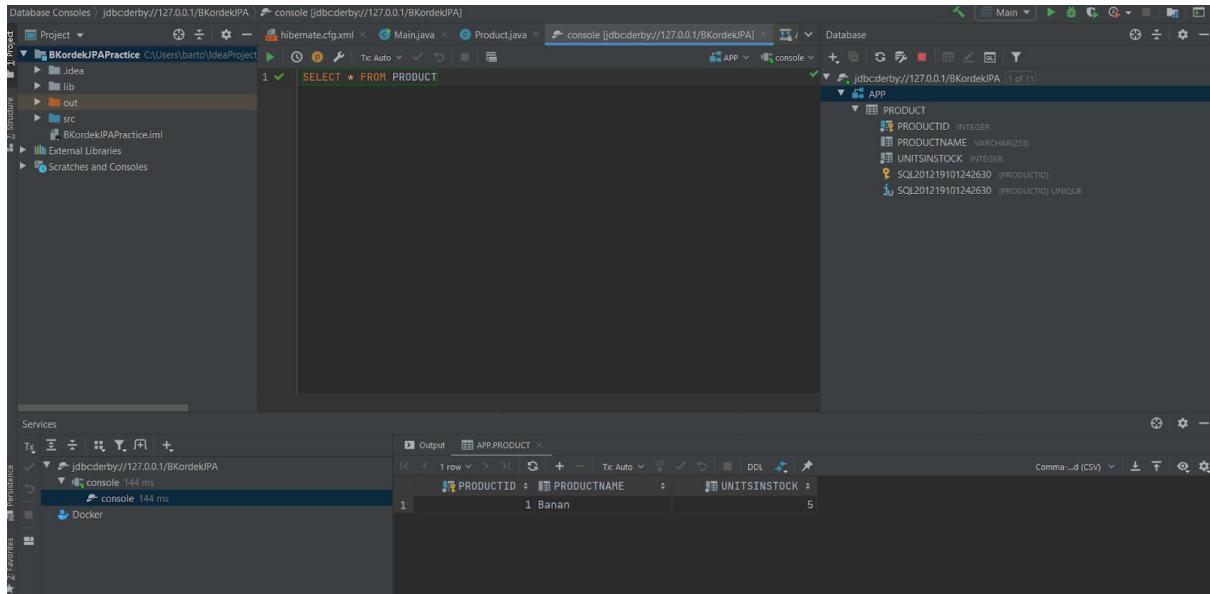
```
public static void main(final String[] args) throws Exception {  
  
    Scanner scanner = new Scanner(System.in);  
    System.out.println("Podaj nazwę produktu: ");  
    String productName = scanner.nextLine();  
    System.out.println("Podaj ilość: ");  
    int unitsInStock = scanner.nextInt();  
    Product product = new Product(productName, unitsInStock);  
  
    final Session session = getSession();
```

```
gru 17, 2020 10:41:07 AM org.hibernate.engine.transaction  
INFO: HHH000490: Using JtaPlatform implementation: [org.h  
Podaj nazwę produktu:  
Banana  
Podaj ilość:  
5  
Hibernate:
```

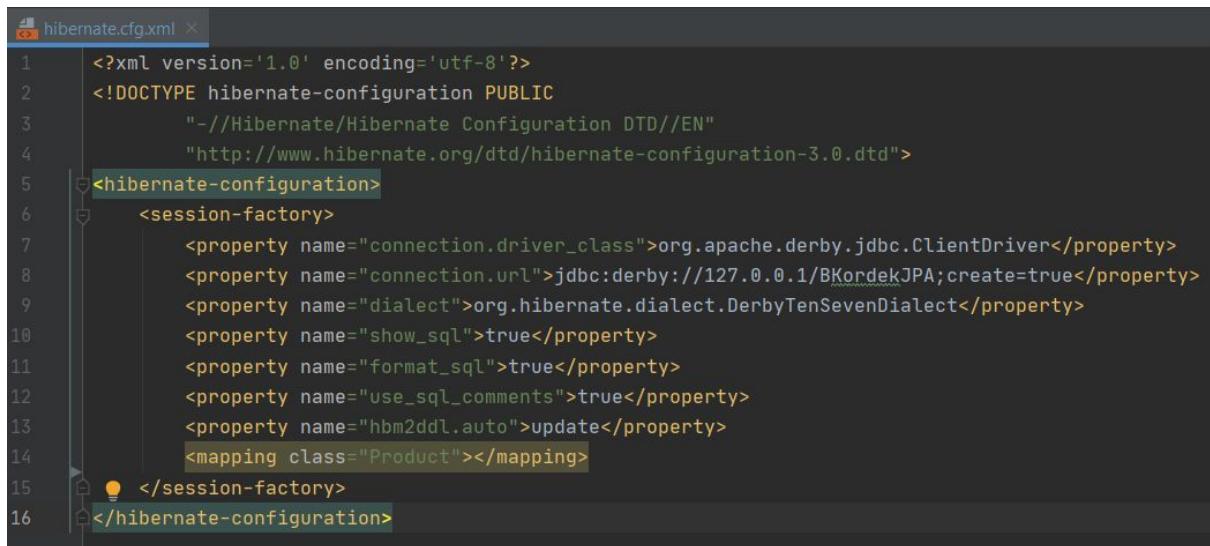
```
Run: Main ×  
[Run] ↑ (ProductName, UnitsInStock, ProductID)  
[Stop] ↓ values  
[Break] ↴ (?, ?, ?)  
[Screenshot] ↵ querying all the managed entities...  
[Minimize] ↶ executing: from Product  
[Maximize] ↷ Hibernate:  
[Close] /*  
from  
Product * select  
product0_.ProductID as producti1_0_,  
product0_.ProductName as productn2_0_,  
product0_.UnitsInStock as unitsins3_0_  
from  
Product product0_  
Product{ProductID=1, ProductName='Banan', UnitsInStock=5}  
  
Process finished with exit code 0
```

## r. Sprawdzenie stanu magazynowego

- W przypadku właściwości create-drop w pliku konfiguracyjnym



- W przypadku właściwości update w pliku konfiguracyjnym



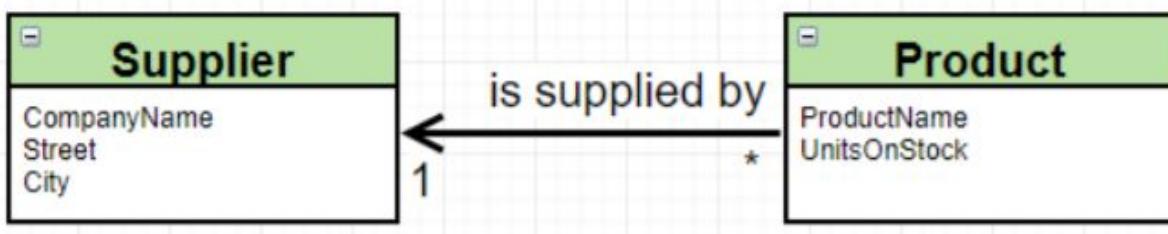
The screenshot shows the IntelliJ IDEA interface with a database console window. The query `SELECT * FROM PRODUCT` is executed, and the results are displayed in a table:

	PRODUCTNAME	UNITSINSTOCK
1	Gruszka	2
2	Melon	4

The database structure on the right side of the interface shows the `PRODUCT` table with columns `PRODUCTID`, `PRODUCTNAME`, and `UNITSINSTOCK`. It also shows the `SUPPLIER` table.

Z powyższego widać, że został pierwszy produkt nie został usunięty po restartie programu.

#### IV. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej:



Dodanie mapowania klasy *Supplier* w pliku konfiguracyjnym *hibernate.cfg.xml*

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
    <property name="connection.url">jdbc:derby://127.0.0.1/BKordekJPA;create=true</property>
    <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="use_sql_comments">true</property>
    <property name="hbm2ddl.auto">create-drop</property>
    <mapping class="Product"></mapping>
    <mapping class="Supplier"></mapping>
  </session-factory>
</hibernate-configuration>
  
```

## Klasa Supplier



The screenshot shows a Java code editor with the file `Supplier.java` open. The code defines a class `Supplier` with annotations for entity status and primary key generation. It includes a constructor, getters for the primary key, and an overridden `toString` method. The code is annotated with various icons from the JavaDoc API documentation.

```
6  @Entity
7  public class Supplier {
8
9      @Id
10     @GeneratedValue(strategy = GenerationType.AUTO)
11     private int SupplierID;
12     private String CompanyName;
13     private String Street;
14     private String City;
15
16     public Supplier(){
17
18     }
19
20     public Supplier(String CompanyName, String Street, String Ci
21         this.CompanyName = CompanyName;
22         this.Street = Street;
23         this.City = City;
24     }
25
26     public int getSupplierID() {
27         return SupplierID;
28     }
29
30     @Override
31     public String toString() {
32         return "Supplier{" +
33             "SupplierID=" + SupplierID +
34             ", CompanyName='" + CompanyName + '\'' +
35             ", Street='" + Street + '\'' +
36             ", City='" + City + '\'' +
37             ...
38     }
39
40 }
```

## Klasa Product

```
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;
    }

    @Override
    public String toString() {
        return "Product{" +
            "ProductID=" + ProductID +
            ", ProductName='" + ProductName + '\'' +
            ", UnitsInStock=" + UnitsInStock +
            '}';
    }
}
```

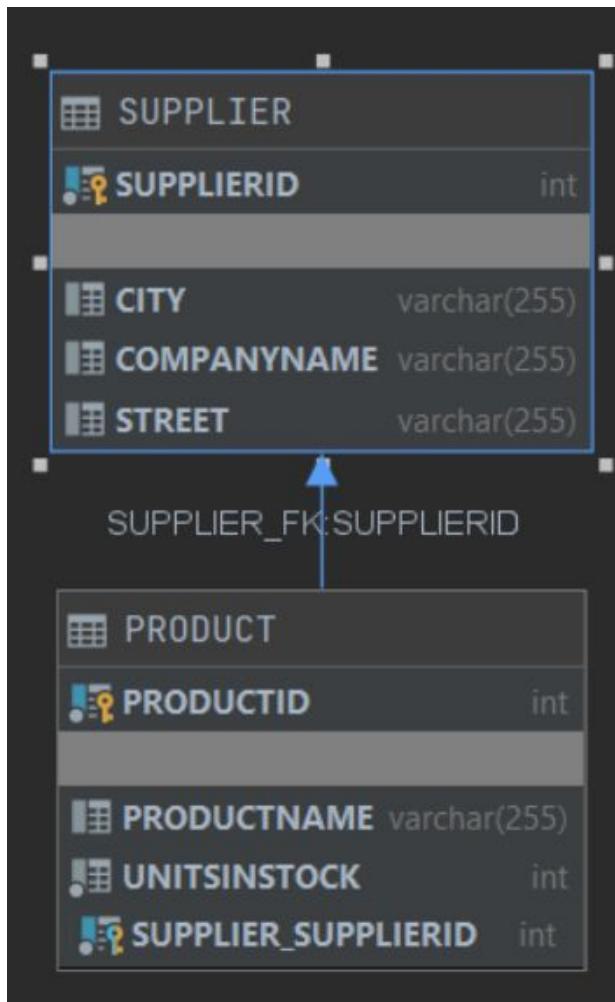
## Klasa sterująca Main - stworzenie i dodanie dostawcy do bazy danych

```
▶  public static void main(final String[] args) throws Exception {  
  /*  
   Scanner scanner = new Scanner(System.in);  
   System.out.println("Podaj nazwę produktu: ");  
   String productName = scanner.nextLine();  
   System.out.println("Podaj ilość: ");  
   int unitsInStock = scanner.nextInt();  
   Product product = new Product(productName, unitsInStock);  
  */  
  final Session session = getSession();  
  
  //Product product = new Product("Jabłka", 100);  
  Transaction tx = session.beginTransaction();  
  
  Product product = session.get(Product.class, serializable: 1);  
  Supplier supplier = new Supplier(CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków");  
  product.setSupplier(supplier);  
  session.save(supplier);  
  session.save(product);  
  
  session.save(product);  
  tx.commit();  
  //session.close();
```

## Wynik działania programu

```
Run: Main ×  
▶  Supplier * select  
    supplier0_.SupplierID as supplier1_1_,  
    supplier0_.City as city2_1_,  
    supplier0_.CompanyName as companyn3_1_,  
    supplier0_.Street as street4_1_  
  from  
    Supplier supplier0_  
  Supplier{SupplierID=3, CompanyName='Tesco', Street='Kapelanka', City='Kraków'  
executing: from Product  
Hibernate:  
  /*  
from  
  Product * select  
    product0_.ProductID as product1_0_,  
    product0_.ProductName as productn2_0_,  
    product0_.UnitsInStock as unitsins3_0_,  
    product0_.supplier_SupplierID as supplier4_0_  
  from  
    Product product0_  
  Product{ProductID=1, ProductName='Gruszka', UnitsInStock=2}  
  Product{ProductID=2, ProductName='Melon', UnitsInStock=4}  
  
Process finished with exit code 0
```

## Schemat bazy danych



## Zawartość bazy danych

Database Consoles > jdbcderby://127.0.0.1/BKordekIPA > console [jdbcderby://127.0.0.1/BKordekIPA]

Project: BKordekJPAPractice C:\Users\bart0\IdeaProject

Structure

Services

Tx Persistence

Output APP.PRODUCT

PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIER_SUPPLIERID
1	1 Gruszka	2	3
2	2 Melon	4	<null>

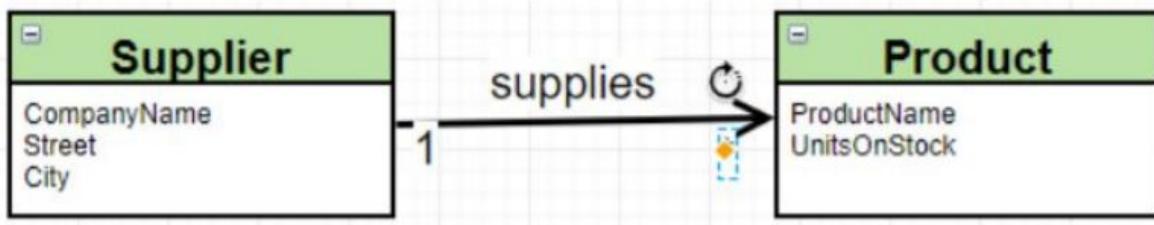
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** Shows the project structure for "BKordekJPAPractice" located at "C:\Users\barto\IdeaProject". It includes a "src" folder containing "Main.java", "Product.java", "Supplier.java", and "BkordekJPAPractice.iml".
- Database Tool Window:** Shows a connection to "jdbcderby://127.0.0.1/BKordekJPAPractice". A query "SELECT \* FROM SUPPLIER" is being run.
- Results Table:** Displays the data from the "SUPPLIER" table:

SUPPLIERID	CITY	COMPANYNAME	STREET	
1	3	Kraków	Tesco	Kapelanka
- Services Tool Window:** Shows a transaction named "Tx" and a persistence layer named "jdbcderby://127.0.0.1/BKordekJPAPractice".

## V. Odwróć relacje zgodnie z poniższym schematem

ODWRÓCONA RELACJA

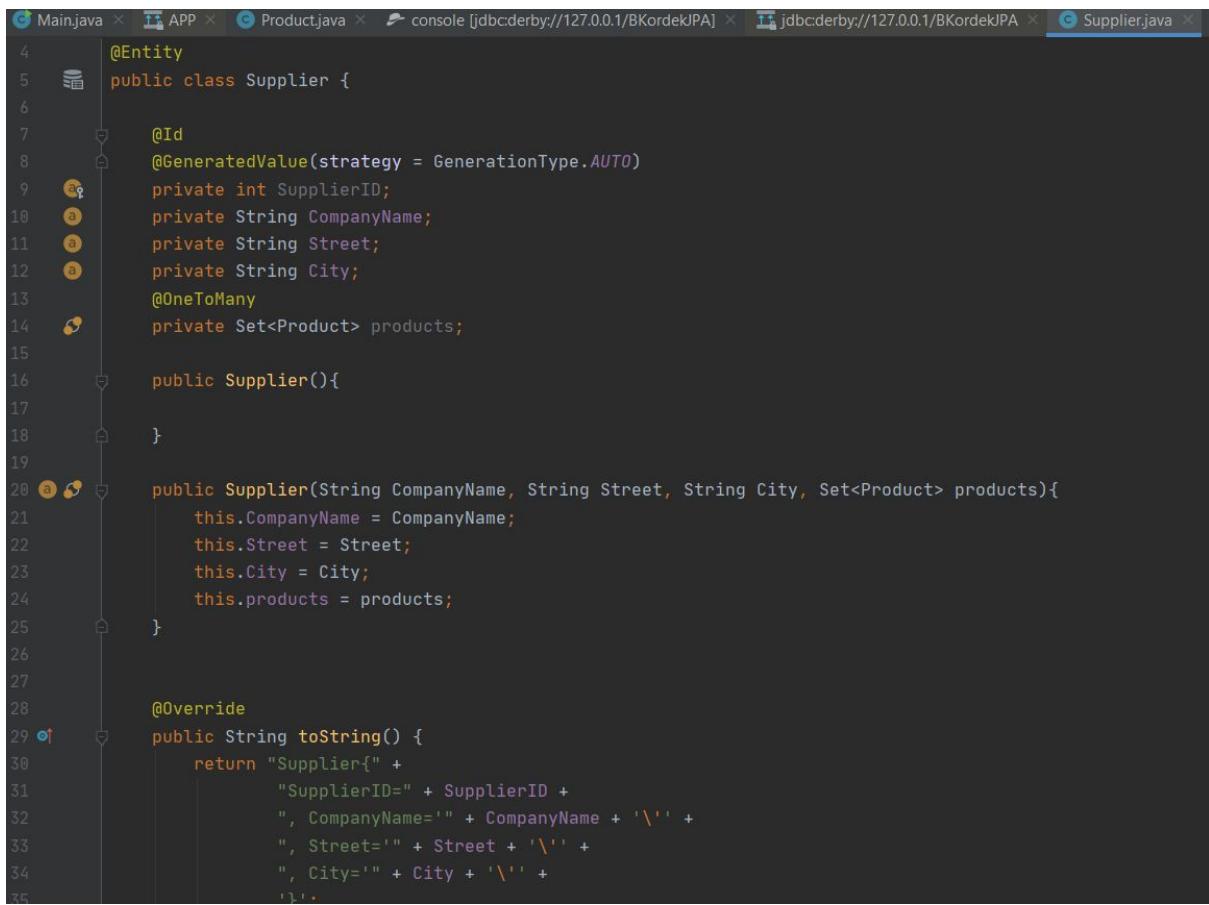


- a) Wariant z tabelą łącznikową

Klasa *Product*

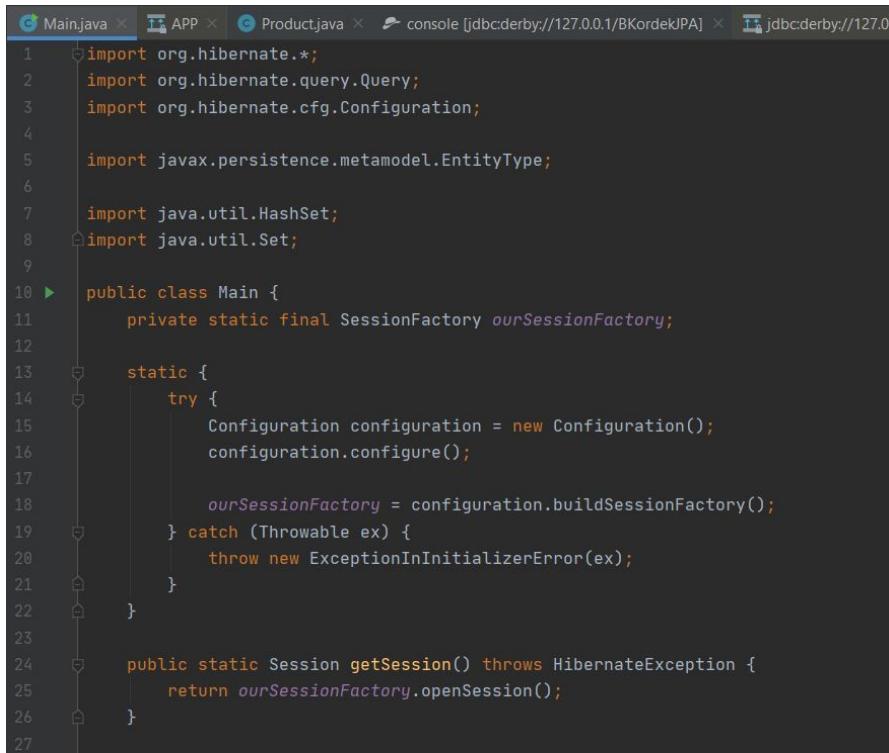
```
hibernate.cfg.xml x Main.java x APP x Product.java x console [jdbc:derby]
1 import javax.persistence.*;
2
3 @Entity
4 public class Product {
5
6     @Id
7     @GeneratedValue(strategy = GenerationType.AUTO)
8     private int ProductID;
9     private String ProductName;
10    private int UnitsInStock;
11
12    public Product(){
13    }
14
15
16    public Product(String ProductName, int UnitsInStock){
17        this.ProductName=ProductName;
18        this.UnitsInStock=UnitsInStock;
19    }
20
21
22    @Override
23    public String toString() {
24        return "Product{" +
25            "ProductID=" + ProductID +
26            ", ProductName='" + ProductName + '\'' +
27            ", UnitsInStock=" + UnitsInStock +
28            '}';
29    }
30 }
```

## Klasa Supplier



```
4     @Entity
5     public class Supplier {
6
7         @Id
8         @GeneratedValue(strategy = GenerationType.AUTO)
9         private int SupplierID;
10        private String CompanyName;
11        private String Street;
12        private String City;
13        @OneToMany
14        private Set<Product> products;
15
16        public Supplier(){}
17
18    }
19
20        public Supplier(String CompanyName, String Street, String City, Set<Product> products){
21            this.CompanyName = CompanyName;
22            this.Street = Street;
23            this.City = City;
24            this.products = products;
25        }
26
27
28        @Override
29        public String toString() {
30            return "Supplier{" +
31                "SupplierID=" + SupplierID +
32                ", CompanyName='" + CompanyName + '\'' +
33                ", Street='" + Street + '\'' +
34                ", City='" + City + '\'\' +
35                '}';
36    }
```

## Klasa sterująca Main - stworzenie kilku produktów i dostarczenie ich przed dostawcę



```
1     import org.hibernate.*;
2     import org.hibernate.query.Query;
3     import org.hibernate.cfg.Configuration;
4
5     import javax.persistence.metamodel.EntityType;
6
7     import java.util.HashSet;
8     import java.util.Set;
9
10    public class Main {
11        private static final SessionFactory ourSessionFactory;
12
13        static {
14            try {
15                Configuration configuration = new Configuration();
16                configuration.configure();
17
18                ourSessionFactory = configuration.buildSessionFactory();
19            } catch (Throwable ex) {
20                throw new ExceptionInInitializerError(ex);
21            }
22        }
23
24        public static Session getSession() throws HibernateException {
25            return ourSessionFactory.openSession();
26        }
27    }
```

```
28 ► Main.java × APP × Product.java × console [jdbc:derby://127.0.0.1/BKordekJPA] × jdbc:derby://127.0.0.1/BKordekJPA × Supplier.java ×
29     public static void main(final String[] args) throws Exception {
30         final Session session = getSession();
31
32         Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20);
33         Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10);
34         Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50);
35
36         Set<Product> productSetForSupplier1 = new HashSet<>();
37         productSetForSupplier1.add(product1);
38         productSetForSupplier1.add(product2);
39         Set<Product> productSetForSupplier2 = new HashSet<>();
40         productSetForSupplier2.add(product3);
41
42         Supplier supplier1 = new Supplier( CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków", productSetForSupplier1);
43         Supplier supplier2 = new Supplier( CompanyName: "Żabka", Street: "Sołtysowka", City: "Kraków", productSetForSupplier2);
44
45         Transaction tx = session.beginTransaction();
46
47         session.save(product1);
48         session.save(product2);
49         session.save(product3);
50
51         session.save(supplier1);
52         session.save(supplier2);
53
54         tx.commit();
55         //session.close();
56
57     try {
58         System.out.println("querying all the managed entities...");
59         //session.close();
60
61     try {
62         System.out.println("querying all the managed entities...");
63         final Metamodel metamodel = session.getSessionFactory().getMetamodel();
64         for (EntityType<?> entityType : metamodel.getEntities()) {
65             final String entityName = entityType.getName();
66             final Query query = session.createQuery( s: "from " + entityName);
67             System.out.println("executing: " + query.getQueryString());
68             for (Object o : query.list()) {
69                 System.out.println(" " + o);
70             }
71         }
72     } finally {
73         session.close();
74     }
75 }
```

Uruchomienie programu

```
next value for hibernate_sequence
Hibernate:
    /* insert Product
     * \ insert
     into
        Product
        (ProductName, UnitsInStock, ProductID)
    values
        (?, ?, ?)

Hibernate:
    /* insert Product
     * \ insert
     into
        Product
        (ProductName, UnitsInStock, ProductID)
    values
        (?, ?, ?)

Hibernate:
    /* insert Product
     * \ insert
     into
        Product
        (ProductName, UnitsInStock, ProductID)
    values
        (?, ?, ?)

Hibernate:
    /* insert Supplier
     * \ insert
     into
```

```
/* insert collection
   row Supplier.products *L insert
   into
      Supplier_Product
      (Supplier_SupplierID, products_ProductID)
   values
      (?, ?)

Hibernate:
/* insert collection
   row Supplier.products *L insert
   into
      Supplier_Product
      (Supplier_SupplierID, products_ProductID)
   values
      (?, ?)

Hibernate:
/* insert collection
   row Supplier.products *L insert
   into
      Supplier_Product
      (Supplier_SupplierID, products_ProductID)
   values
      (?, ?)

querying all the managed entities...
executing: from Supplier
Hibernate:
/*
from
   Supplier *L select
```

```
Run: Main ×
executing: from Supplier
Hibernate:
    /*
from
    Supplier *| select
        supplier0_.SupplierID as supplier1_1_,
        supplier0_.City as city2_1_,
        supplier0_.CompanyName as companyn3_1_,
        supplier0_.Street as street4_1_
from
    Supplier supplier0_
Supplier{SupplierID=4, CompanyName='Tesco', Street='Kapelanka', City='Kraków'}
Supplier{SupplierID=5, CompanyName='Żabka', Street='Sołtysowka', City='Kraków'}
executing: from Product
Hibernate:
    /*
from
    Product *| select
        product0_.ProductID as product1_0_,
        product0_.ProductName as productn2_0_,
        product0_.UnitsInStock as unitsins3_0_
from
    Product product0_
Product{ProductID=1, ProductName='Frytki', UnitsInStock=20}
Product{ProductID=2, ProductName='Pizza', UnitsInStock=10}
Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50}

Process finished with exit code 0
```

## Schemat bazy danych

jdbc:derby://127.0.0.1/BKordekJPA 1 of 11

APP

PRODUCT

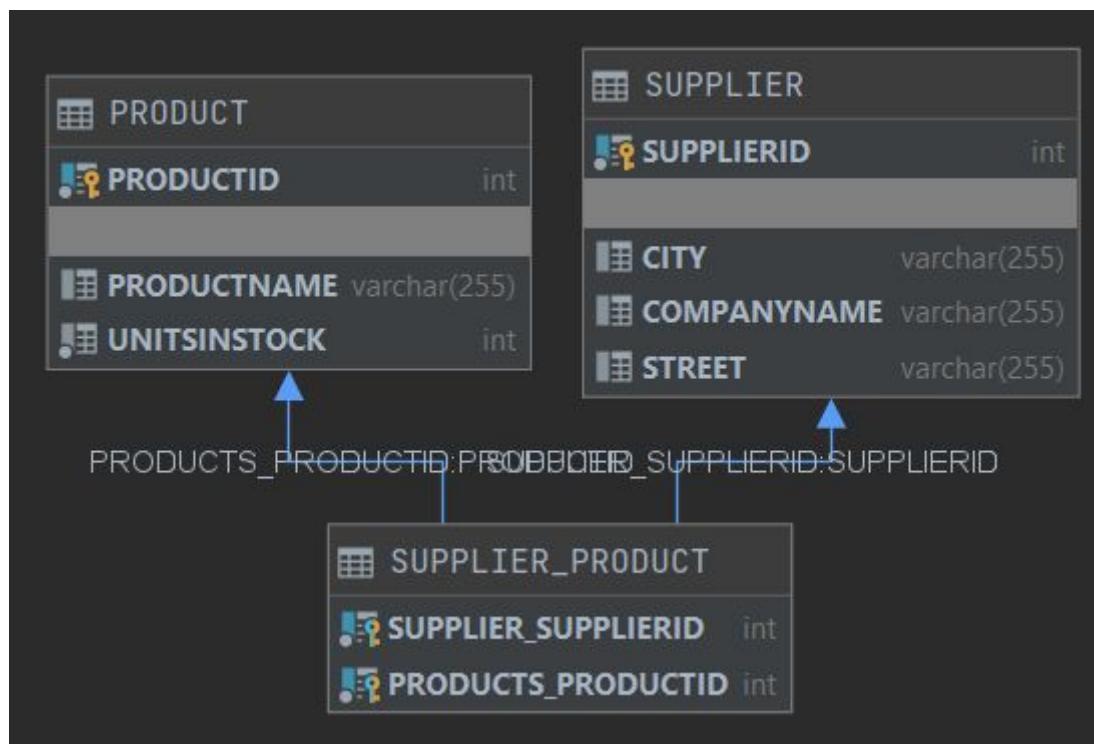
- PRODUCTID INTEGER
- PRODUCTNAME VARCHAR(255)
- UNITSINSTOCK INTEGER
- SQL201231154134030 (PRODUCTID)
- SQL201231154134030 (PRODUCTID) UNIQUE

SUPPLIER

- SUPPLIERID INTEGER
- CITY VARCHAR(255)
- COMPANYNAME VARCHAR(255)
- STREET VARCHAR(255)
- SQL201231154134060 (SUPPLIERID)
- SQL201231154134060 (SUPPLIERID) UNIQUE

SUPPLIER\_PRODUCT

- SUPPLIER\_SUPPLIERID INTEGER
- PRODUCTS\_PRODUCTID INTEGER
- SQL201231154134090 (SUPPLIER\_SUPPLIERID, PRODUCTS\_PRODUCTID)
- SQL201231154134110 (PRODUCTS\_PRODUCTID)
- FKHG38RTWWPUUXDYN0SKOF2WOU5 (SUPPLIER\_SUPPLIERID) → SUPPLIER (SUPPLIERID)
- FKSA8NJ74B82QQBBHDF94JLB28N (PRODUCTS\_PRODUCTID) → PRODUCT (PRODUCTID)
- SQL201231154134090 (SUPPLIER\_SUPPLIERID, PRODUCTS\_PRODUCTID) UNIQUE
- SQL201231154134110 (PRODUCTS\_PRODUCTID) UNIQUE
- SQL201231154134130 (PRODUCTS\_PRODUCTID)
- SQL201231154134140 (SUPPLIER\_SUPPLIERID)



### Zawartość tabel

APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]			
	PRODUCTID	PRODUCTNAME	UNITSINSTOCK
1	1	Frytki	20
2	2	Pizza	10
3	3	Chipsy	50

APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA]				
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Tesco	Kapelanka
2	5	Kraków	Żabka	Sołtysówka

The screenshot shows a database interface with a table named APP.SUPPLIER\_PRODUCT. The table has two columns: SUPPLIER\_SUPPLIERID and PRODUCTS\_PRODUCTID. The data consists of three rows:

	SUPPLIER_SUPPLIERID	PRODUCTS_PRODUCTID
1	4	1
2	4	2
3	5	3

Jak widać powyżej, została stworzona tabela łącznikowa *SUPPLIER\_PRODUCT* zawierająca klucze obce *SupplierID* i przyporządkowane *ProductID*.

b) Wariant bez tabeli łącznikowej

Klasa *Product*

```
1 import javax.persistence.*;
2
3 @Entity
4 public class Product {
5
6     @Id
7     @GeneratedValue(strategy = GenerationType.AUTO)
8     private int ProductID;
9     private String ProductName;
10    private int UnitsInStock;
11
12    public Product(){
13    }
14
15    public Product(String ProductName, int UnitsInStock){
16        this.ProductName=ProductName;
17        this.UnitsInStock=UnitsInStock;
18    }
19
20
21    @Override
22    public String toString() {
23        return "Product{" +
24            "ProductID=" + ProductID +
25            ", ProductName='" + ProductName + '\'' +
26            ", UnitsInStock=" + UnitsInStock +
27            '}';
28    }
29}
30}
```

## Klasa *Supplier*

```
5     @Entity
6     public class Supplier {
7
8         @Id
9         @GeneratedValue(strategy = GenerationType.AUTO)
10        private int SupplierID;
11        private String CompanyName;
12        private String Street;
13        private String City;
14        @OneToMany
15        @JoinColumn(name="SUPPLIER_FK")
16        private Set<Product> products;
17
18        public Supplier(){
19
20    }
21
22        public Supplier(String CompanyName, String Street, String City, Set<Product> products){
23            this.CompanyName = CompanyName;
24            this.Street = Street;
25            this.City = City;
26            this.products = products;
27        }
28
29
30        @Override
31        public String toString() {
32            return "Supplier{" +
33                    "SupplierID=" + SupplierID +
34                    ", CompanyName='" + CompanyName + '\'' +
35                    ", Street='" + Street + '\'' +
36                    ", City='" + City + '\''
37        }
38
39
40
41
42
```

Klasa sterująca *Main* - stworzenie kilku produktów i dostarczenie ich przed dostawcę

```
1  import org.hibernate.*;  
2  import org.hibernate.query.Query;  
3  import org.hibernate.cfg.Configuration;  
4  
5  import javax.persistence.metamodel.EntityType;  
6  
7  import java.util.HashSet;  
8  import java.util.Set;  
9  
10 > public class Main {  
11     private static final SessionFactory ourSessionFactory;  
12  
13     static {  
14         try {  
15             Configuration configuration = new Configuration();  
16             configuration.configure();  
17  
18             ourSessionFactory = configuration.buildSessionFactory();  
19         } catch (Throwable ex) {  
20             throw new ExceptionInInitializerError(ex);  
21         }  
22     }  
23  
24     public static Session getSession() throws HibernateException {  
25         return ourSessionFactory.openSession();  
26     }  
27  
28 >     public static void main(final String[] args) throws Exception {  
29         final Session session = getSession();  
30  
31         Product product1 = new Product(ProductName: "Frytki", UnitsInStock: 20);
```

```
27
28 ► ↴ public static void main(final String[] args) throws Exception {
29     final Session session = getSession();
30
31     Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20 );
32     Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10 );
33     Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50 );
34
35     Set<Product> productSetForSupplier1 = new HashSet<>();
36     productSetForSupplier1.add(product1);
37     productSetForSupplier1.add(product2);
38     Set<Product> productSetForSupplier2 = new HashSet<>();
39     productSetForSupplier2.add(product3);
40
41     Supplier supplier1 = new Supplier( CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków", productSetForSupplier1 );
42     Supplier supplier2 = new Supplier( CompanyName: "Żabka", Street: "Sołtysowka", City: "Kraków", productSetForSupplier2 );
43
44     Transaction tx = session.beginTransaction();
45
46     session.save(product1);
47     session.save(product2);
48     session.save(product3);
49
50     session.save(supplier1);
51     session.save(supplier2);
52
53     tx.commit();
54     //session.close();
55
56     try {
57         System.out.println("querying all the managed entities...");
58
59         try {
60             final Metamodel metamodel = session.getSessionFactory().getMetamodel();
61             for (EntityType<?> entityType : metamodel.getEntities()) {
62                 final String entityName = entityType.getName();
63                 final Query query = session.createQuery( s: "from " + entityName );
64                 System.out.println("executing: " + query.getQueryString());
65                 for (Object o : query.list()) {
66                     System.out.println(" " + o);
67                 }
68             }
69         } finally {
70             session.close();
71         }
72     }
```

Wynik działania programu

```
↑ Hibernate:  
↓     /* insert Product  
      *l insert  
      into  
         Product  
         (ProductName, UnitsInStock, ProductID)  
      values  
         (?, ?, ?)  
Hibernate:  
      /* insert Product  
      *l insert  
      into  
         Product  
         (ProductName, UnitsInStock, ProductID)  
      values  
         (?, ?, ?)  
Hibernate:  
      /* insert Product  
      *l insert  
      into  
         Product  
         (ProductName, UnitsInStock, ProductID)  
      values  
         (?, ?, ?)  
Hibernate:  
      /* insert Supplier  
      *l insert  
      into  
         Supplier
```

```
values
      (?, ?, ?, ?)

Hibernate:
/* insert Supplier
 * \ insert
into
    Supplier
    (City, CompanyName, Street, SupplierID)
values
      (?, ?, ?, ?)

Hibernate:
/* create one-to-many row Supplier.products */ update
    Product
set
    SUPPLIER_FK=?
where
    ProductID=?

Hibernate:
/* create one-to-many row Supplier.products */ update
    Product
set
    SUPPLIER_FK=?
where
    ProductID=?

Hibernate:
/* create one-to-many row Supplier.products */ update
    Product
set
    SUPPLIER_FK=?
```

```
↑ executing: from Supplier
↓ Hibernate:
  /*
from
  Supplier * select
    supplier0_.SupplierID as supplier1_1_,
    supplier0_.City as city2_1_,
    supplier0_.CompanyName as companyn3_1_,
    supplier0_.Street as street4_1_
  from
    Supplier supplier0_
Supplier{SupplierID=4, CompanyName='Tesco', Street='Kapelanka', City='Kraków'}
Supplier{SupplierID=5, CompanyName='Żabka', Street='Sołtysowka', City='Kraków'}
executing: from Product
Hibernate:
  /*
from
  Product * select
    product0_.ProductID as producti1_0_,
    product0_.ProductName as productn2_0_,
    product0_.UnitsInStock as unitsins3_0_
  from
    Product product0_
Product{ProductID=1, ProductName='Frytki', UnitsInStock=20}
Product{ProductID=2, ProductName='Pizza', UnitsInStock=10}
Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50}

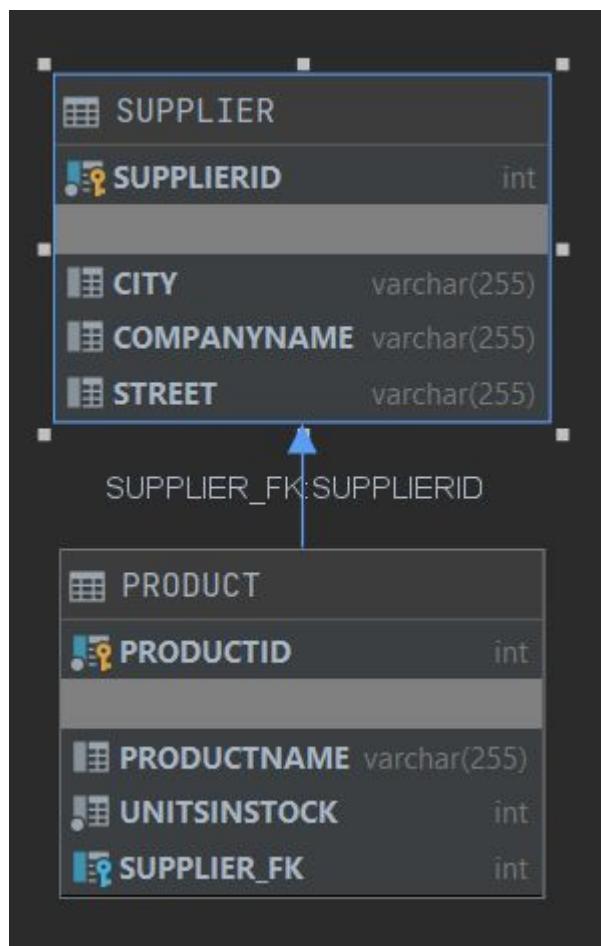
Process finished with exit code 0
```

## Schemat bazy danych

Database

The screenshot shows a database schema browser interface. At the top, there are standard toolbar icons for creating, deleting, and modifying tables. Below the toolbar, the connection information is displayed: `jdbc:derby://127.0.0.1/BKordekJPA` and `1 of 11`. The main pane displays two tables under the schema `APP`:

- PRODUCT**: Contains columns `PRODUCTID` (INTEGER, primary key), `PRODUCTNAME` (VARCHAR(255)), `UNITSINSTOCK` (INTEGER), `SUPPLIER_FK` (INTEGER), and three constraints: `SQL201231165755990` (primary key constraint on `PRODUCTID`), `FKEURY2HXL2J8URLKMW36585TKR` (foreign key constraint from `SUPPLIER_FK` to `SUPPLIER(SUPPLIERID)`), and `SQL201231165755990` (unique constraint on `PRODUCTID`). It also contains a comment `i`.
- SUPPLIER**: Contains columns `SUPPLIERID` (INTEGER, primary key), `CITY` (VARCHAR(255)), `COMPANYNAME` (VARCHAR(255)), `STREET` (VARCHAR(255)), and three constraints: `SQL201231165756010` (primary key constraint on `SUPPLIERID`), `SQL201231165756010` (unique constraint on `SUPPLIERID`), and `i`.



### Zawartość tabel

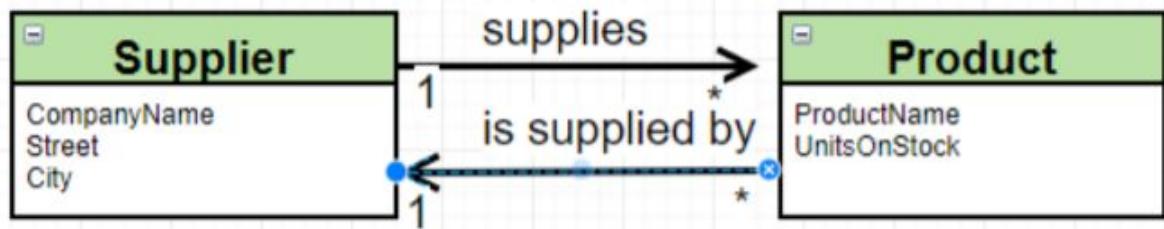
APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]				
1	1	Frytki	20	4
2	2	Pizza	10	4
3	3	Chipsy	50	5

APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]					APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA]			
1	4	Kraków	Tesco		1	4	Kapelanka	
2	5	Kraków	Żabka		2	5	Sołtysowka	

Jak widać powyżej zostały stworzone tylko dwie tabele, gdzie w tabeli *PRODUCT* znajduje się klucz obcy *SUPPLIER\_FK*, będącym ID Dostawcy.

## VI. Zamodeluj relacje dwustronną jak poniżej



Klasa *Product*

```
Product.java X Main.java X APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] X APP.SUPPLIER [jdbc:derby/]

1  @Entity
2  public class Product {
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.AUTO)
6      private int ProductID;
7      private String ProductName;
8      private int UnitsInStock;
9
10     @ManyToOne
11     @JoinColumn(name="SUPPLIER_FK")
12     private Supplier supplier;
13
14     public Product(){
15
16     }
17
18     public Product(String ProductName, int UnitsInStock, Supplier supplier){
19         this.ProductName=ProductName;
20         this.UnitsInStock=UnitsInStock;
21         this.supplier=supplier;
22     }
23
24
25     @Override
26     public String toString() {
27         return "Product{" +
28             "ProductID=" + ProductID +
29             ", ProductName='" + ProductName + '\'' +
30             ", UnitsInStock=" + UnitsInStock +
31             '}';
32     }
}
```

## Klasa Supplier

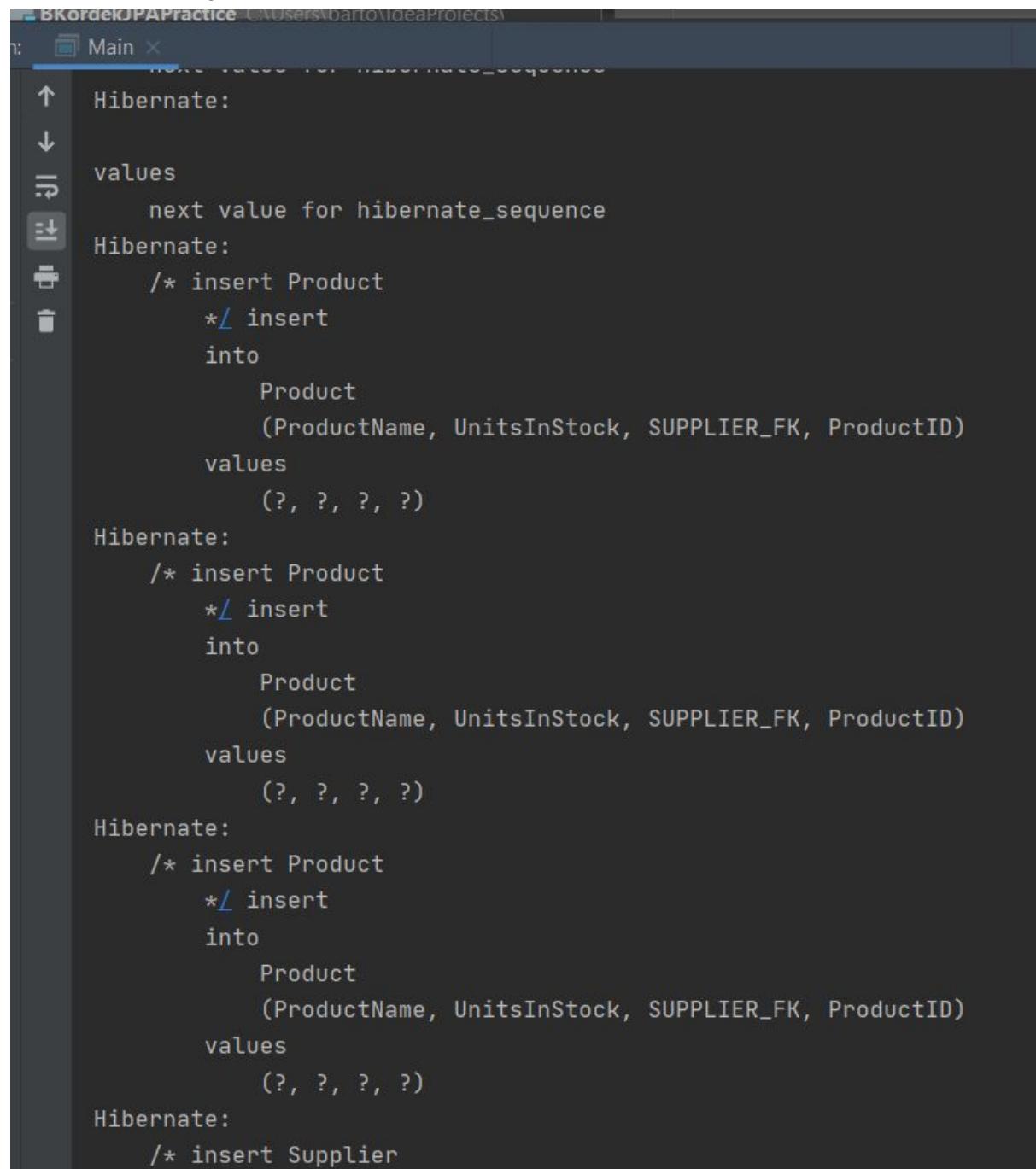
```
ct.java x Mainjava x APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] x APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] x APP x Supplier.java x
5     @Entity
6     public class Supplier {
7
8         @Id
9         @GeneratedValue(strategy = GenerationType.AUTO)
10        private int SupplierID;
11        private String CompanyName;
12        private String Street;
13        private String City;
14
15        @OneToMany
16        @JoinColumn(name="SUPPLIER_FK")
17        private Set<Product> products;
18
19        public Supplier(){
20
21
22        public Supplier(String CompanyName, String Street, String City, Set<Product> products){
23            this.CompanyName = CompanyName;
24            this.Street = Street;
25            this.City = City;
26            this.products = products;
27
28
29
30        @Override
31        public String toString() {
32            return "Supplier{" +
33                "SupplierID=" + SupplierID +
34                ", CompanyName='" + CompanyName + '\'' +
35                ", Street='" + Street + '\'' +
```

Klasa sterująca *Main* - stworzenie produktów i dodanie ich do stworzonego dostawcy

```
it.java x C Main.java x APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] x APP.SUPPLIER [jdbc:derb
1 import org.hibernate.*;
2 import org.hibernate.query.Query;
3 import org.hibernate.cfg.Configuration;
4
5 import javax.persistence.metamodel.EntityType;
6
7 import java.util.HashSet;
8 import java.util.Set;
9
10 public class Main {
11     private static final SessionFactory ourSessionFactory;
12
13     static {
14         try {
15             Configuration configuration = new Configuration();
16             configuration.configure();
17
18             ourSessionFactory = configuration.buildSessionFactory();
19         } catch (Throwable ex) {
20             throw new ExceptionInInitializerError(ex);
21         }
22     }
23
24     public static Session getSession() throws HibernateException {
25         return ourSessionFactory.openSession();
26     }
27
28     public static void main(final String[] args) throws Exception {
29         final Session session = getSession();
30
31         Set<Product> productSetForSupplier1 = new HashSet<>();
```

```
src.java × Main.java × APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] × APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] × APP × Supplier.java ×
28 ► public static void main(final String[] args) throws Exception {
29     final Session session = getSession();
30
31     Set<Product> productSetForSupplier1 = new HashSet<>();
32     Set<Product> productSetForSupplier2 = new HashSet<>();
33
34     Supplier supplier1 = new Supplier( CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków", productSetForSupplier1 );
35     Supplier supplier2 = new Supplier( CompanyName: "Zabka", Street: "Sołtysówka", City: "Kraków", productSetForSupplier2 );
36
37     Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20, supplier1 );
38     Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10, supplier1 );
39     Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50, supplier2 );
40
41     productSetForSupplier1.add(product1);
42     productSetForSupplier1.add(product2);
43     productSetForSupplier2.add(product3);
44
45     Transaction tx = session.beginTransaction();
46
47     session.save(product1);
48     session.save(product2);
49     session.save(product3);
50
51     session.save(supplier1);
52     session.save(supplier2);
53
54     tx.commit();
55     //session.close();
56
57     try {
58         System.out.println("querying all the managed entities...");
59         final Metamodel metamodel = session.getSessionFactory().getMetamodel();
60         for (EntityType<?> entityType : metamodel.getEntities()) {
61             final String entityName = entityType.getName();
62             final Query query = session.createQuery( s: "from " + entityName );
63             System.out.println("executing: " + query.getQueryString());
64             for (Object o : query.list()) {
65                 System.out.println(" " + o);
66             }
67         }
68     } finally {
69         session.close();
70     }
71 }
72 }
```

## Uruchomienie programu



The screenshot shows a code editor window with a dark theme. The title bar reads "B Kordek JPA Practice C:\Users\barto\IdeaProjects\". The editor pane contains a file named "Main.sql". The code is a series of SQL statements generated by Hibernate, intended for an Oracle database. It includes several "Hibernate:" comments, each preceding a block of SQL. The first block sets the sequence value. Subsequent blocks insert data into the "Product" table, with each insert statement having five question mark placeholders for parameters. The final block inserts data into the "Supplier" table.

```
Hibernate:  
    next value for hibernate_sequence  
Hibernate:  
/* insert Product  
 *  
 * insert  
 into  
     Product  
     (ProductName, UnitsInStock, SUPPLIER_FK, ProductID)  
 values  
     (?, ?, ?, ?)  
Hibernate:  
/* insert Product  
 *  
 * insert  
 into  
     Product  
     (ProductName, UnitsInStock, SUPPLIER_FK, ProductID)  
 values  
     (?, ?, ?, ?)  
Hibernate:  
/* insert Product  
 *  
 * insert  
 into  
     Product  
     (ProductName, UnitsInStock, SUPPLIER_FK, ProductID)  
 values  
     (?, ?, ?, ?)  
Hibernate:  
/* insert Supplier
```

```
        (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
       *L insert
       into
           Supplier
           (City, CompanyName, Street, SupplierID)
       values
           (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
       *L insert
       into
           Supplier
           (City, CompanyName, Street, SupplierID)
       values
           (?, ?, ?, ?)
Hibernate:
    /* update
       Product *L update
           Product
           set
               ProductName=?,
               UnitsInStock=?,
               SUPPLIER_FK=?
           where
               ProductID=?
Hibernate:
    /* update
       Product *L update
```

```
        where
            ProductID=?

Hibernate:
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?
    where
        ProductID=?

Hibernate:
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?
    where
        ProductID=?

Hibernate:
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?
    where
        ProductID=?

querying all the managed entities...
executing: from Supplier
Hibernate:
/*
from
Supplier *L select
    supplier0_.SupplierID as supplier1_1_,
```

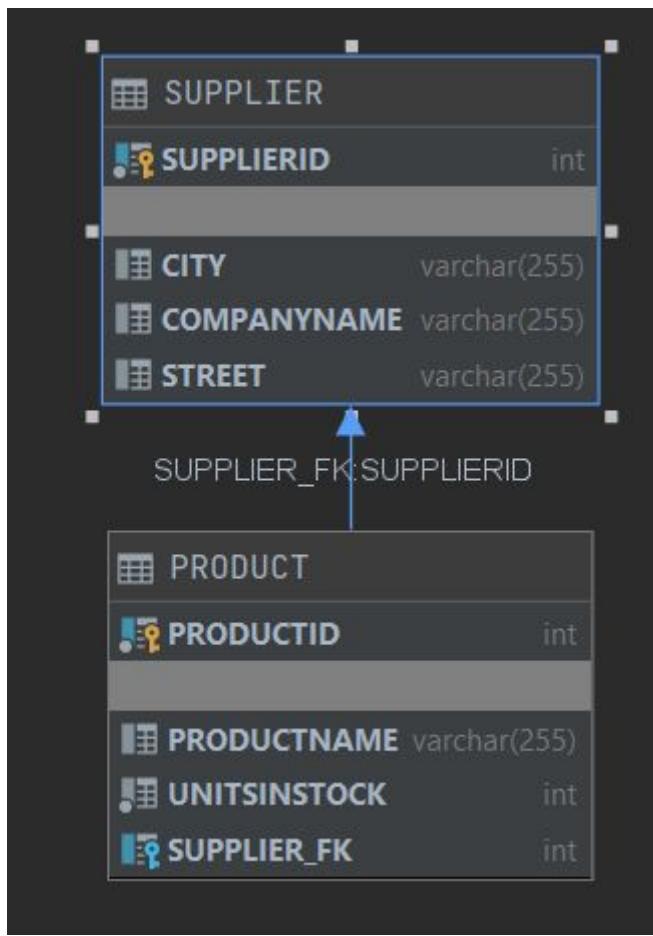
```
Run: Main ×
querying all the managed entities...
executing: from Supplier
Hibernate:
/*
from
    Supplier * select
        supplier0_.SupplierID as supplier1_1_,
        supplier0_.City as city2_1_,
        supplier0_.CompanyName as companyn3_1_,
        supplier0_.Street as street4_1_
    from
        Supplier supplier0_
Supplier{SupplierID=4, CompanyName='Tesco', Street='Kapelanka', City='Kraków'}
Supplier{SupplierID=5, CompanyName='Żabka', Street='Sołtysowka', City='Kraków'}
executing: from Product
Hibernate:
/*
from
    Product * select
        product0_.ProductID as product1_0_,
        product0_.ProductName as productn2_0_,
        product0_.UnitsInStock as unitsins3_0_,
        product0_.SUPPLIER_FK as supplier4_0_
    from
        Product product0_
Product{ProductID=1, ProductName='Frytki', UnitsInStock=20}
Product{ProductID=2, ProductName='Pizza', UnitsInStock=10}
Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50}

Process finished with exit code 0
```

## Schemat bazy danych

The screenshot shows the MySQL Workbench interface displaying the schema for the 'APP' database. The 'PRODUCT' table contains columns: PRODUCTID (primary key), PRODUCTNAME (VARCHAR(255)), UNITSINSTOCK (INTEGER), SUPPLIER\_FK (INTEGER), and three unique constraints: SQL201231173430440 (PRODUCTID), FKEURYHXL2J8URLKMW36585TKR (SUPPLIER\_FK) → SUPPLIER (SUPPLIERID), and SQL201231173430440 (PRODUCTID) UNIQUE. The 'SUPPLIER' table contains columns: SUPPLIERID (primary key), CITY (VARCHAR(255)), COMPANYNAME (VARCHAR(255)), STREET (VARCHAR(255)), and one unique constraint: SQL201231173430460 (SUPPLIERID) UNIQUE.

Table	Column	Type	Properties
PRODUCT	PRODUCTID	INTEGER	
	PRODUCTNAME	VARCHAR(255)	
	UNITSINSTOCK	INTEGER	
	SUPPLIER_FK	INTEGER	
	SQL201231173430440	(PRODUCTID)	
	FKEURYHXL2J8URLKMW36585TKR	(SUPPLIER_FK) → SUPPLIER (SUPPLIERID)	
	SQL201231173430440	(PRODUCTID) UNIQUE	
SUPPLIER	SUPPLIERID	INTEGER	
	CITY	VARCHAR(255)	
	COMPANYNAME	VARCHAR(255)	
	STREET	VARCHAR(255)	
	SQL201231173430460	(SUPPLIERID)	
	SQL201231173430460	(SUPPLIERID) UNIQUE	



### Zawartość tabel

	APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]	APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA]	APP [ ]	
Q < Filter Criteria >				
	PRODUCTID	PRODUCTNAME	SUPPLIER_FK	
1	1	Frytki	20	4
2	2	Pizza	10	4
3	3	Chipsy	50	5

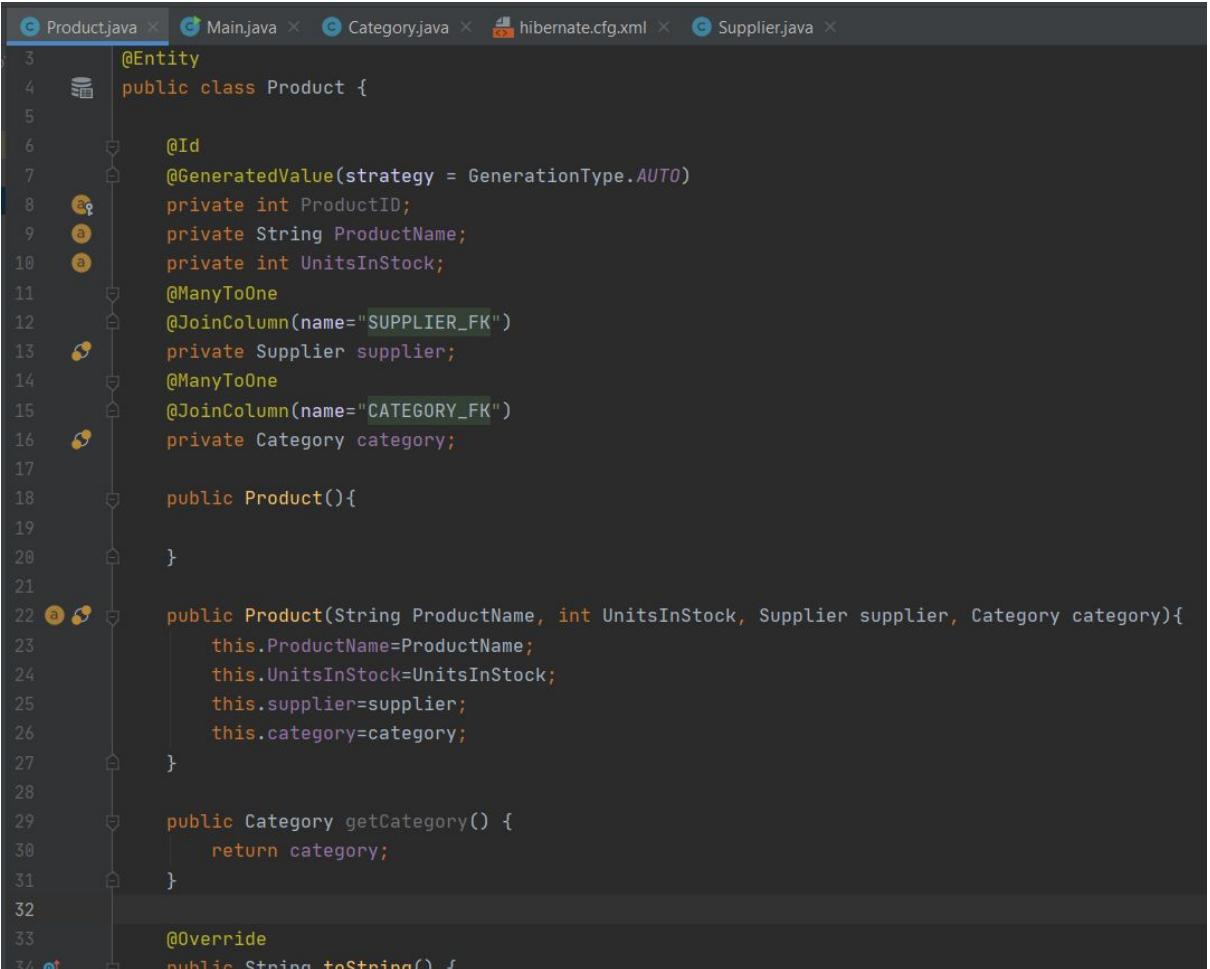
  

	APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]	APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA]	APP [ ]	
Q < Filter Criteria >				
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Tesco	Kapelanka
2	5	Kraków	Żabka	Sołtysowka

Z powyższego widać, że schemat jest taki sam jak w przypadku V. bez tabeli łącznikowej. Można natomiast dodać produkt przypisując do niego dostawcę i dostawcę przypisując do niego produkty.

## VII. Dodaj klasę Category z property int CategoryID, String Name oraz listą produktów List<Product> Product

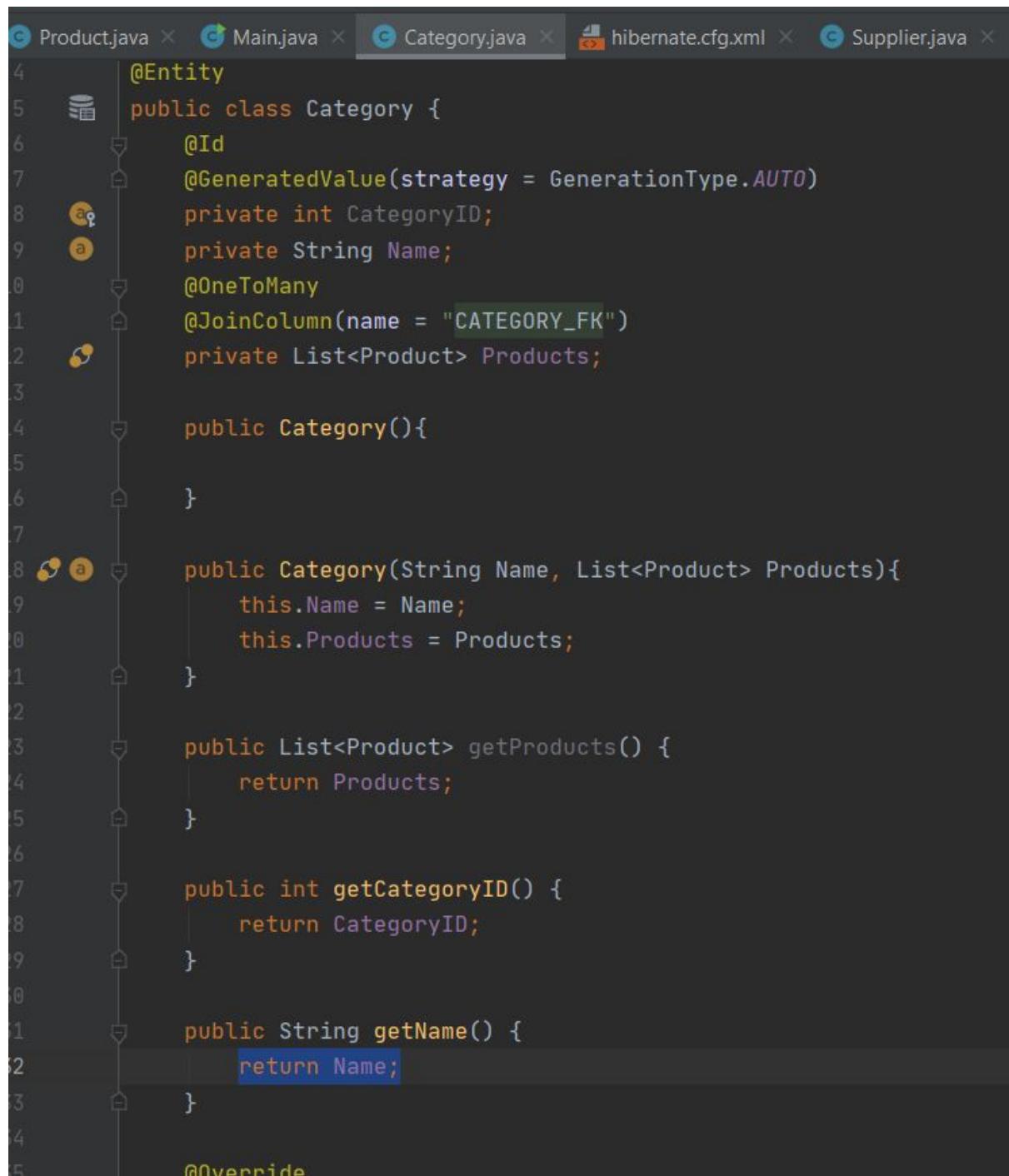
Klasa *Product* - modyfikacja produktów dodając wskazanie na kategorie do której należy



The screenshot shows a Java code editor with the Product.java file selected. The code defines a Product entity with attributes for ProductID (auto-generated), ProductName, UnitsInStock, supplier (ManyToOne), and category (ManyToOne). It includes a constructor, a getCategory() method, and an overridden toString() method.

```
3  @Entity
4  public class Product {
5
6      @Id
7      @GeneratedValue(strategy = GenerationType.AUTO)
8      private int ProductID;
9      private String ProductName;
10     private int UnitsInStock;
11
12     @ManyToOne
13     @JoinColumn(name="SUPPLIER_FK")
14     private Supplier supplier;
15
16     @ManyToOne
17     @JoinColumn(name="CATEGORY_FK")
18     private Category category;
19
20     public Product(){
21
22     }
23
24     public Product(String ProductName, int UnitsInStock, Supplier supplier, Category category){
25         this.ProductName=ProductName;
26         this.UnitsInStock=UnitsInStock;
27         this.supplier=supplier;
28         this.category=category;
29     }
30
31     public Category getCategory() {
32         return category;
33     }
34
35     @Override
36     public String toString() {
```

## Klasa Category

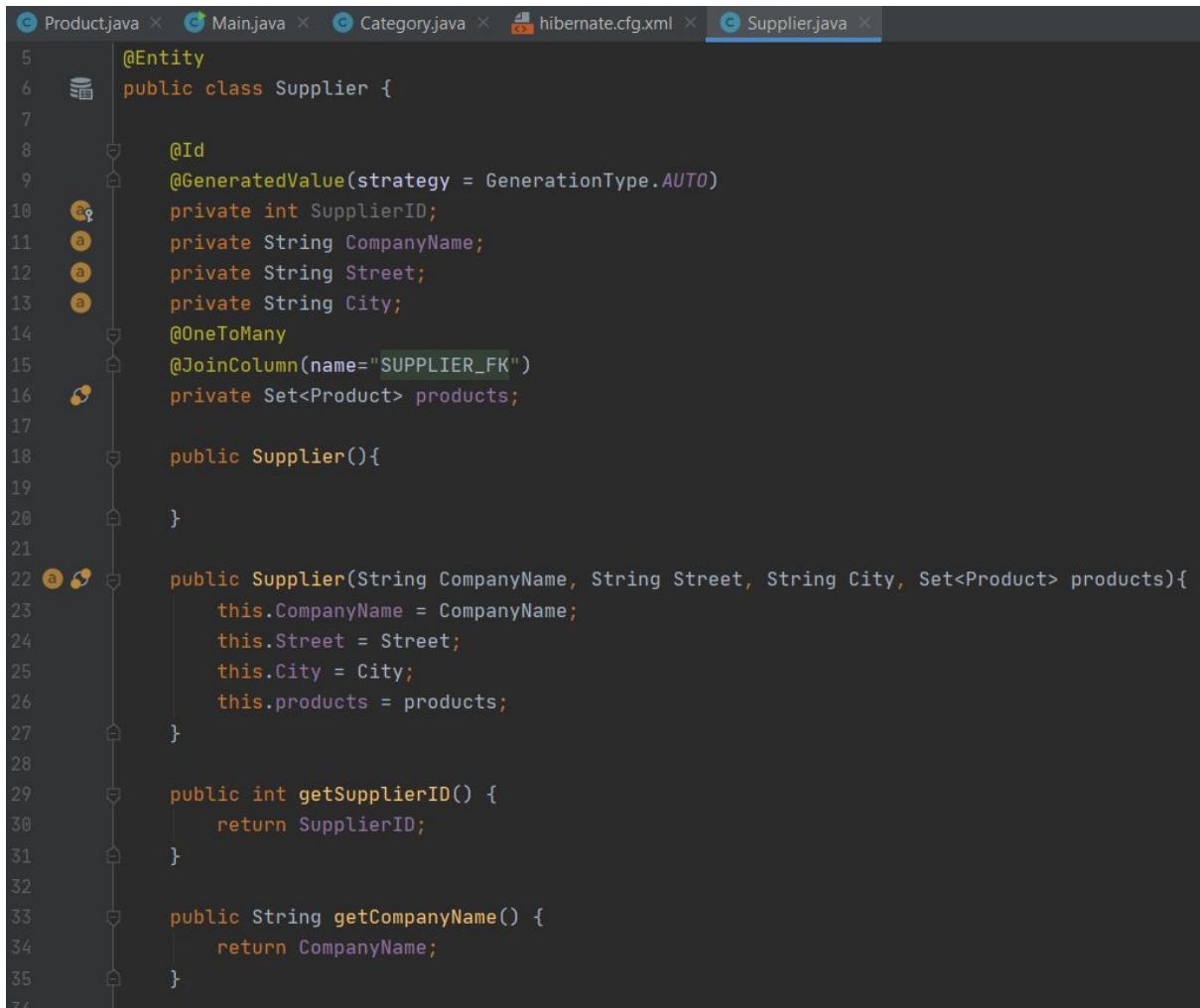


The screenshot shows a Java code editor with several tabs at the top: Product.java, Main.java, Category.java (which is the active tab), hibernate.cfg.xml, and Supplier.java. The Category.java tab contains the following code:

```
4     | @Entity
5  public class Category {
6      |     @Id
7      |     @GeneratedValue(strategy = GenerationType.AUTO)
8      |     private int CategoryID;
9      |     private String Name;
10     |     @OneToMany
11     |     @JoinColumn(name = "CATEGORY_FK")
12     |     private List<Product> Products;
13
14     |     public Category(){
15
16     |     }
17
18     |     public Category(String Name, List<Product> Products){
19         |         this.Name = Name;
20         |         this.Products = Products;
21     }
22
23     |     public List<Product> getProducts() {
24         |         return Products;
25     }
26
27     |     public int getCategoryID() {
28         |         return CategoryID;
29     }
30
31     |     public String getName() {
32         |         return Name;
33     }
34
35     |     @Override
```

The code defines a `Category` class with an `@Entity` annotation. It has an `@Id` field with `GenerationType.AUTO`, a `Name` field, and a `Products` field of type `List<Product>`. The `Products` field is annotated with `@OneToMany` and `@JoinColumn(name = "CATEGORY_FK")`. There are two constructors: one empty and one taking `Name` and `Products`. It also has `getProducts`, `getCategoryID`, and `getName` methods. The `getName` method is annotated with `@Override`.

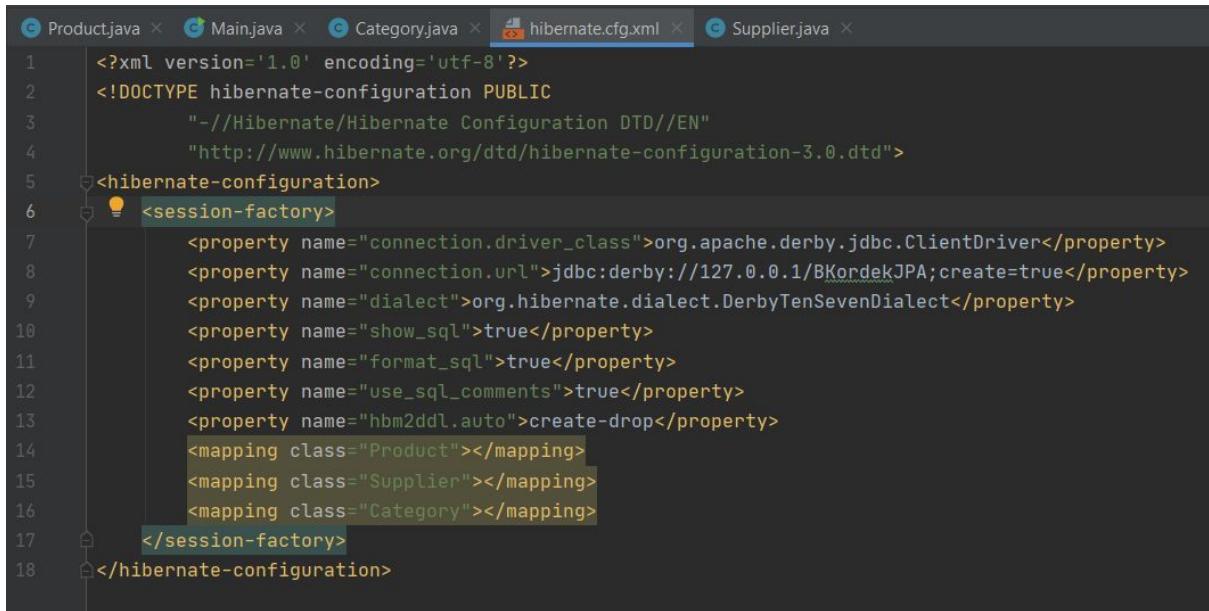
## Klasa Supplier



```
5     @Entity
6     public class Supplier {
7
8         @Id
9         @GeneratedValue(strategy = GenerationType.AUTO)
10        private int SupplierID;
11        private String CompanyName;
12        private String Street;
13        private String City;
14        @OneToMany
15        @JoinColumn(name="SUPPLIER_FK")
16        private Set<Product> products;
17
18        public Supplier(){
19
20        }
21
22        public Supplier(String CompanyName, String Street, String City, Set<Product> products){
23            this.CompanyName = CompanyName;
24            this.Street = Street;
25            this.City = City;
26            this.products = products;
27        }
28
29        public int getSupplierID() {
30            return SupplierID;
31        }
32
33        public String getCompanyName() {
34            return CompanyName;
35        }

```

## Modyfikacja Pliku konfiguracyjnego - dodanie klasy Category do mappingu



```
1      <?xml version='1.0' encoding='utf-8'?>
2      <!DOCTYPE hibernate-configuration PUBLIC
3          "-//Hibernate/Hibernate Configuration DTD//EN"
4          "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5
6      <hibernate-configuration>
7          <session-factory>
8              <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
9              <property name="connection.url">jdbc:derby:///127.0.0.1/BKordekJPA;create=true</property>
10             <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
11             <property name="show_sql">true</property>
12             <property name="format_sql">true</property>
13             <property name="use_sql_comments">true</property>
14             <property name="hbm2ddl.auto">create-drop</property>
15             <mapping class="Product"></mapping>
16             <mapping class="Supplier"></mapping>
17             <mapping class="Category"></mapping>
18
19         </session-factory>
20
21     </hibernate-configuration>
```

## Klasa sterująca Main - stworzenie kilku produktów i kilku kategorii

```
Product.java X Main.java X Category.java X hibernate.cfg.xml X Supplier.java X
28 }
29
30 public static void main(final String[] args) throws Exception {
31     final Session session = getSession();
32
33     Set<Product> productSetForSupplier1 = new HashSet<>();
34     Set<Product> productSetForSupplier2 = new HashSet<>();
35
36     List<Product> productListForCategory1 = new ArrayList<>();
37     List<Product> productListForCategory2 = new ArrayList<>();
38
39     Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
40     Category category2 = new Category( Name: "Przekaski", productListForCategory2);
41
42     Supplier supplier1 = new Supplier( CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków", productSetForSupplier1);
43     Supplier supplier2 = new Supplier( CompanyName: "Zabka", Street: "Słotysówka", City: "Kraków", productSetForSupplier2);
44
45     Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20, supplier1, category1);
46     Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10, supplier1, category1);
47     Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50, supplier2, category2);
48
49     productSetForSupplier1.add(product1);
50     productSetForSupplier1.add(product2);
51     productSetForSupplier2.add(product3);
52
53     productListForCategory1.add(product1);
54     productListForCategory1.add(product2);
55     productListForCategory2.add(product3);
56
57     Transaction tx = session.beginTransaction();
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72 try {
73     System.out.println("querying all the managed entities...");
74     final Metamodel metamodel = session.getSessionFactory().getMetamodel();
75     for (EntityType<?> entityType : metamodel.getEntities()) {
76         final String entityName = entityType.getName();
77         final Query query = session.createQuery( s: "from " + entityName);
78         System.out.println("executing: " + query.getQueryString());
79         for (Object o : query.list()) {
80             System.out.println(" " + o);
81         }
82     }
83 } finally {
84     session.close();
85 }
```

## Uruchomienie programu

```
next value for nextval_seq
▶ ↑ Hibernate:
  /* insert Product
   * \ insert
   into
     Product
     (ProductName, UnitsInStock, CATEGORY_FK, SUPPLIER_FK, ProductID)
   values
     (?, ?, ?, ?, ?)
Hibernate:
  /* insert Product
   * \ insert
   into
     Product
     (ProductName, UnitsInStock, CATEGORY_FK, SUPPLIER_FK, ProductID)
   values
     (?, ?, ?, ?, ?)
Hibernate:
  /* insert Product
   * \ insert
   into
     Product
     (ProductName, UnitsInStock, CATEGORY_FK, SUPPLIER_FK, ProductID)
   values
     (?, ?, ?, ?, ?)
Hibernate:
  /* insert Supplier
   * \ insert
   into
     Supplier
```

```
Run: Main ×  
      (? , ? , ? , ? , ? )  
Hibernate:  
    /* insert Supplier  
     *L insert  
     into  
       Supplier  
       (City, CompanyName, Street, SupplierID)  
     values  
       (? , ? , ? , ? )  
Hibernate:  
    /* insert Supplier  
     *L insert  
     into  
       Supplier  
       (City, CompanyName, Street, SupplierID)  
     values  
       (? , ? , ? , ? )  
Hibernate:  
    /* insert Category  
     *L insert  
     into  
       Category  
       (Name, CategoryID)  
     values  
       (? , ? )  
Hibernate:  
    /* insert Category  
     *L insert  
     into  
       Category
```

```
Run: Main ×  
Hibernate:  
    /* update  
        Product */ update  
            Product  
            set  
                ProductName=? ,  
                UnitsInStock=? ,  
                CATEGORY_FK=? ,  
                SUPPLIER_FK=?  
            where  
                ProductID=?  
  
Hibernate:  
    /* update  
        Product */ update  
            Product  
            set  
                ProductName=? ,  
                UnitsInStock=? ,  
                CATEGORY_FK=? ,  
                SUPPLIER_FK=?  
            where  
                ProductID=?  
  
Hibernate:  
    /* update  
        Product */ update  
            Product  
            set  
                ProductName=? ,  
                UnitsInStock=? ,  
                CATEGORY_FK=?
```

Run: Main ×

Hibernate:

```
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?  
where
        ProductID=?
```

Hibernate:

```
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?  
where
        ProductID=?
```

Hibernate:

```
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?  
where
        ProductID=?
```

Hibernate:

```
    /* create one-to-many row Category.Products */ update
        Product
    set
        CATEGORY_FK=?  
where
        ProductID=?
```

Hibernate:

```
RuM: Main >
querying all the managed entities...
executing: from Supplier
Hibernate:
/*
from
    Supplier */ select
        supplier0_.SupplierID as supplier1_2_,
        supplier0_.City as city2_2.,
        supplier0_.CompanyName as companyn3_2.,
        supplier0_.Street as street4_2_
from
    Supplier supplier0_
Supplier{SupplierID=4, CompanyName='Tesco', Street='Kapelanka', City='Kraków', products=[Product{ProductID=2, ProductName='Pizza', UnitsInStock=10, supplier=4 Tesco, category=6 Mrożonki}, Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, supplier=5 Żabka, category=7 Przekąski}]
executing: from Product
Hibernate:
/*
from
    Product */ select
        product0_.ProductID as product1_1.,
        product0_.ProductName as productn2_1.,
        product0_.UnitsInStock as unitins3_1.,
        product0_.CATEGORY_FK as category4_1.,
        product0_.SUPPLIER_FK as supplier5_1_
from
    Product product0_
Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, supplier=4 Tesco, category=6 Mrożonki}
Product{ProductID=2, ProductName='Pizza', UnitsInStock=10, supplier=4 Tesco, category=6 Mrożonki}
Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, supplier=5 Żabka, category=7 Przekąski}
executing: from Category
```

## Schemat bazy danych

Database

jdbc:derby://127.0.0.1/BKordekJPA 1 of 11

APP

CATEGORY

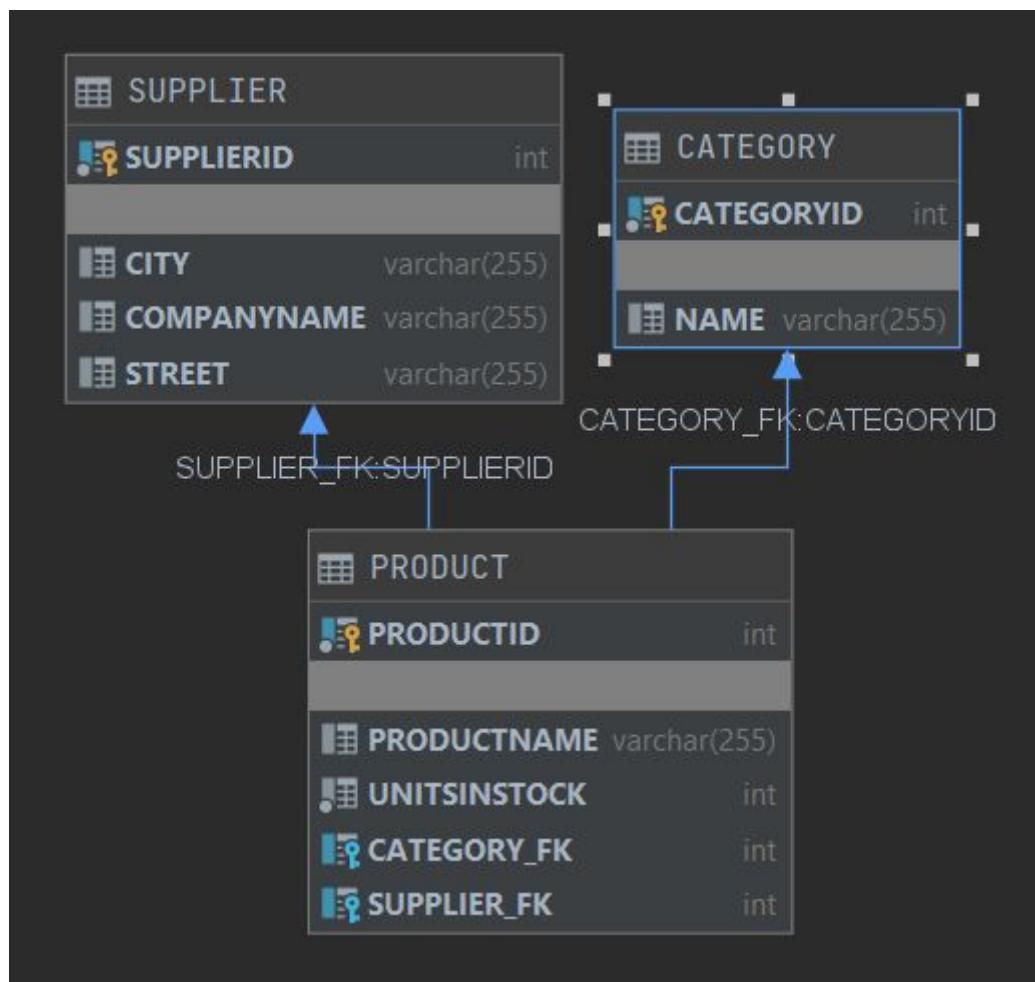
KEY	CATEGORYID	INTEGER
TEXT	NAME	VARCHAR(255)
KEY	SQL210102204607990	(CATEGORYID)
UNIQUE	SQL210102204607990	(CATEGORYID) UNIQUE

PRODUCT

KEY	PRODUCTID	INTEGER
TEXT	PRODUCTNAME	VARCHAR(255)
TEXT	UNITSINSTOCK	INTEGER
KEY	CATEGORY_FK	INTEGER
KEY	SUPPLIER_FK	INTEGER
KEY	SQL210102204608010	(PRODUCTID)
KEY	FKEURY2HXLJ8URLKMW36585TKR	(SUPPLIER_FK) → SUPPLIER (SUPPLIERID)
KEY	FKPURGJJ563MV2VAV0MGGDXEFD7	(CATEGORY_FK) → CATEGORY (CATEGORYID)
UNIQUE	SQL210102204608010	(PRODUCTID) UNIQUE
INDEX	SQL210102204608030	(CATEGORY_FK)
INDEX	SQL210102204608060	(SUPPLIER_FK)

SUPPLIER

KEY	SUPPLIERID	INTEGER
TEXT	CITY	VARCHAR(255)
TEXT	COMPANYNAME	VARCHAR(255)
TEXT	STREET	VARCHAR(255)
KEY	SQL210102204608020	(SUPPLIERID)
UNIQUE	SQL210102204608020	(SUPPLIERID) UNIQUE



### Zawartość tabel

APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA]	
<Filter Criteria>	
1	6 Mrożonki
2	7 Przekąski

APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]				
<Filter Criteria>				
1	1 Frytki	20	6	4
2	2 Pizza	10	6	4
3	3 Chipsy	50	7	5

APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA]

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Tesco	Kapelanka
2	5	Kraków	Żabka	Sołtysowka

Z powyższego widać, że do produktów zostały przypisane kategorie.

### Wydobycie produktów z wybranej kategorii oraz kategorię do której należy wybrany produkt

Modyfikacja pliku konfiguracyjnego - zmiana właściwości z *create-drop* na *update*

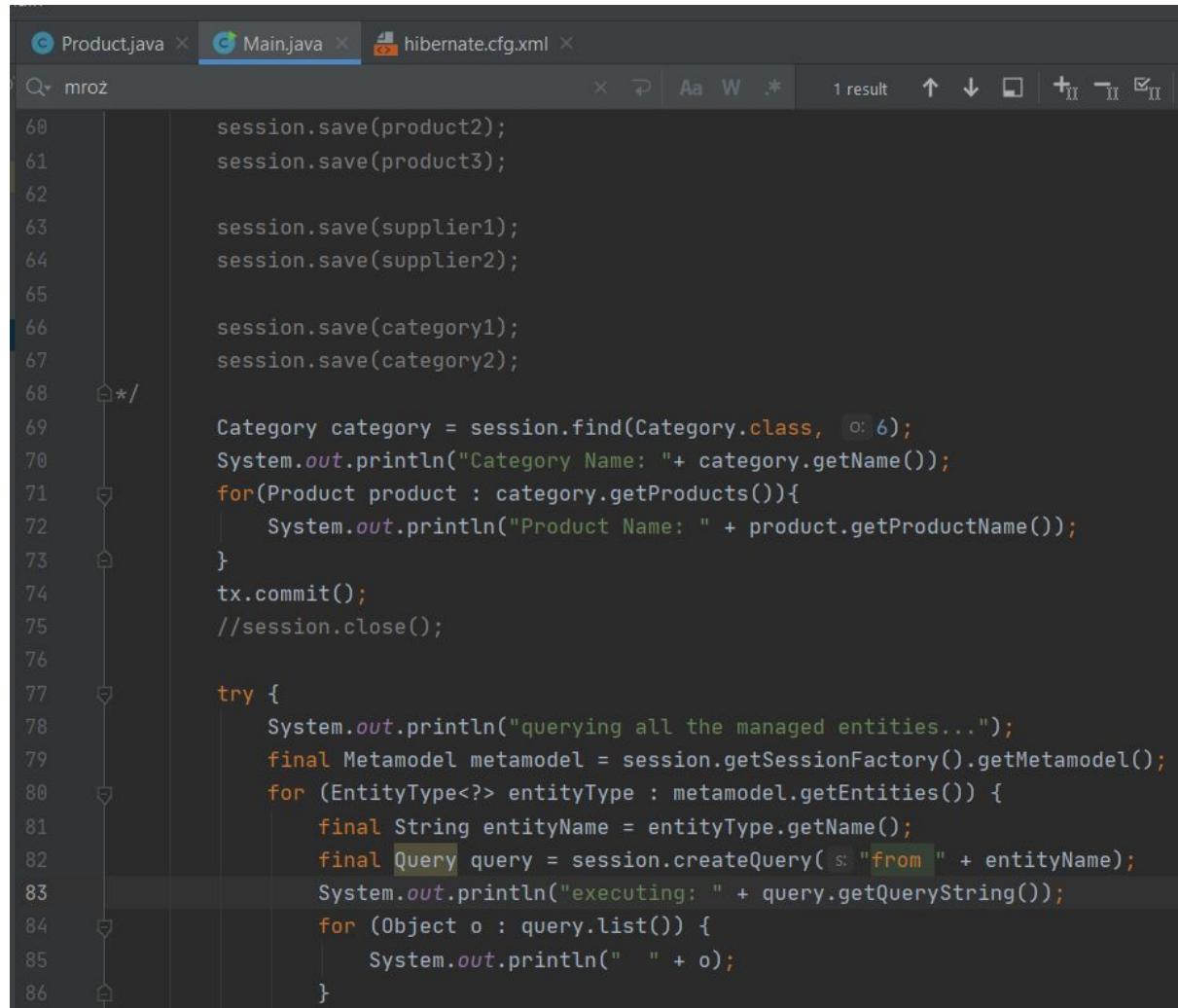
APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] Main.java hibernate.cfg.xml

```

1  <?xml version='1.0' encoding='utf-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3      "-//Hibernate/Hibernate Configuration DTD//EN"
4      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5  <hibernate-configuration>
6      <session-factory>
7          <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
8          <property name="connection.url">jdbc:derby://127.0.0.1/BKordekJPA;create=true</property>
9          <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
10         <property name="show_sql">true</property>
11         <property name="format_sql">true</property>
12         <property name="use_sql_comments">true</property>
13         <property name="hbm2ddl.auto">update</property>
14         <mapping class="Product"></mapping>
15         <mapping class="Supplier"></mapping>
16         <mapping class="Category"></mapping>
17     </session-factory>
18 </hibernate-configuration>

```

## Modyfikacja klasy *Main* - wyszukanie produktów danej kategorii



The screenshot shows a Java code editor with three tabs: Product.java, Main.java (selected), and hibernate.cfg.xml. The Main.java code is as follows:

```
Q mroż
  60     session.save(product2);
  61     session.save(product3);
  62
  63     session.save(supplier1);
  64     session.save(supplier2);
  65
  66     session.save(category1);
  67     session.save(category2);
  68 */
  69
  70     Category category = session.find(Category.class, 6);
  71     System.out.println("Category Name: " + category.getName());
  72     for(Product product : category.getProducts()){
  73         System.out.println("Product Name: " + product.getProductName());
  74     }
  75     tx.commit();
  76     //session.close();
  77
  78     try {
  79         System.out.println("querying all the managed entities...");
  80         final Metamodel metamodel = session.getSessionFactory().getMetamodel();
  81         for (EntityType<?> entityType : metamodel.getEntities()) {
  82             final String entityName = entityType.getName();
  83             final Query query = session.createQuery("from " + entityName);
  84             System.out.println("executing: " + query.getQueryString());
  85             for (Object o : query.list()) {
  86                 System.out.println(" " + o);
  87             }
  88         }
  89     } catch (Exception e) {
  90         e.printStackTrace();
  91     }
  92 }
```

## Uruchomienie programu

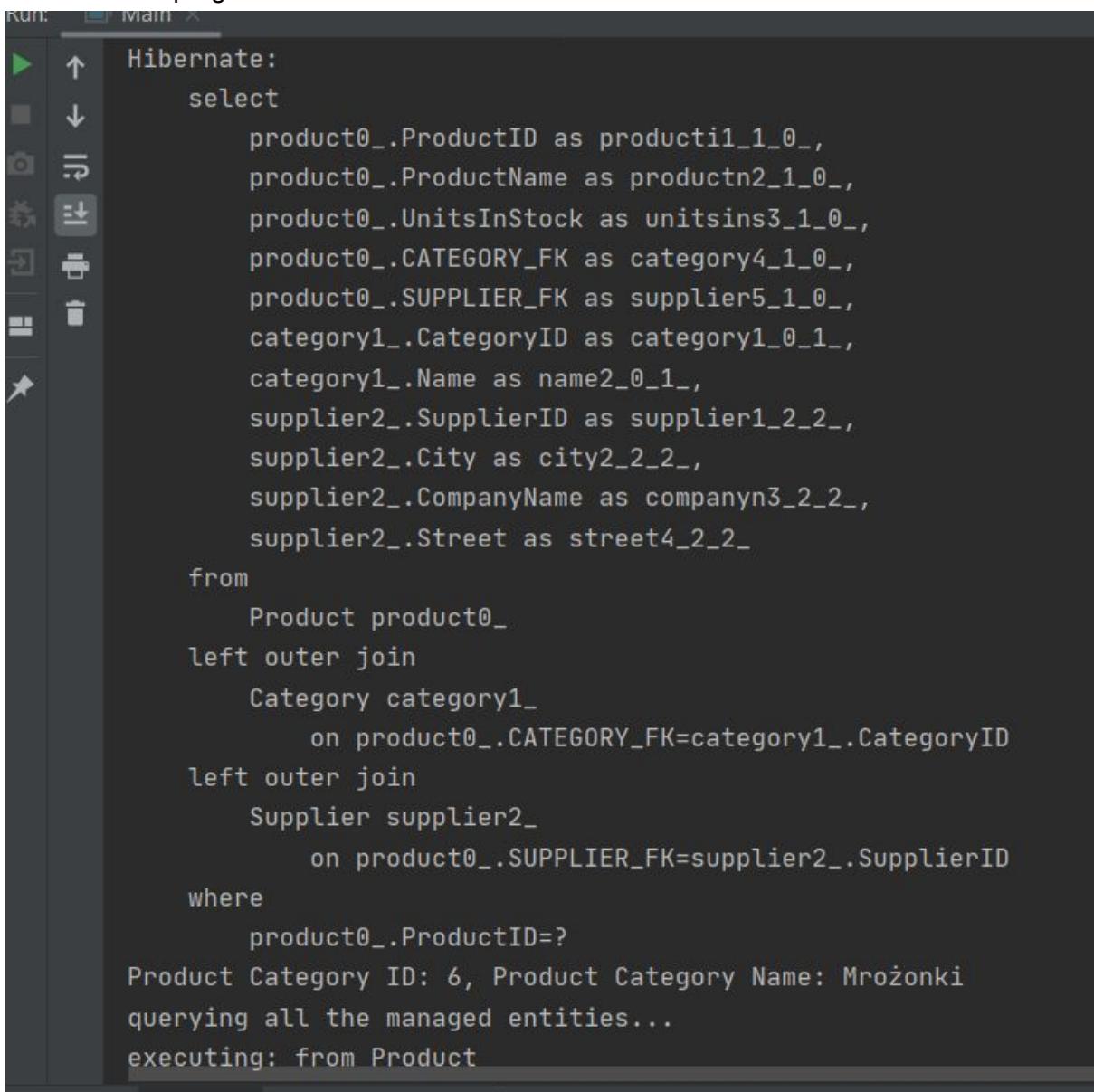
```
Run: Main x
Hibernate:
    select
        category0_.CategoryID as category1_0_0_,
        category0_.Name as name2_0_0_
    from
        Category category0_
    where
        category0_.CategoryID=?
Category Name: Mrożonki
Hibernate:
    select
        products0_.CATEGORY_FK as category4_1_0_,
        products0_.ProductID as product1_1_0_,
        products0_.ProductID as product1_1_1_,
        products0_.ProductName as productn2_1_1_,
        products0_.UnitsInStock as unitsins3_1_1_,
        products0_.CATEGORY_FK as category4_1_1_1,
        products0_.SUPPLIER_FK as supplier5_1_1_1,
        supplier1_.SupplierID as supplier1_2_2_1,
        supplier1_.City as city2_2_2_1,
        supplier1_.CompanyName as companyn3_2_2_1,
        supplier1_.Street as street4_2_2_1
    from
        Product products0_
    left outer join
        Supplier supplier1_
            on products0_.SUPPLIER_FK=supplier1_.SupplierID
    where
        products0_.CATEGORY_FK=?
Product Name: Frytki
Product Name: Pizza
querying all the managed entities..
```

Z powyższego widać, że kategoria o ID 6 ma przypisane następujące produkty: Frytki, Pizza.

Modyfikacja klasy *Main* - wyszukanie kategorii do której należy dany produkt

```
74     }
75
76     */
77     Product product = session.find(Product.class, 1);
78     System.out.println("Product Category ID: " + product.getCategory().getCategoryID()+
79                         ", Product Category Name: " + product.getCategory().getName());
80
81     tx.commit();
82     //session.close();
83
84     try {
85         System.out.println("querying all the managed entities...");
86         final Metamodel metamodel = session.getSessionFactory().getMetamodel();
```

Uruchomienie programu



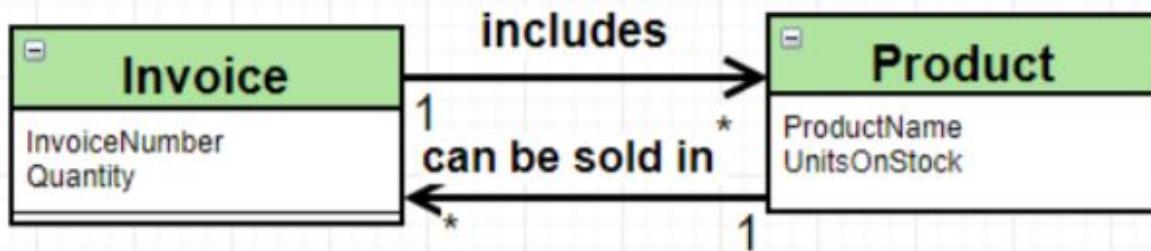
The screenshot shows the IntelliJ IDEA interface with the 'Run' tool window open. The configuration 'Main' is selected. The output pane displays the following SQL query and its execution results:

```
Hibernate:
    select
        product0_.ProductID as product1_1_0_,
        product0_.ProductName as productn2_1_0_,
        product0_.UnitsInStock as unitsins3_1_0_,
        product0_.CATEGORY_FK as category4_1_0_,
        product0_.SUPPLIER_FK as supplier5_1_0_,
        category1_.CategoryID as category1_0_1_,
        category1_.Name as name2_0_1_,
        supplier2_.SupplierID as supplier1_2_2_,
        supplier2_.City as city2_2_2_,
        supplier2_.CompanyName as companyn3_2_2_,
        supplier2_.Street as street4_2_2_
    from
        Product product0_
    left outer join
        Category category1_
            on product0_.CATEGORY_FK=category1_.CategoryID
    left outer join
        Supplier supplier2_
            on product0_.SUPPLIER_FK=supplier2_.SupplierID
    where
        product0_.ProductID=?
```

Product Category ID: 6, Product Category Name: Mrożonki  
querying all the managed entities...  
executing: from Product

Z powyższego widać, że produkt o ID 1 należy do kategorii o ID 6 i nazwie Mrożonki.

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

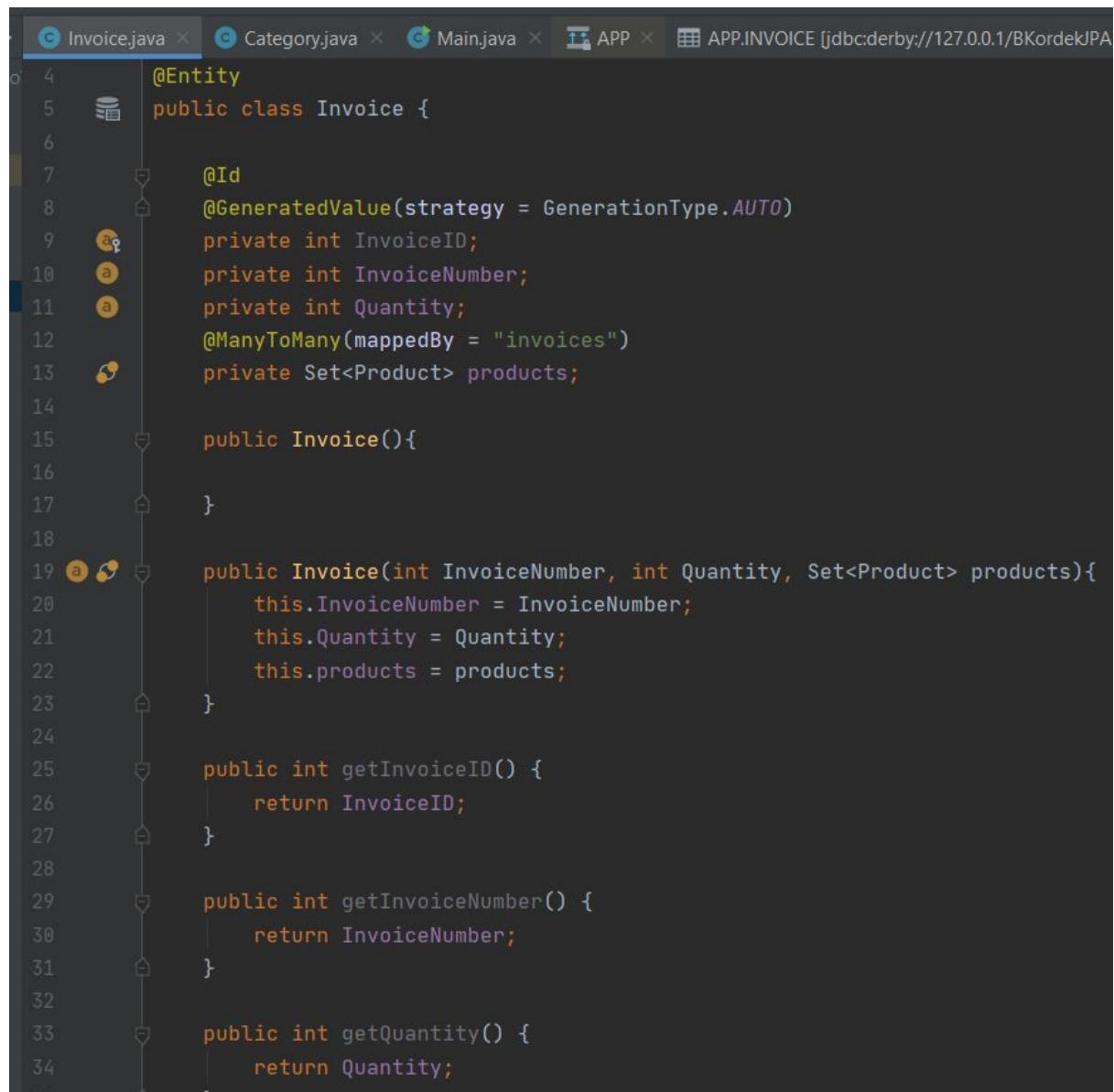


Klasa *Category*

```
Category.java X Main.java X APP X APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA
```

```
4     @Entity
5     public class Category {
6         @Id
7         @GeneratedValue(strategy = GenerationType.AUTO)
8         private int CategoryID;
9         private String Name;
10        @OneToMany
11        @JoinColumn(name = "CATEGORY_FK")
12        private List<Product> Products;
13
14        public Category(){
15        }
16
17
18        public Category(String Name, List<Product> Products){
19            this.Name = Name;
20            this.Products = Products;
21        }
22
23        public List<Product> getProducts() {
24            return Products;
25        }
26
27        public int getCategoryID() {
28            return CategoryID;
29        }
30
31        public String getName() {
32            return Name;
33        }
34    }
```

## Klasa Invoice



The screenshot shows a Java code editor with the file `Invoice.java` open. The code defines a class `Invoice` annotated with `@Entity` and `@Id`. It has a constructor that takes `InvoiceNumber`, `Quantity`, and a `Set<Product>`. It also has three getter methods: `getInvoiceID`, `getInvoiceNumber`, and `getQuantity`. The code is part of a project named `APP` connected to a database `APP.INVOICE`.

```
4  @Entity
5  public class Invoice {
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.AUTO)
9      private int InvoiceID;
10     private int InvoiceNumber;
11     private int Quantity;
12     @ManyToMany(mappedBy = "invoices")
13     private Set<Product> products;
14
15     public Invoice(){
16     }
17
18
19     public Invoice(int InvoiceNumber, int Quantity, Set<Product> products){
20         this.InvoiceNumber = InvoiceNumber;
21         this.Quantity = Quantity;
22         this.products = products;
23     }
24
25     public int getInvoiceID() {
26         return InvoiceID;
27     }
28
29     public int getInvoiceNumber() {
30         return InvoiceNumber;
31     }
32
33     public int getQuantity() {
34         return Quantity;
35     }
36 }
```

```

36
37     public Set<Product> getProducts() {
38         return products;
39     }
40
41     @Override
42     public String toString() {
43         return "Invoice{" +
44             "InvoiceID=" + InvoiceID +
45             ", InvoiceNumber=" + InvoiceNumber +
46             ", Quantity=" + Quantity +
47             ", products=" + products +
48             '}';
49     }
50 }

```

### Klasa Product

```

Product.java × Invoice.java × Category.java × Main.java × APP × APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA] ×
4     @Entity
5     public class Product {
6
7         @Id
8         @GeneratedValue(strategy = GenerationType.AUTO)
9         private int ProductID;
10        private String ProductName;
11        private int UnitsInStock;
12        @ManyToOne
13        @JoinColumn(name="SUPPLIER_FK")
14        private Supplier supplier;
15        @ManyToOne
16        @JoinColumn(name="CATEGORY_FK")
17        private Category category;
18        @ManyToMany
19        private Set<Invoice> invoices;
20
21        public Product(){
22
23    }
24
25        public Product(String ProductName, int UnitsInStock, Supplier supplier, Category category,
26                      Set<Invoice> invoices){
27            this.ProductName=ProductName;
28            this.UnitsInStock=UnitsInStock;
29            this.supplier=supplier;
30            this.category=category;
31            this.invoices=invoices;
32        }
33
34        public String getProductName() {
35            return ProductName;

```

```

33
34     public String getProductName() {
35         return ProductName;
36     }
37
38     public Category getCategory() { return category; }
39
40     public Set<Invoice> getInvoices(){
41         return invoices;
42     }
43
44
45     @Override
46     public String toString() {
47         return "Product{" +
48             "ProductID=" + ProductID +
49             ", ProductName='" + ProductName + '\'' +
50             ", UnitsInStock=" + UnitsInStock +
51             ", supplier=" + supplier.getSupplierID() + ' ' + supplier.getCompanyName() +
52             ", category=" + category.getCategoryID() + ' ' + category.getName() +
53             '}';
54     }
55
56 }

```

## Klasa Supplier

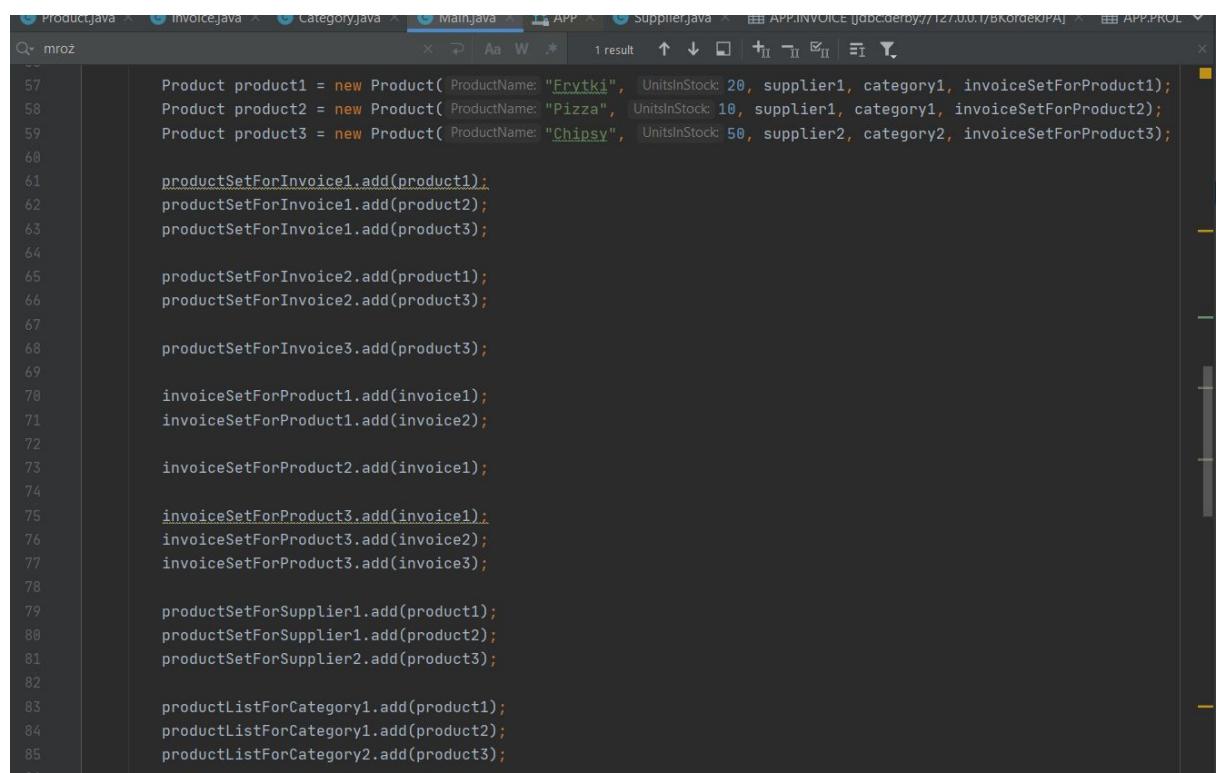
```

Product.java × Invoice.java × Category.java × Main.java × APP × Supplier.java × APP.INVOICE [jdbc:derby://127.0.0.1:1234/APP]
5     @Entity
6     public class Supplier {
7
8         @Id
9         @GeneratedValue(strategy = GenerationType.AUTO)
10        private int SupplierID;
11        private String CompanyName;
12        private String Street;
13        private String City;
14        @OneToMany
15        @JoinColumn(name="SUPPLIER_FK")
16        private Set<Product> products;
17
18        public Supplier(){
19
20    }
21
22        public Supplier(String CompanyName, String Street, String City, Set<Product> products){
23            this.CompanyName = CompanyName;
24            this.Street = Street;
25            this.City = City;
26            this.products = products;
27        }
28
29        public int getSupplierID() {
30            return SupplierID;
31        }
32
33        public String getCompanyName() {
34            return CompanyName;
35        }

```

## Zmiany w klasie sterującej Main - stworzenie kilku produktów i sprzedanie ich w kilku transakcjach

```
30  public static void main(final String[] args) throws Exception {
31      final Session session = getSession();
32
33      Set<Product> productSetForSupplier1 = new HashSet<>();
34      Set<Product> productSetForSupplier2 = new HashSet<>();
35
36      List<Product> productListForCategory1 = new ArrayList<>();
37      List<Product> productListForCategory2 = new ArrayList<>();
38
39      Set<Product> productSetForInvoice1 = new HashSet<>();
40      Set<Product> productSetForInvoice2 = new HashSet<>();
41      Set<Product> productSetForInvoice3 = new HashSet<>();
42
43      Set<Invoice> invoiceSetForProduct1 = new HashSet<>();
44      Set<Invoice> invoiceSetForProduct2 = new HashSet<>();
45      Set<Invoice> invoiceSetForProduct3 = new HashSet<>();
46
47      Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
48      Category category2 = new Category( Name: "Przekąski", productListForCategory2);
49
50      Supplier supplier1 = new Supplier( CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków", productSetForSupplier1);
51      Supplier supplier2 = new Supplier( CompanyName: "Żabka", Street: "Sotyśówka", City: "Kraków", productSetForSupplier2);
52
53      Invoice invoice1 = new Invoice( InvoiceNumber: 123, Quantity: 10, productSetForInvoice1);
54      Invoice invoice2 = new Invoice( InvoiceNumber: 124, Quantity: 10, productSetForInvoice2);
55      Invoice invoice3 = new Invoice( InvoiceNumber: 125, Quantity: 10, productSetForInvoice3);
```



```
Q mroż
57  Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20, supplier1, category1, invoiceSetForProduct1);
58  Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10, supplier1, category1, invoiceSetForProduct2);
59  Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50, supplier2, category2, invoiceSetForProduct3);
60
61  productSetForInvoice1.add(product1);
62  productSetForInvoice1.add(product2);
63  productSetForInvoice1.add(product3);
64
65  productSetForInvoice2.add(product1);
66  productSetForInvoice2.add(product3);
67
68  productSetForInvoice3.add(product3);
69
70  invoiceSetForProduct1.add(invoice1);
71  invoiceSetForProduct1.add(invoice2);
72
73  invoiceSetForProduct2.add(invoice1);
74
75  invoiceSetForProduct3.add(invoice1);
76  invoiceSetForProduct3.add(invoice2);
77  invoiceSetForProduct3.add(invoice3);
78
79  productSetForSupplier1.add(product1);
80  productSetForSupplier1.add(product2);
81  productSetForSupplier2.add(product3);
82
83  productListForCategory1.add(product1);
84  productListForCategory1.add(product2);
85  productListForCategory2.add(product3);
```

```

86
87     Transaction tx = session.beginTransaction();
88
89     session.save(product1);
90     session.save(product2);
91     session.save(product3);
92
93     session.save(supplier1);
94     session.save(supplier2);
95
96     session.save(category1);
97     session.save(category2);
98
99     session.save(invoice1);
100    session.save(invoice2);
101    session.save(invoice3);
102

```

### Modyfikacja pliku konfiguracyjnego - dodanie mapowania klasy *Invoice*

```

1  <?xml version='1.0' encoding='utf-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3      "-//Hibernate/Hibernate Configuration DTD//EN"
4      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5  <hibernate-configuration>
6      <session-factory>
7          <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
8          <property name="connection.url">jdbc:derby://127.0.0.1/BKordekJPA;create=true</property>
9          <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
10         <property name="show_sql">true</property>
11         <property name="format_sql">true</property>
12         <property name="use_sql_comments">true</property>
13         <property name="hbm2ddl.auto">create-drop</property>
14         <mapping class="Product"></mapping>
15         <mapping class="Supplier"></mapping>
16         <mapping class="Category"></mapping>
17         <mapping class="Invoice"></mapping>
18     </session-factory>
19 </hibernate-configuration>

```

### Uruchomienie programu

```

Run: Main x
  hibernate:
    /*
     * 
    */
  from
    Invoice */ select
      invoice0_.InvoiceID as invoice1_1_,
      invoice0_.InvoiceNumber as invoice2_1_,
      invoice0_.Quantity as quantity3_1_
  from
    Invoice invoice0_
  Invoice{InvoiceID=8, InvoiceNumber=123, Quantity=10, products=[Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, supplier=4 Tesco, category=6 Mrożonki}, Product{ProductID=9, InvoiceNumber=124, Quantity=10, products=[Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, supplier=4 Tesco, category=6 Mrożonki}, Product{ProductID=10, InvoiceNumber=125, Quantity=10, products=[Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, supplier=5 Zabka, category=7 Przekąski]}]}
  executing: from Product
  Hibernate:
    /*
     */
  from
    Product */ select
      product0_.ProductID as product1_2_,
      product0_.ProductName as productn2_2_,
      product0_.UnitsInStock as unitsins2_2_,
      product0_.CATEGORY_FK as category4_2_,
      product0_.SUPPLIER_FK as supplier5_2_
  from
    Product product0_
  Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, supplier=4 Tesco, category=6 Mrożonki}
  Product{ProductID=2, ProductName='Pizza', UnitsInStock=10, supplier=4 Tesco, category=6 Mrożonki}
  Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, supplier=5 Zabka, category=7 Przekąski}

Process finished with exit code 0

```

## Schemat bazy danych

Database

jdbc:derby://127.0.0.1/BKordekJPA 1 of 11

APP

▼ CATEGORY

- KEY CATEGORYID INTEGER
- NAME VARCHAR(255)
- SQL210102225313290 (CATEGORYID)
- SQL210102225313290 (CATEGORYID) UNIQUE

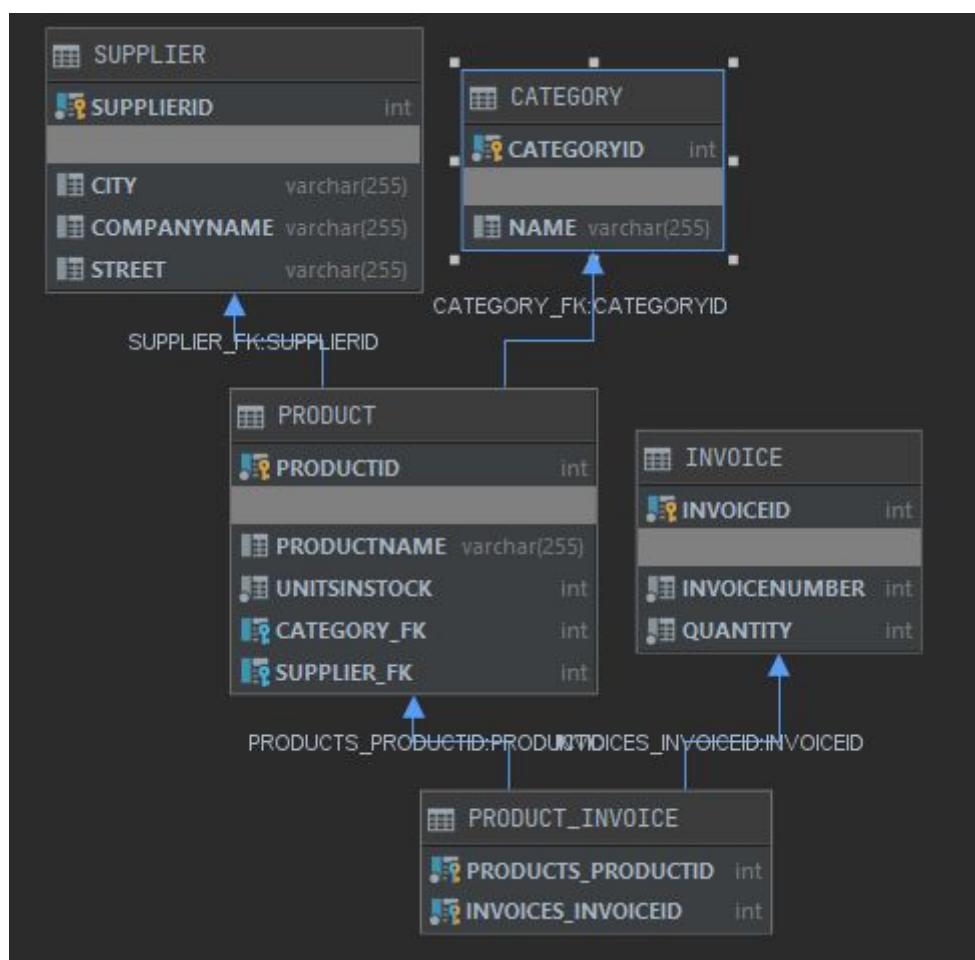
▼ INVOICE

- KEY INVOICEID INTEGER
- INVOICENUMBER INTEGER
- QUANTITY INTEGER
- SQL210102225313310 (INVOICEID)
- SQL210102225313310 (INVOICEID) UNIQUE

▼ PRODUCT

- KEY PRODUCTID INTEGER
- PRODUCTNAME VARCHAR(255)
- UNITSINSTOCK INTEGER
- KEY CATEGORY\_FK INTEGER
- KEY SUPPLIER\_FK INTEGER
- SQL210102225313320 (PRODUCTID)
- FKEURY2HXL2J8URLKMW36585TKR (SUPPLIER\_FK) → SUPPLIER (SUPPLIERID)
- FKPURGJJ563MV2VAV0MGGDXEFD7 (CATEGORY\_FK) → CATEGORY (CATEGORYID)
- SQL210102225313320 (PRODUCTID) UNIQUE
- SQL210102225313370 (CATEGORY\_FK)
- SQL210102225313380 (SUPPLIER\_FK)

1	SQL210102225313380	(SUPPLIER_FK)
▼ PRODUCT_INVOICE		
	PRODUCTS_PRODUCTID	INTEGER
	INVOICES_INVOICEID	INTEGER
?	SQL210102225313340	(PRODUCTS_PRODUCTID, INVOICES_INVOICEID)
?	FK30TSIIUDMV7Y4P1360MNQ4V7R	(INVOICES_INVOICEID) → INVOICE (INVOICEID)
?	FKMCMOHEDFHSUOR6LTI20A304QD	(PRODUCTS_PRODUCTID) → PRODUCT (PRODUCTID)
ju	SQL210102225313340	(PRODUCTS_PRODUCTID, INVOICES_INVOICEID) UNIQUE
i	SQL210102225313390	(INVOICES_INVOICEID)
i	SQL210102225313400	(PRODUCTS_PRODUCTID)
▼ SUPPLIER		
	SUPPLIERID	INTEGER
	CITY	VARCHAR(255)
	COMPANYNAME	VARCHAR(255)
	STREET	VARCHAR(255)
?	SQL210102225313350	(SUPPLIERID)
ju	SQL210102225313350	(SUPPLIERID) UNIQUE



## Zawartość tabel

The screenshot shows a database management interface with four tabs open, each displaying the contents of a table:

- APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA]**: Contains 2 rows.
- APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]**: Contains 3 rows.
- APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]**: Contains 3 rows.
- APP.PRODUCT\_INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]**: Contains 6 rows.

**APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA]**

	CATEGORYID	NAME
1	6	Mrożonki
2	7	Przekąski

**APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]**

	INVOICEID	INVOICENUMBER	QUANTITY
1	8	123	10
2	9	124	10
3	10	125	10

**APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]**

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Frytki	20	6	4
2	2	Pizza	10	6	4
3	3	Chipsy	50	7	5

**APP.PRODUCT\_INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]**

	PRODUCTS_PRODUCTID	INVOICES_INVOICEID
1	1	8
2	1	9
3	2	8
4	3	8
5	3	9
6	3	10

APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] X

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Tesco	Kapelanka
2	5	Kraków	Żabka	Sołtysowka

Z powyższego widać, że została stworzona tabela łącznikowa *PRODUCT\_INVOICE* zawierająca klucze obce, będące ID produktu i przypisanej do niej ID faktury.

### Pokaż produkty sprzedane w ramach wybranej faktury/transakcji

Modyfikacja klasy sterującej *Main* - wyciągnięcie z bazy produktów znajdujących się na danej fakturze

```

102     */
103     Invoice invoice = session.find(Invoice.class, 8);
104     for(Product product : invoice.getProducts())
105         System.out.println("Product Name: " + product.getProductName());
106
107     /*

```

Uruchomienie programu

Run: Main X

```

Supplier supplier3_
    on product1_.SUPPLIER_FK=supplier3_.SupplierID
    where
        products0_.invoices_InvoiceID=?
Product Name: Frytki
Product Name: Pizza
Product Name: Chipsy
querying all the managed entities...
executing: from Supplier
Hibernate:
    /*
from
    Supplier * select
        supplier0_.SupplierID as supplier1_4_,
        supplier0_.City as city2_4_,
        supplier0_.CompanyName as companyn3_4_,
        supplier0_.Street as street4_4_

```

Z powyższego widać, że na fakturze o ID 8 znajdują się następujące produkty: Frytki, Pizza, Chipsy.

## Pokaż faktury w ramach których był sprzedany wybrany produkt

Modyfikacja klasy sterującej *Main* - wyciągnięcie z bazy faktur, do których przypisany jest dany produkt.

```
107     */
108     Product product = session.find(Product.class,  o: 1);
109     for(Invoice invoice : product.getInvoices())
110         System.out.println("Invoice ID: " + invoice.getInvoiceID() +
111             " Invoice Number: " + invoice.getInvoiceNumber());
112     /*

```

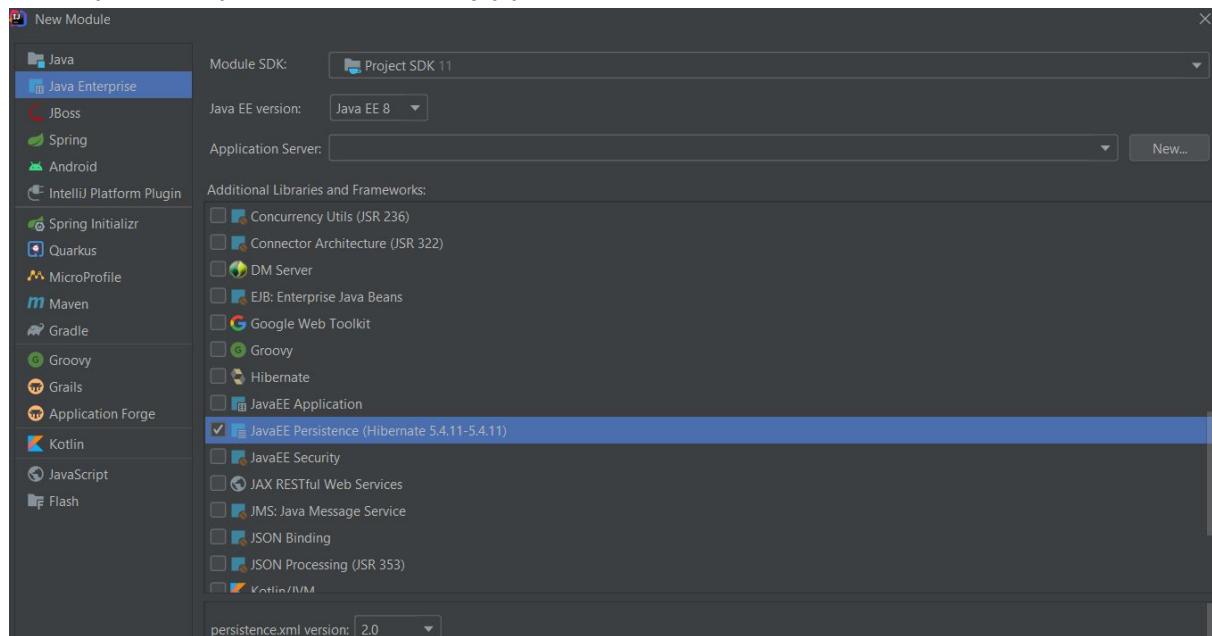
## Uruchomienie programu

```
Run: Main ×
▶
    Product_Invoice invoices0_
    inner join
        Invoice invoice1_
        on invoices0_.invoices_InvoiceID=invoice1_.InvoiceID
    where
        invoices0_.products_ProductID=?
    Invoice ID: 9 Invoice Number: 124
    Invoice ID: 8 Invoice Number: 123
    querying all the managed entities...
    executing: from Category
    Hibernate:
        /*
    from
        Category * select
            category0_.CategoryID as category1_0_,
            category0_.Name as name2_0_
    from
        Category category0_
    Category{CategoryID=6, Name='Mrożonki'}
```

Z powyższego widać, że produkt o ID równym 1 jest przypisany do kategorii o ID równym 6 i nazwie Mrożonki.

## X. JPA

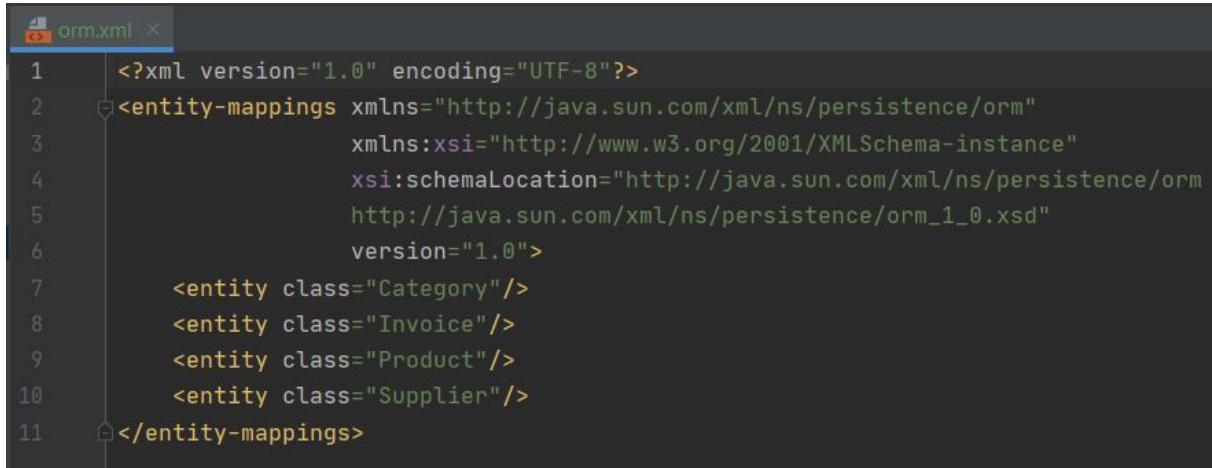
Stworzyłem nowy moduł - JPA bazujący na bibliotece i frameworku **JavaEE Persistence**



Modyfikacja pliku konfiguracyjnego

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
    version="2.0">
    <persistence-unit name="BKordekJPA"
        transaction-type="RESOURCE_LOCAL">
        <mapping-file>META-INF/orm.xml</mapping-file>
        <properties>
            <property name="hibernate.connection.driver_class"
                value="org.apache.derby.jdbc.ClientDriver"/>
            <property name="hibernate.connection.url"
                value="jdbc:derby://127.0.0.1/BKordekJPA"/>
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.use_sql_comments" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="create" />
        </properties>
    </persistence-unit>
</persistence>
```

Utworzenie pliku konfiguracyjnego z mappingiem klas.



The screenshot shows a code editor window with the file 'orm.xml' open. The XML code defines entity mappings for four classes: Category, Invoice, Product, and Supplier. The code is color-coded for readability, with tags in blue and attributes in green.

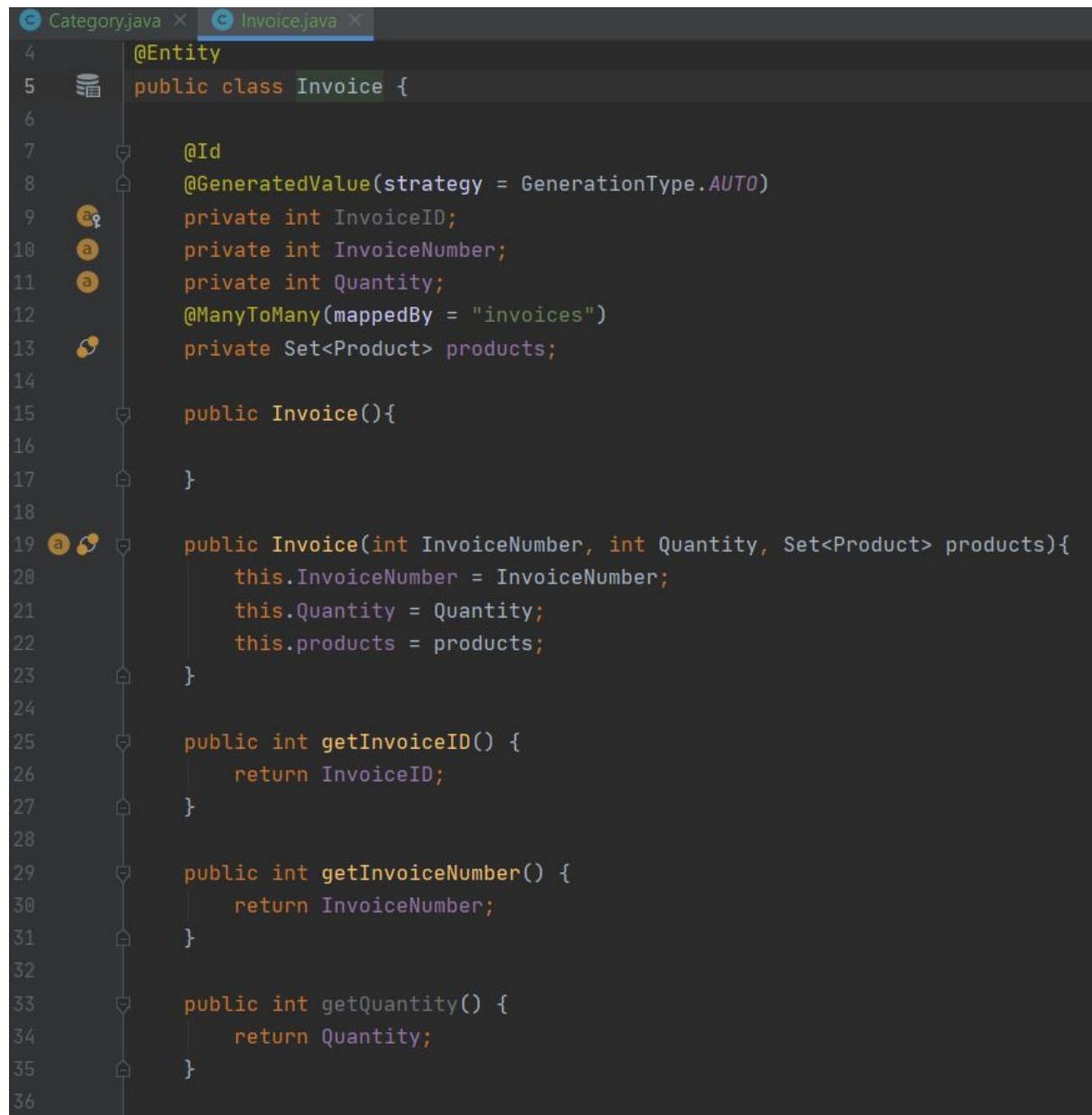
```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
        http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
    version="1.0">
    <entity class="Category"/>
    <entity class="Invoice"/>
    <entity class="Product"/>
    <entity class="Supplier"/>
</entity-mappings>
```

Klasy *Category*, *Invoice*, *Product*, *Supplier* są takie same jak poprzednio.

### Klasa *Category*

```
c Category.java ×
4     @Entity
5     public class Category {
6         @Id
7         @GeneratedValue(strategy = GenerationType.AUTO)
8         private int CategoryID;
9         private String Name;
10        @OneToMany
11        @JoinColumn(name = "CATEGORY_FK")
12        private List<Product> Products;
13
14        public Category(){
15
16        }
17
18        public Category(String Name, List<Product> Products){
19            this.Name = Name;
20            this.Products = Products;
21        }
22
23        public List<Product> getProducts() {
24            return Products;
25        }
26
27        public int getCategoryID() {
28            return CategoryID;
29        }
30
31        public String getName() {
32            return Name;
33        }
34
35        @Override
36        public String toString() {
```

## Klasa Invoice



The screenshot shows a code editor with two tabs: "Category.java" and "Invoice.java". The "Invoice.java" tab is active, displaying the following Java code:

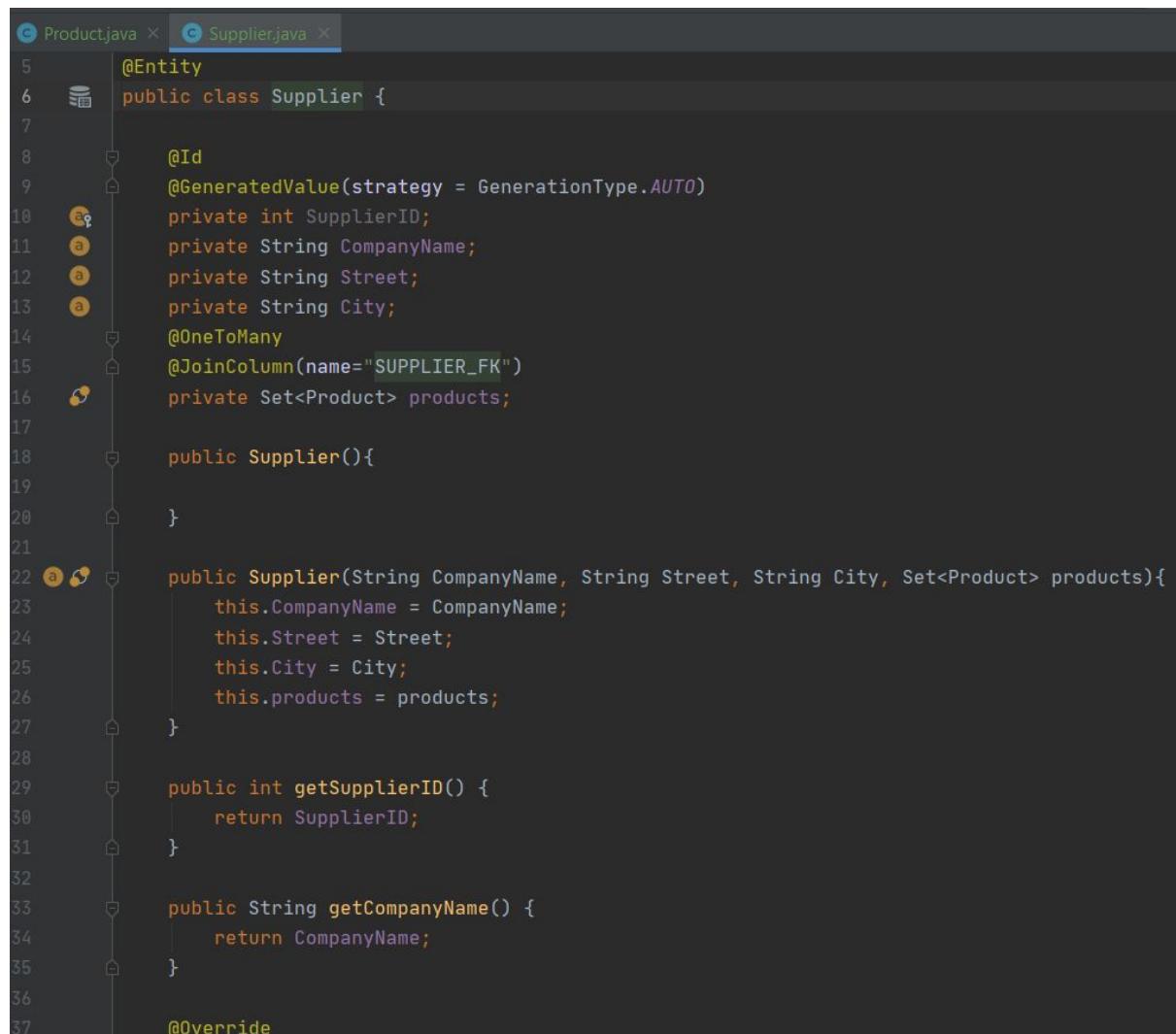
```
4  @Entity
5  public class Invoice {
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.AUTO)
9      private int InvoiceID;
10     private int InvoiceNumber;
11     private int Quantity;
12     @ManyToMany(mappedBy = "invoices")
13     private Set<Product> products;
14
15     public Invoice(){
16     }
17
18
19     public Invoice(int InvoiceNumber, int Quantity, Set<Product> products){
20         this.InvoiceNumber = InvoiceNumber;
21         this.Quantity = Quantity;
22         this.products = products;
23     }
24
25     public int getInvoiceID() {
26         return InvoiceID;
27     }
28
29     public int getInvoiceNumber() {
30         return InvoiceNumber;
31     }
32
33     public int getQuantity() {
34         return Quantity;
35     }
36 }
```

The code defines a class named "Invoice" with annotations for Entity, Id, and ManyToMany. It includes constructor methods, getter methods for InvoiceID, InvoiceNumber, and Quantity, and a getter method for the products set.

## Klasa Product

```
Product.java ×
4  | @Entity
5  | public class Product {
6  |
7  |     @Id
8  |     @GeneratedValue(strategy = GenerationType.AUTO)
9  |     private int ProductID;
10 |     private String ProductName;
11 |     private int UnitsInStock;
12 |     @ManyToOne
13 |     @JoinColumn(name="SUPPLIER_FK")
14 |     private Supplier supplier;
15 |     @ManyToOne
16 |     @JoinColumn(name="CATEGORY_FK")
17 |     private Category category;
18 |     @ManyToMany
19 |     private Set<Invoice> invoices;
20 |
21 |     public Product(){}
22 |
23 | }
24 |
25 | public Product(String ProductName, int UnitsInStock, Supplier supplier, Category category,
26 |                  Set<Invoice> invoices){
27 |     this.ProductName=ProductName;
28 |     this.UnitsInStock=UnitsInStock;
29 |     this.supplier=supplier;
30 |     this.category=category;
31 |     this.invoices=invoices;
32 | }
33 |
34 | public String getProductName() { return ProductName; }
35 |
36 | public Category getCategory() { return category; }
```

## Klasa Supplier



```
5  @Entity
6  public class Supplier {
7
8      @Id
9      @GeneratedValue(strategy = GenerationType.AUTO)
10     private int SupplierID;
11     private String CompanyName;
12     private String Street;
13     private String City;
14
15     @OneToMany
16     @JoinColumn(name="SUPPLIER_FK")
17     private Set<Product> products;
18
19     public Supplier(){
20
21
22     public Supplier(String CompanyName, String Street, String City, Set<Product> products){
23         this.CompanyName = CompanyName;
24         this.Street = Street;
25         this.City = City;
26         this.products = products;
27     }
28
29     public int getSupplierID() {
30         return SupplierID;
31     }
32
33     public String getCompanyName() {
34         return CompanyName;
35     }
36
37     @Override
```

Klasa sterująca *Main* - stworzenie tego co w punkcie VIII z wykorzystaniem JPA

Różnicą jest wykorzystanie klas *EntityManagerFactory*, *EntityManager*, *EntityTransaction*

```
>Main.java ×
1 import javax.persistence.EntityManager;
2 import javax.persistence.EntityManagerFactory;
3 import javax.persistence.EntityTransaction;
4 import javax.persistence.Persistence;
5 import java.util.ArrayList;
6 import java.util.HashSet;
7 import java.util.List;
8 import java.util.Set;
9
10 public class Main {
11     public static void main(String[] args) {
12         EntityManagerFactory emf = Persistence.createEntityManagerFactory("BKordekJPA");
13         EntityManager em = emf.createEntityManager();
14         EntityTransaction etx = em.getTransaction();
15         etx.begin();
16
17         Set<Product> productSetForSupplier1 = new HashSet<>();
18         Set<Product> productSetForSupplier2 = new HashSet<>();
19
20         List<Product> productListForCategory1 = new ArrayList<>();
21         List<Product> productListForCategory2 = new ArrayList<>();
22
23         Set<Product> productSetForInvoice1 = new HashSet<>();
24         Set<Product> productSetForInvoice2 = new HashSet<>();
25         Set<Product> productSetForInvoice3 = new HashSet<>();
26
27         Set<Invoice> invoiceSetForProduct1 = new HashSet<>();
28         Set<Invoice> invoiceSetForProduct2 = new HashSet<>();
29         Set<Invoice> invoiceSetForProduct3 = new HashSet<>();
30
31         Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
32         Category category2 = new Category( Name: "Przekąski", productListForCategory2);
33 }
```

```
>Main.java ×
33
34 Supplier supplier1 = new Supplier( CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków", productSetForSupplier1);
35 Supplier supplier2 = new Supplier( CompanyName: "Żabka", Street: "Słottysówka", City: "Kraków", productSetForSupplier2);
36
37 Invoice invoice1 = new Invoice( InvoiceNumber: 123, Quantity: 10, productSetForInvoice1);
38 Invoice invoice2 = new Invoice( InvoiceNumber: 124, Quantity: 10, productSetForInvoice2);
39 Invoice invoice3 = new Invoice( InvoiceNumber: 125, Quantity: 10, productSetForInvoice3);
40
41 Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20, supplier1, category1, invoiceSetForProduct1);
42 Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10, supplier1, category1, invoiceSetForProduct1);
43 Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50, supplier2, category2, invoiceSetForProduct2);
44
45 productSetForInvoice1.add(product1);
46 productSetForInvoice1.add(product2);
47 productSetForInvoice1.add(product3);
48
49 productSetForInvoice2.add(product1);
50 productSetForInvoice2.add(product3);
51
52 productSetForInvoice3.add(product3);
53
54 invoiceSetForProduct1.add(invoice1);
55 invoiceSetForProduct1.add(invoice2);
56
57 invoiceSetForProduct2.add(invoice1);
58
59 invoiceSetForProduct3.add(invoice1);
60 invoiceSetForProduct3.add(invoice2);
61 invoiceSetForProduct3.add(invoice3);
62
63 productSetForSupplier1.add(product1);
64 productSetForSupplier1.add(product2);
65 productSetForSupplier2.add(product3);
```

```
Main.java X
65     productListForSupplier2.add(product3);
66
67     productListForCategory1.add(product1);
68     productListForCategory1.add(product2);
69     productListForCategory2.add(product3);
70
71     em.persist(product1);
72     em.persist(product2);
73     em.persist(product3);
74
75     em.persist(supplier1);
76     em.persist(supplier2);
77
78     em.persist(category1);
79     em.persist(category2);
80
81     em.persist(invoice1);
82     em.persist(invoice2);
83     em.persist(invoice3);
84
85     Invoice invoice = em.find(Invoice.class, 8);
86     for(Product product : invoice.getProducts())
87         System.out.println("Product Name: " + product.getProductName());
88
89     Product product = em.find(Product.class, 1);
90     for(Invoice inv : product.getInvoices())
91         System.out.println("Invoice ID: " + inv.getInvoiceID() +
92                           " Invoice Number: " + inv.getInvoiceNumber());
93
94     etx.commit();
95     em.close();
96 }
97 }
```

Uruchomienie programu

```
Run: Main (1) X
values
    next value for hibernate_sequence
Product Name: Chipsy
Product Name: Frytki
Product Name: Pizza
Invoice ID: 8 Invoice Number: 123
Invoice ID: 9 Invoice Number: 124
Hibernate:
    /* insert Product
       *_ insert
          into
             Product
```

## Schemat bazy danych

Database

jdbc:derby://127.0.0.1/BKordekJPA 1 of 11

APP

CATEGORY

- KEY CATEGORYID INTEGER
- NAME VARCHAR(255)
- SQL210103022759690 (CATEGORYID)
- SQL210103022759690 (CATEGORYID) UNIQUE

INVOICE

- KEY INVOICEID INTEGER
- INVOICENUMBER INTEGER
- QUANTITY INTEGER
- SQL210103022759720 (INVOICEID)
- SQL210103022759720 (INVOICEID) UNIQUE

PRODUCT

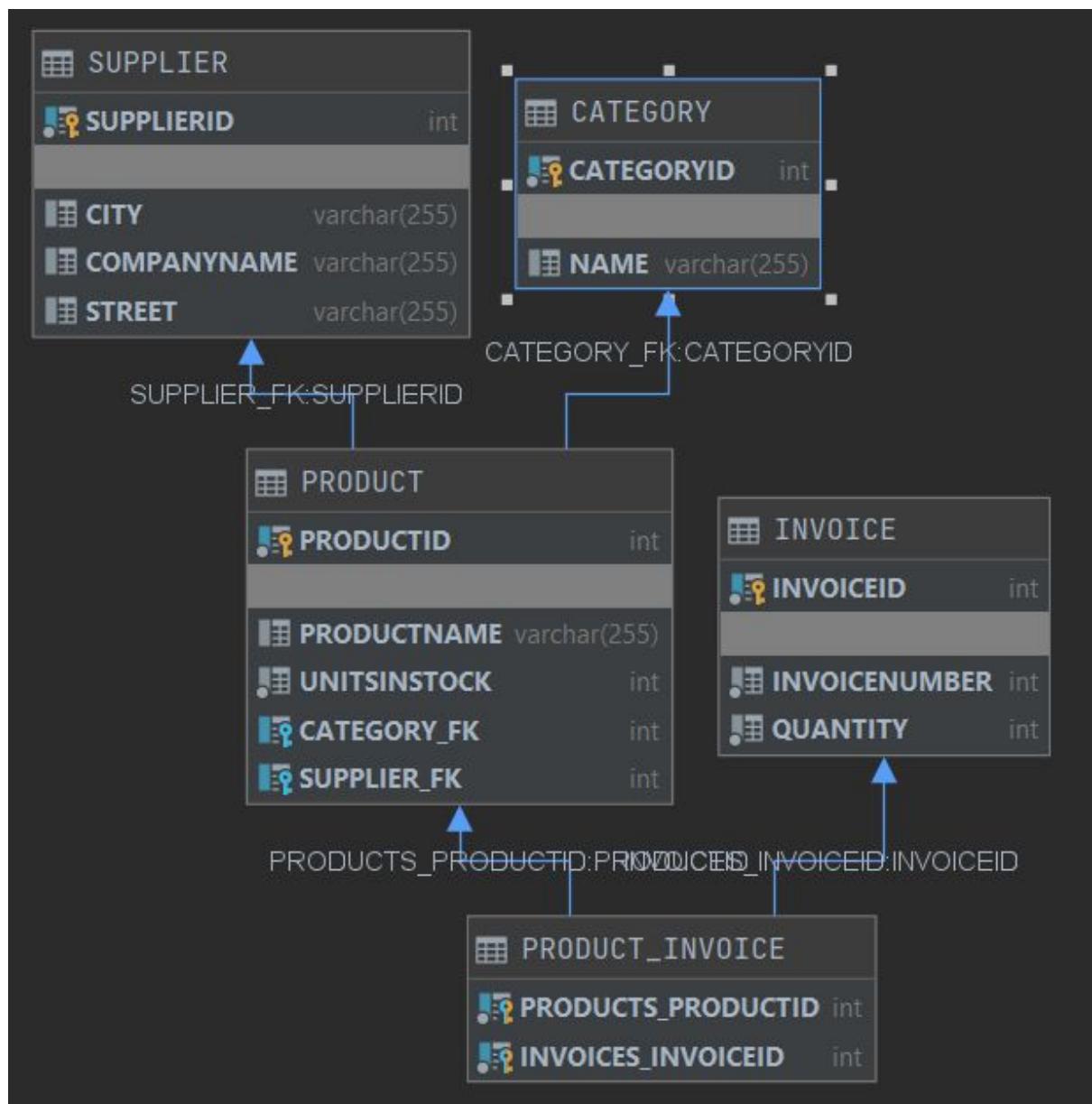
- KEY PRODUCTID INTEGER
- PRODUCTNAME VARCHAR(255)
- UNITSINSTOCK INTEGER
- KEY CATEGORY\_FK INTEGER
- KEY SUPPLIER\_FK INTEGER
- SQL210103022759740 (PRODUCTID)
- FKEURY2HXL2J8URLKMW36585TKR (SUPPLIER\_FK) → SUPPLIER (SUPPLIERID)
- FKPURGJJ563MV2VAV0MGGDXEFD7 (CATEGORY\_FK) → CATEGORY (CATEGORYID)
- SQL210103022759740 (PRODUCTID) UNIQUE
- SQL210103022759810 (CATEGORY\_FK)
- SQL210103022759840 (SUPPLIER\_FK)

PRODUCT\_INVOICE

- KEY PRODUCTS\_PRODUCTID INTEGER
- KEY INVOICES\_INVOICEID INTEGER
- SQL210103022759760 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID)
- FK30TSIIUDMV7Y4P1360MNQ4V7R (INVOICES\_INVOICEID) → INVOICE (INVOICEID)
- FKMCMOHEDFHSUOR6LT120A304QD (PRODUCTS\_PRODUCTID) → PRODUCT (PRODUCTID)
- SQL210103022759760 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID) UNIQUE
- SQL210103022759870 (INVOICES\_INVOICEID)
- SQL210103022759890 (PRODUCTS\_PRODUCTID)

SUPPLIER

- KEY SUPPLIERID INTEGER
- CITY VARCHAR(255)
- COMPANYNAME VARCHAR(255)
- STREET VARCHAR(255)
- SQL210103022759780 (SUPPLIERID)
- SQL210103022759780 (SUPPLIERID) UNIQUE



Zawartość tabel

APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA]	
<	>
2 rows	
1	6 Mrożonki
2	7 Przekąski

**APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]**

	INVOICEID	INVOICENUMBER	QUANTITY
1	8	123	10
2	9	124	10
3	10	125	10

**APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]**

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Frytki	20	6	4
2	2	Pizza	10	6	4
3	3	Chipsy	50	7	5

**APP.PRODUCT\_INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]**

	PRODUCTS_PRODUCTID	INVOICES_INVOICEID
1	1	8
2	1	9
3	2	8
4	3	8
5	3	9
6	3	10

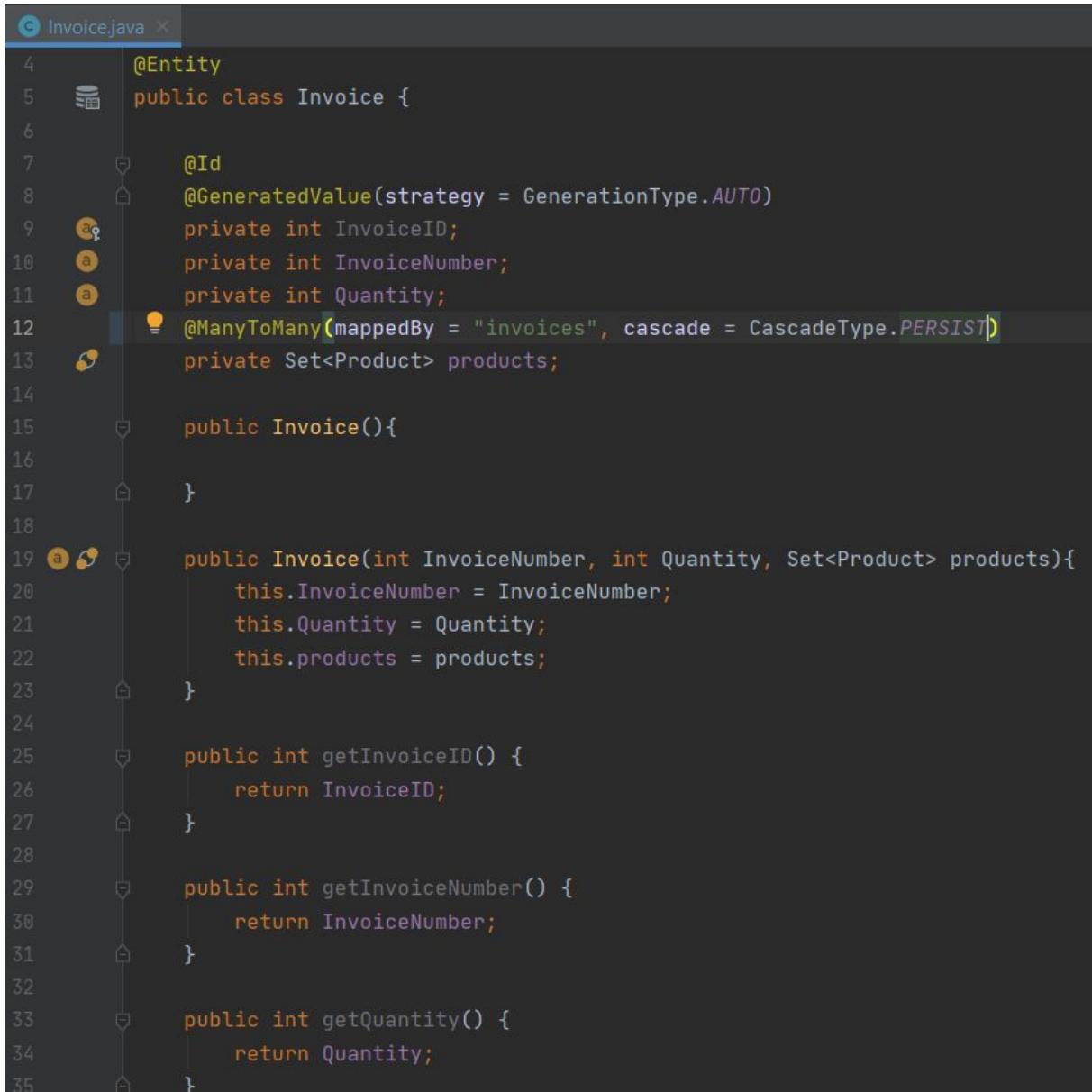
**APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA]**

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Tesco	Kapelanka
2	5	Kraków	Żabka	Sołtysowka

Jak widać powyżej, wynik jest taki sam jak w przypadku wykorzystania Hibernate.

## XI. Kaskady

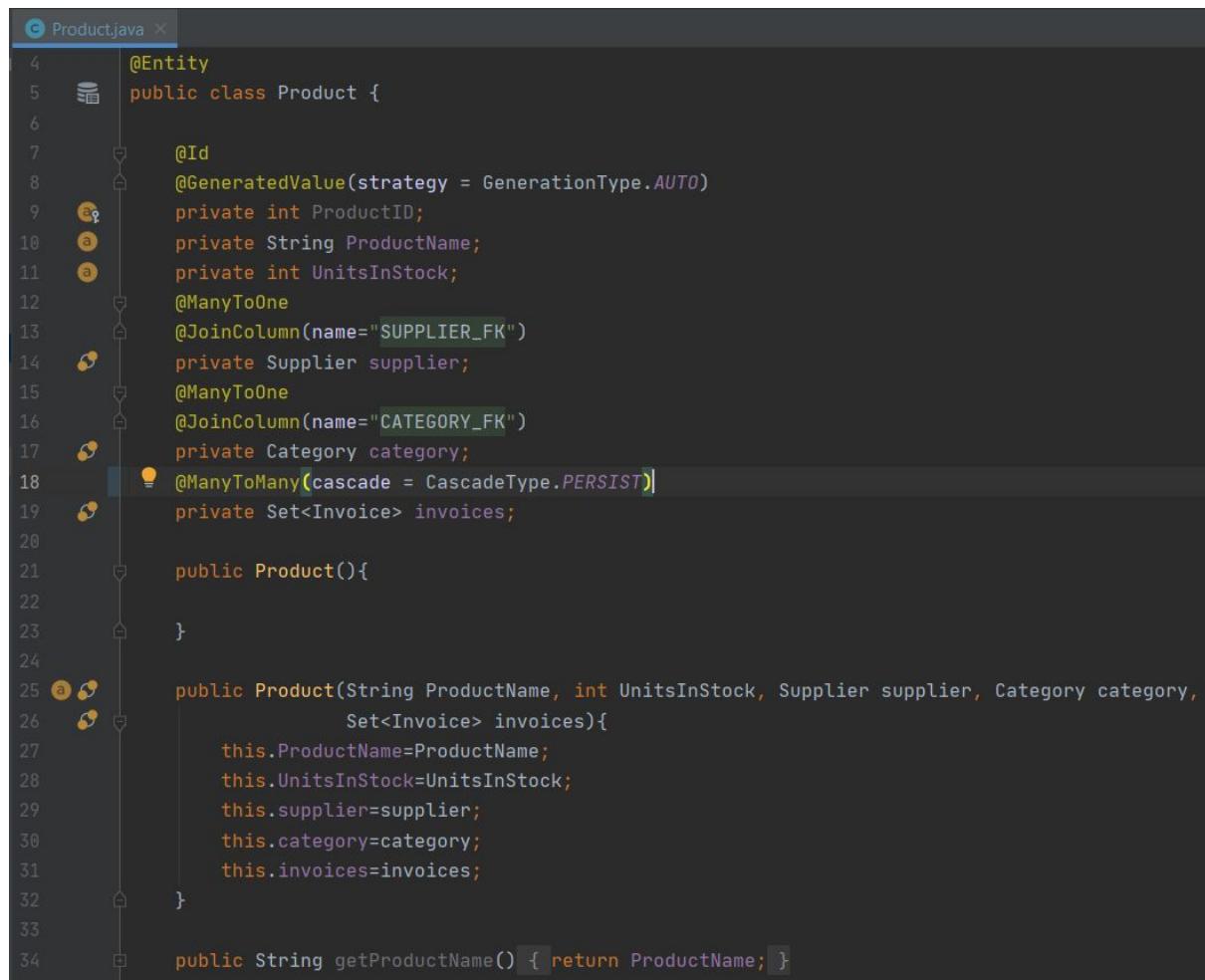
### Klasa *Invoice*



The screenshot shows a code editor window for a Java file named `Invoice.java`. The code defines a class `Invoice` annotated with `@Entity` and `@Id`. It has a private field `InvoiceID` and a private field `InvoiceNumber`. A many-to-many relationship is defined with `@ManyToMany`, mapped by the association name "invoices", with cascade type `PERSIST`. The class contains a constructor, three getters, and one setter for the `products` field.

```
4  @Entity
5  public class Invoice {
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.AUTO)
9      private int InvoiceID;
10     private int InvoiceNumber;
11     private int Quantity;
12
13     @ManyToMany(mappedBy = "invoices", cascade = CascadeType.PERSIST)
14     private Set<Product> products;
15
16     public Invoice(){
17     }
18
19     public Invoice(int InvoiceNumber, int Quantity, Set<Product> products){
20         this.InvoiceNumber = InvoiceNumber;
21         this.Quantity = Quantity;
22         this.products = products;
23     }
24
25     public int getInvoiceID() {
26         return InvoiceID;
27     }
28
29     public int getInvoiceNumber() {
30         return InvoiceNumber;
31     }
32
33     public int getQuantity() {
34         return Quantity;
35     }
}
```

## Klasa Product



```
Product.java X
4  @Entity
5  public class Product {
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.AUTO)
9      private int ProductID;
10     private String ProductName;
11     private int UnitsInStock;
12
13     @ManyToOne
14     @JoinColumn(name="SUPPLIER_FK")
15     private Supplier supplier;
16
17     @ManyToOne
18     @JoinColumn(name="CATEGORY_FK")
19     private Category category;
20
21     @ManyToMany(cascade = CascadeType.PERSIST)
22     private Set<Invoice> invoices;
23
24
25     public Product(String ProductName, int UnitsInStock, Supplier supplier, Category category,
26                     Set<Invoice> invoices){
27         this.ProductName=ProductName;
28         this.UnitsInStock=UnitsInStock;
29         this.supplier=supplier;
30         this.category=category;
31         this.invoices=invoices;
32     }
33
34     public String getProductName() { return ProductName; }
```

## Klasa sterująca *Main*

Zakomentowałem dodanie produktów nr 2 i 3 oraz faktur nr 2 i 3

```
1 Main.java
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
```

The code in the screenshot is a Java file named Main.java. It contains several annotations: yellow question marks on lines 64, 65, 67, 68, 71, 72, 73, 75, 76, 78, 79, 81, 82, 83, 85, 86, 89, 90, 91, 92, 93, and 94; and a blue bracket icon on line 84. The code itself is as follows:

```
productSetForSupplier1.add(product2);
productSetForSupplier2.add(product3);

productListForCategory1.add(product1);
productListForCategory1.add(product2);
productListForCategory2.add(product3);

em.persist(product1);
//em.persist(product2);
//em.persist(product3);

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

em.persist(category1);
em.persist(category2);

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

/*
Invoice invoice = em.find(Invoice.class, 8);
for(Product product : invoice.getProducts())
    System.out.println("Product Name: " + product.getProductName());

Product product = em.find(Product.class, 1);
for(Invoice inv : product.getInvoices())
    System.out.println("Invoice ID: " + inv.getInvoiceID() +
        " Invoice Number: " + inv.getInvoiceNumber());

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

## Uruchomienie programu

```
Run: Main (1) ×

Hibernate:
    /* update
       Product */ update
       Product
       set
           ProductName=?,
           UnitsInStock=?,
           CATEGORY_FK=?,
           SUPPLIER_FK=?
       where
           ProductID=?

Hibernate:
    /* insert collection
       row Product.invoices */ insert
       into
           Product_Invoice
           (products_ProductID, invoices_InvoiceID)
       values
           (?, ?)

Hibernate:
    /* insert collection
       row Product.invoices */ insert
       into
           Product_Invoice
           (products_ProductID, invoices_InvoiceID)
       values
```

## Schemat bazy danych

Database

jdbc:derby://127.0.0.1/BKordekJPA 1 of 11

APP

CATEGORY

- KEY CATEGORYID INTEGER
- NAME VARCHAR(255)
- SQL210103022759690 (CATEGORYID)
- SQL210103022759690 (CATEGORYID) UNIQUE

INVOICE

- KEY INVOICEID INTEGER
- INVOICENUMBER INTEGER
- QUANTITY INTEGER
- SQL210103022759720 (INVOICEID)
- SQL210103022759720 (INVOICEID) UNIQUE

PRODUCT

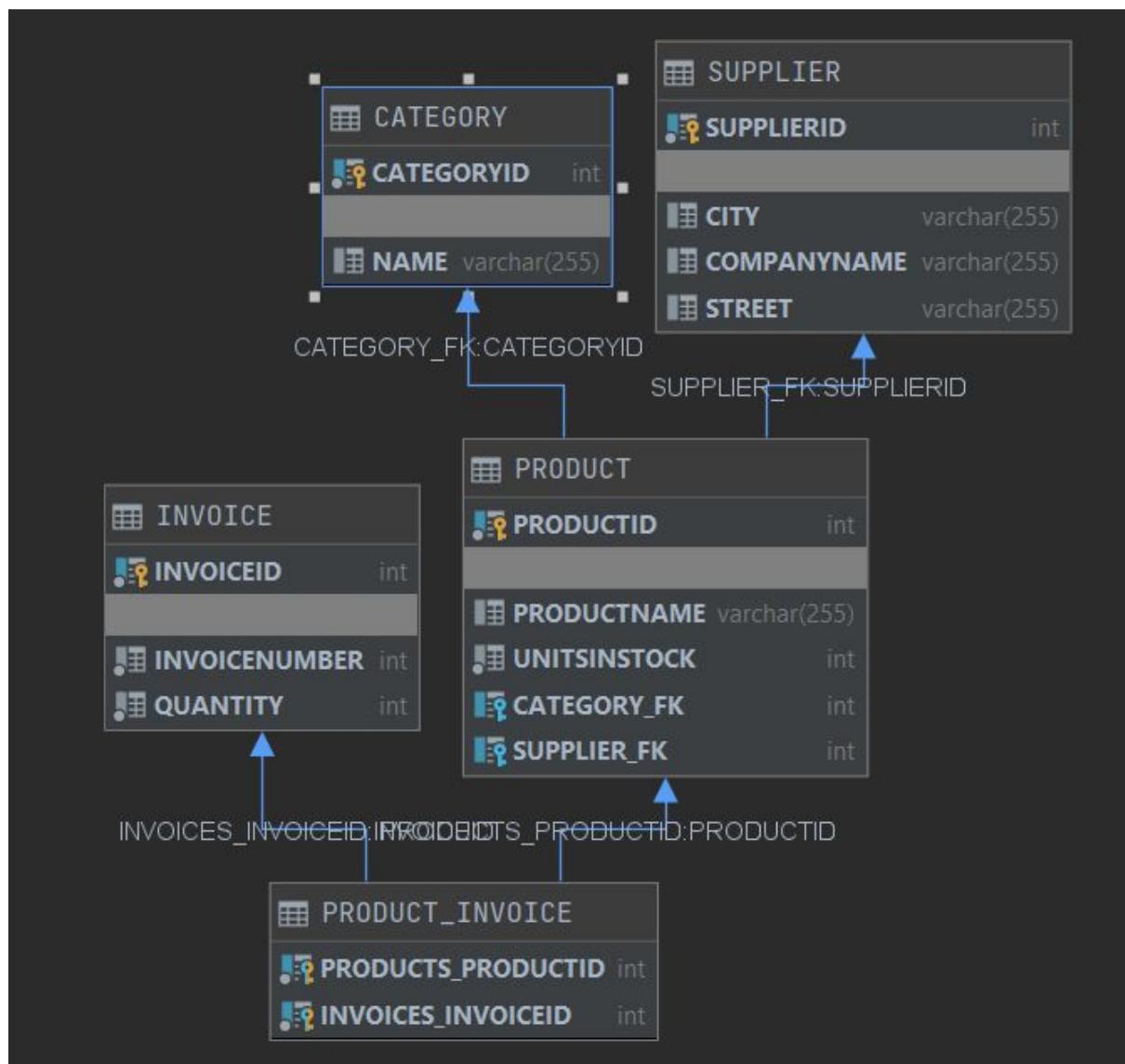
- KEY PRODUCTID INTEGER
- PRODUCTNAME VARCHAR(255)
- UNITSINSTOCK INTEGER
- KEY CATEGORY\_FK INTEGER
- KEY SUPPLIER\_FK INTEGER
- SQL210103022759740 (PRODUCTID)
- FKEURY2HXL2J8URLKMW36585TKR (SUPPLIER\_FK) → SUPPLIER (SUPPLIERID)
- FKPURGJJ563MV2VAV0MGGDXEFD7 (CATEGORY\_FK) → CATEGORY (CATEGORYID)
- SQL210103022759740 (PRODUCTID) UNIQUE
- SQL210103022759810 (CATEGORY\_FK)
- SQL210103022759840 (SUPPLIER\_FK)

PRODUCT\_INVOICE

- KEY PRODUCTS\_PRODUCTID INTEGER
- KEY INVOICES\_INVOICEID INTEGER
- SQL210103022759760 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID)
- FK30TSIIUDMV7Y4P1360MNQ4V7R (INVOICES\_INVOICEID) → INVOICE (INVOICEID)
- FKMCMOHEDFHSUOR6LTI20A304QD (PRODUCTS\_PRODUCTID) → PRODUCT (PRODUCTID)
- SQL210103022759760 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID) UNIQUE
- SQL210103022759870 (INVOICES\_INVOICEID)
- SQL210103022759890 (PRODUCTS\_PRODUCTID)

SUPPLIER

- KEY SUPPLIERID INTEGER
- CITY VARCHAR(255)
- COMPANYNAME VARCHAR(255)
- STREET VARCHAR(255)
- SQL210103022759780 (SUPPLIERID)
- SQL210103022759780 (SUPPLIERID) UNIQUE



## Zawartość bazy danych

The screenshot shows four database tables in a tool like DBeaver:

- APP.PRODUCT** [jdbc:derby://127.0.0.1/BKordekJPA] (3 rows):
 

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Frytki	20	9	7
2	3	Chipsy	50	10	8
3	6	Pizza	10	9	7
- APP.CATEGORY** [jdbc:derby://127.0.0.1/BKordekJPA] (2 rows):
 

	CATEGORYID	NAME
1	9	Mrożonki
2	10	Przekąski
- APP.PRODUCT\_INVOICE** [jdbc:derby://127.0.0.1/BKordekJPA] (6 rows):
 

	PRODUCTS_PRODUCTID	INVOICES_INVOICEID
1	1	2
2	1	5
3	3	2
4	3	4
5	3	5
6	6	5
- APP.SUPPLIER** [jdbc:derby://127.0.0.1/BKordekJPA] (2 rows):
 

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	7	Kraków	Tesco	Kapelanka
2	8	Kraków	Żabka	Sołtysowka

Jak widać, mimo że produkty nr 2 i 3 oraz faktury nr 2 i 3 nie były bezpośrednio dodane do bazy, ale dzięki mechanizmowi kaskad, zostały zapisane w bazie (wystarczyło tylko wywołanie persist na produkcie nr 1 i fakturze nr 1).

## XII. Embedded class

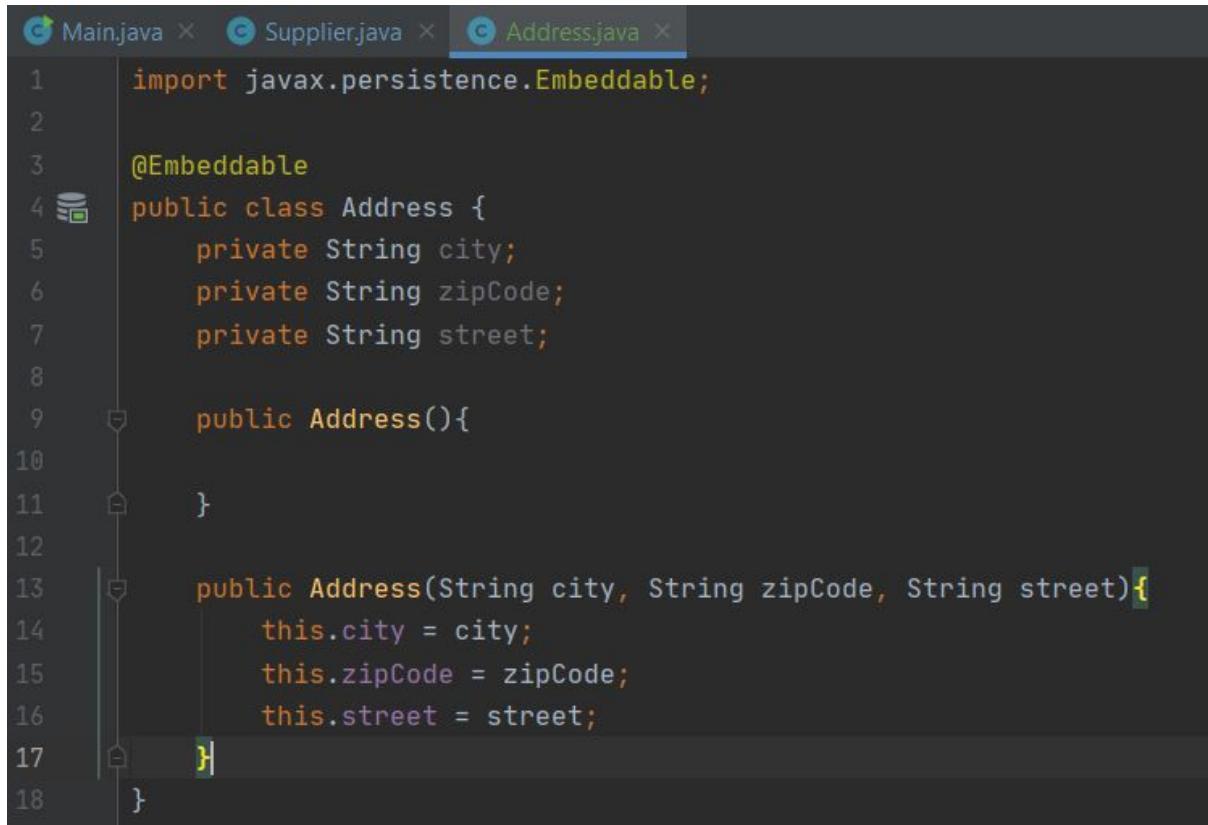
Dodaj do modelu klasę adres. „Wbuduj” ją do tabeli Dostawców

Klasa *Supplier* - modyfikacja

The screenshot shows the Supplier.java file in an IDE. The code defines a Supplier entity with an ID, a collection of products, and an embedded Address object. Annotations are used for persistence and relationships.

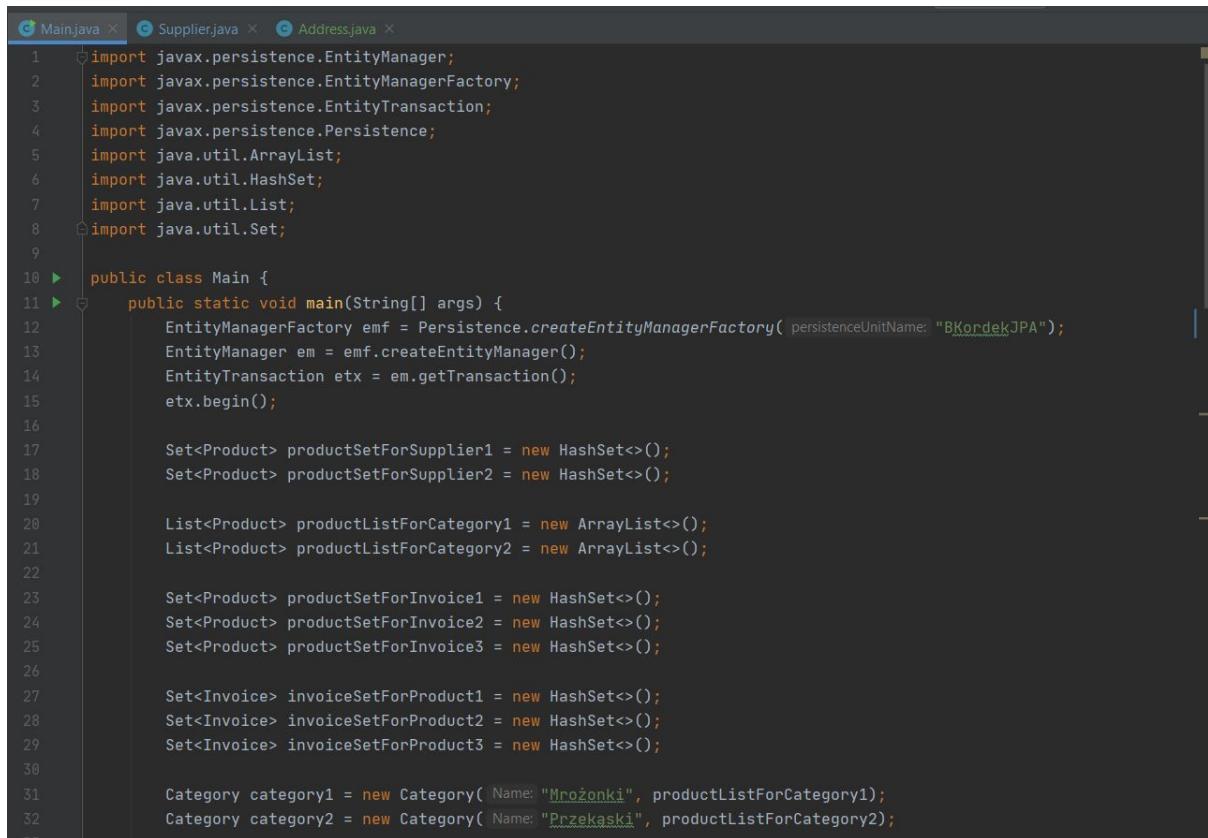
```
1 import javax.persistence.*;
2 import java.util.Set;
3
4 @Entity
5 public class Supplier {
6
7     @Id
8     @GeneratedValue(strategy = GenerationType.AUTO)
9     private int SupplierID;
10    private String CompanyName;
11    //private String Street;
12    //private String City;
13    @OneToMany
14    @JoinColumn(name="SUPPLIER_FK")
15    private Set<Product> products;
16    @Embedded
17    private Address address;
18
19    public Supplier(){
20    }
21
22
23
24    public Supplier(String CompanyName, Set<Product> products,
25                    Address address){
26        this.CompanyName = CompanyName;
27        //this.Street = Street;
28        //this.City = City;
29        this.products = products;
30        this.address = address;
31    }
32
33    public int getSupplierID() { return SupplierID; }
```

### Klasa Address - utworzenie mowej



```
1 import javax.persistence.Embeddable;
2
3 @Embeddable
4 public class Address {
5     private String city;
6     private String zipCode;
7     private String street;
8
9     public Address(){
10 }
11
12     public Address(String city, String zipCode, String street){
13         this.city = city;
14         this.zipCode = zipCode;
15         this.street = street;
16     }
17 }
18 }
```

### Klasa sterująca Main - dodanie adresów i przypisanie ich do poszczególnych dostawców



```
1 import javax.persistence.EntityManager;
2 import javax.persistence.EntityManagerFactory;
3 import javax.persistence.EntityTransaction;
4 import javax.persistence.Persistence;
5 import java.util.ArrayList;
6 import java.util.HashSet;
7 import java.util.List;
8 import java.util.Set;
9
10 public class Main {
11     public static void main(String[] args) {
12         EntityManagerFactory emf = Persistence.createEntityManagerFactory("BKordekJPA");
13         EntityManager em = emf.createEntityManager();
14         EntityTransaction etx = em.getTransaction();
15         etx.begin();
16
17         Set<Product> productSetForSupplier1 = new HashSet<>();
18         Set<Product> productSetForSupplier2 = new HashSet<>();
19
20         List<Product> productListForCategory1 = new ArrayList<>();
21         List<Product> productListForCategory2 = new ArrayList<>();
22
23         Set<Product> productSetForInvoice1 = new HashSet<>();
24         Set<Product> productSetForInvoice2 = new HashSet<>();
25         Set<Product> productSetForInvoice3 = new HashSet<>();
26
27         Set<Invoice> invoiceSetForProduct1 = new HashSet<>();
28         Set<Invoice> invoiceSetForProduct2 = new HashSet<>();
29         Set<Invoice> invoiceSetForProduct3 = new HashSet<>();
29
30         Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
31         Category category2 = new Category( Name: "Przekąski", productListForCategory2);
```

```
Main.java × Supplier.java × Address.java ×
34 Address supplier1Address = new Address( city: "Kraków", zipCode: "30-226", street: "Kapelanka");
35 Address supplier2Address = new Address( city: "Kraków", zipCode: "31-589", street: "Sotyssowka");
36
37 Supplier supplier1 = new Supplier( CompanyName: "Tesco", productSetForSupplier1, supplier1Address);
38 Supplier supplier2 = new Supplier( CompanyName: "Zabka", productSetForSupplier2, supplier2Address);
39
40 Invoice invoice1 = new Invoice( InvoiceNumber: 123, Quantity: 10, productSetForInvoice1);
41 Invoice invoice2 = new Invoice( InvoiceNumber: 124, Quantity: 10, productSetForInvoice2);
42 Invoice invoice3 = new Invoice( InvoiceNumber: 125, Quantity: 10, productSetForInvoice3);
43
44 Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20, supplier1, category1, invoiceSetForProduct1);
45 Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10, supplier1, category1, invoiceSetForProduct2);
46 Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50, supplier2, category2, invoiceSetForProduct3);
47
48 productSetForInvoice1.add(product1);
49 productSetForInvoice1.add(product2);
50 productSetForInvoice1.add(product3);
51
52 productSetForInvoice2.add(product1);
53 productSetForInvoice2.add(product3);
54
55 productSetForInvoice3.add(product3);
56
57 invoiceSetForProduct1.add(invoice1);
58 invoiceSetForProduct1.add(invoice2);
59
60 invoiceSetForProduct2.add(invoice1);
61
62 invoiceSetForProduct3.add(invoice1);
63 invoiceSetForProduct3.add(invoice2);
64 invoiceSetForProduct3.add(invoice3);
65
```

```
Main.java × Supplier.java × Address.java ×
70 productListForCategory1.add(product1);
71 productListForCategory1.add(product2);
72 productListForCategory2.add(product3);
73
74 em.persist(product1);
75 //em.persist(product2);
76 //em.persist(product3);
77
78 em.persist(supplier1);
79 em.persist(supplier2);
80
81 em.persist(category1);
82 em.persist(category2);
83
84 em.persist(invoice1);
85 //em.persist(invoice2);
86 //em.persist(invoice3);
87 /*
88 Invoice invoice = em.find(Invoice.class, 8);
89 for(Product product : invoice.getProducts())
90     System.out.println("Product Name: " + product.getProductName());
91
92 Product product = em.find(Product.class, 1);
93 for(Invoice inv : product.getInvoices())
94     System.out.println("Invoice ID: " + inv.getInvoiceID() +
95                         " Invoice Number: " + inv.getInvoiceNumber());
96 */
97 emtx.commit();
98 em.close();
99 }
100 }
```

## Zmiany w klasie Main

```
33  
34     Address supplier1Address = new Address( city: "Kraków", zipCode: "30-226", street: "Kapelanka");  
35     Address supplier2Address = new Address( city: "Kraków", zipCode: "31-589", street: "Sołtysowka");  
36  
37     Supplier supplier1 = new Supplier( CompanyName: "Tesco", productSetForSupplier1, supplier1Address);  
38     Supplier supplier2 = new Supplier( CompanyName: "Żabka", productSetForSupplier2, supplier2Address);  
39
```

## Uruchomienie programu



The screenshot shows the 'Run' configuration window in an IDE. The title bar says 'Run: Main (1)'. The configuration panel has several sections: 'Up', 'Down', 'Hibernate:', 'create table Product\_Invoice (products\_ProductID integer not null, invoices\_InvoiceID integer not null, primary key (products\_ProductID, invoices\_InvoiceID))', 'Hibernate:', 'create table Supplier (SupplierID integer not null, CompanyName varchar(255), city varchar(255), street varchar(255), zipCode varchar(255), primary key (SupplierID))', 'Hibernate:', 'alter table Product add constraint FKpurjj563mv2vav0mggdxe7d7 foreign key (CATEGORY\_FK) references Category', and 'Hibernate:'.

```
Run: Main (1)  
  
Up ↓ Hibernate:  
create table Product_Invoice (  
    products_ProductID integer not null,  
    invoices_InvoiceID integer not null,  
    primary key (products_ProductID, invoices_InvoiceID)  
)  
Hibernate:  
  
create table Supplier (  
    SupplierID integer not null,  
    CompanyName varchar(255),  
    city varchar(255),  
    street varchar(255),  
    zipCode varchar(255),  
    primary key (SupplierID)  
)  
Hibernate:  
  
alter table Product  
    add constraint FKpurjj563mv2vav0mggdxe7d7  
        foreign key (CATEGORY_FK)  
            references Category  
Hibernate:
```

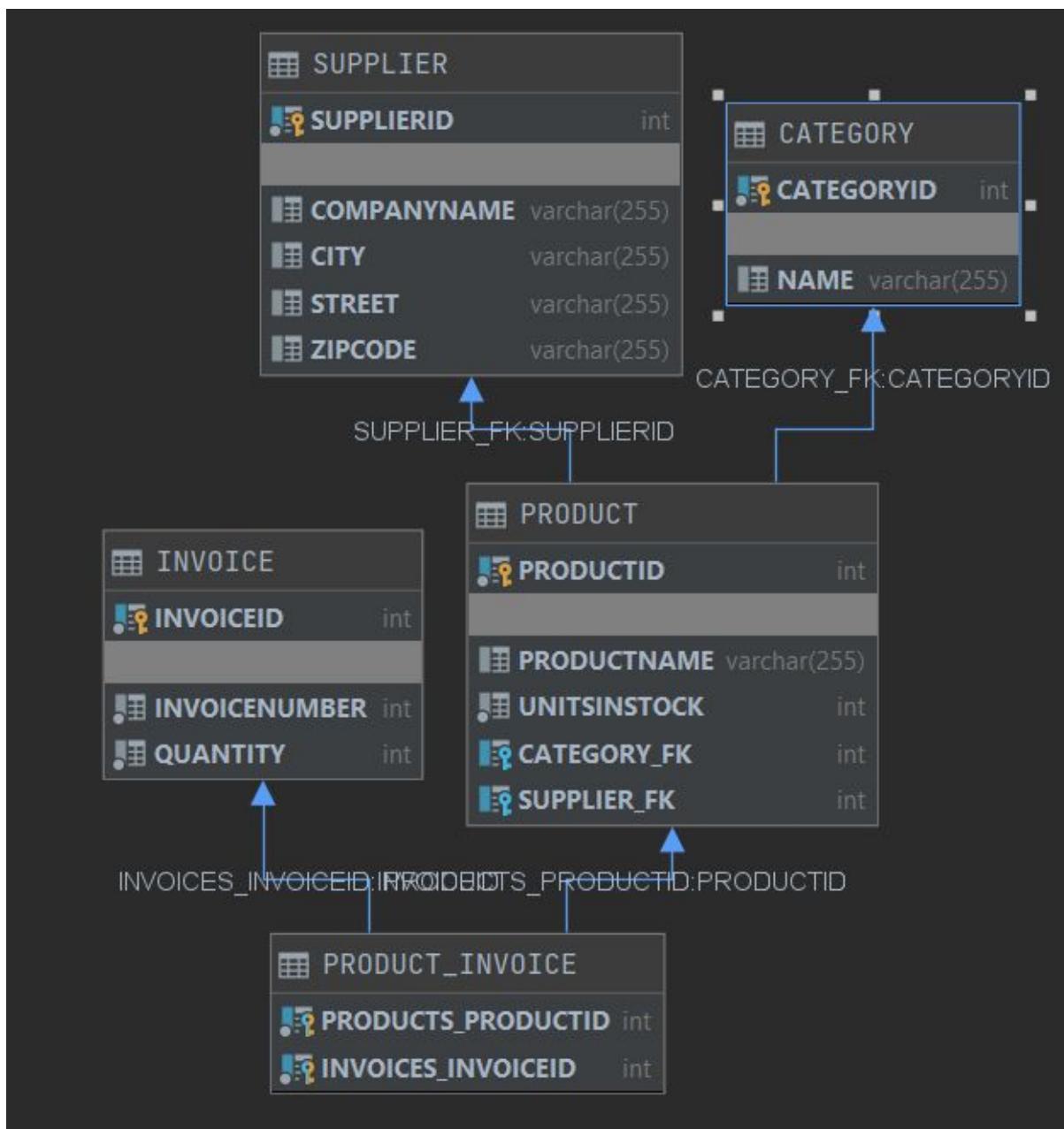
## Schemat bazy danych

Database

jdbc:derby://127.0.0.1/BKordekJPA [1 of 11]

APP

- ▼ **CATEGORY**
  - **CATEGORYID** INTEGER  
■ **NAME** VARCHAR(255)
    - SQL210103173053430 (CATEGORYID)
    - SQL210103173053430 (CATEGORYID) UNIQUE
- ▼ **INVOICE**
  - **INVOICEID** INTEGER
  - **INVOICENUMBER** INTEGER
  - **QUANTITY** INTEGER
    - SQL210103173053450 (INVOICEID)
    - SQL210103173053450 (INVOICEID) UNIQUE
- ▼ **PRODUCT**
  - **PRODUCTID** INTEGER
  - **PRODUCTNAME** VARCHAR(255)
  - **UNITSINSTOCK** INTEGER
  - **CATEGORY\_FK** INTEGER
    - SQL210103173053470 (PRODUCTID)
    - FKEURY2HXL2J8URLKMW36585TKR (SUPPLIER\_FK) → SUPPLIER (SUPPLIERID)
    - FKPURGJJ563MV2VAV0MGGDXEFD7 (CATEGORY\_FK) → CATEGORY (CATEGORYID)
    - SQL210103173053470 (PRODUCTID) UNIQUE
    - SQL210103173053550 (CATEGORY\_FK)
    - SQL210103173053560 (SUPPLIER\_FK)
- ▼ **PRODUCT\_INVOICE**
  - **PRODUCTS\_PRODUCTID** INTEGER
  - **INVOICES\_INVOICEID** INTEGER
    - SQL210103173053500 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID)
    - FK30TSIIUDMV7Y4P1360MNQ4V7R (INVOICES\_INVOICEID) → INVOICE (INVOICEID)
    - FKCMCOHEDFHSUOR6LTI20A304QD (PRODUCTS\_PRODUCTID) → PRODUCT (PRODUCTID)
    - SQL210103173053500 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID) UNIQUE
    - SQL210103173053580 (INVOICES\_INVOICEID)
    - SQL210103173053600 (PRODUCTS\_PRODUCTID)
- ▼ **SUPPLIER**
  - **SUPPLIERID** INTEGER
  - **COMPANYNAME** VARCHAR(255)
  - **CITY** VARCHAR(255)
  - **STREET** VARCHAR(255)
  - **ZIPCODE** VARCHAR(255)
    - SQL210103173053520 (SUPPLIERID)
    - SQL210103173053520 (SUPPLIERID) UNIQUE



Zawartość tabel

APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA]	
1	9 Mrożonki
<Filter Criteria>	
1	9 Mrożonki
2	10 Przekąski

APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA] ×

< < 3 rows > > | ↺ | + - | >> Comma...d (CSV) | ↓ ↑ | ⚡

Q <Filter Criteria>

	INVOICEID	INVOICENUMBER	QUANTITY
1	2	123	10
2	4	125	10
3	5	124	10

APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] ×

< < 3 rows > > | ↺ | + - | Tx: Auto | DB | ✓ | DDL | >> Comma...d (CSV) | ↓ ↑ | ⚡ APP.PROD

Q <Filter Criteria>

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Frytki	20	9	7
2	3	Chipsy	50	10	8
3	6	Pizza	10	9	7

APP.PRODUCT\_INVOICE [jdbc:derby://127.0.0.1/BKordekJPA] ×

< < 0 rows > > | ↺ | + - | DDL | ⚡ | Comma...d (

Q <Filter Criteria>

	PRODUCTS_PRODUCTID	INVOICES_INVOICEID
1	1	2
2	1	5
3	3	2
4	3	4
5	3	5
6	6	2

APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] ×

< < 2 rows > > | ↺ | + - | Tx: Auto | DB | ✓ | DDL | >> Comma...d (CSV) | ↓ ↑ | ⚡ APP

Q <Filter Criteria>

	SUPPLIERID	COMPANYNAME	CITY	STREET	ZIPCODE
1	7	Tesco	Kraków	Kapelanka	30-226
2	8	Żabka	Kraków	Sołtysówka	31-589

Została utworzona wspólna tabela dla klasy *Supplier* i *Address* o nazwie *SUPPLIER*.

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

Klasa *Supplier*

```
1 import javax.persistence.*;
2 import java.util.Set;
3
4
5 @Entity
6 @SecondaryTable(name = "ADDRESS")
7 public class Supplier {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.AUTO)
11    private int SupplierID;
12    private String CompanyName;
13    //private String Street;
14    //private String City;
15    @OneToMany
16    @JoinColumn(name="SUPPLIER_FK")
17    private Set<Product> products;
18    /*@Embedded
19    private Address address;*/
20    @Column(table = "ADDRESS")
21    private String city;
22    @Column(table = "ADDRESS")
23    private String zipCode;
24    @Column(table = "ADDRESS")
25    private String street;
26
27
28    public Supplier(){
29
30    }
```

```
31
32     public Supplier(String CompanyName, Set<Product> products,
33             String city, String zipCode, String street){
34         this.CompanyName = CompanyName;
35         //this.Street = Street;
36         //this.City = City;
37         this.products = products;
38         //this.address = address;
39         this.city = city;
40         this.zipCode = zipCode;
41         this.street = street;
42     }
43
44     public int getSupplierID() { return SupplierID; }
45
46     public String getCompanyName() { return CompanyName; }
47
48     @Override
49     public String toString() {
50         return "Supplier{" +
51                 "SupplierID=" + SupplierID +
52                 ", CompanyName='" + CompanyName + '\'' +
53                 ", products=" + products +
54                 '}';
55     }
56 }
```

Modyfikacja klasy sterującej *Main* - zakomentkowanie obiektów klasy *Address*, dodanie pól związanych z danymi adresowymi do obiektów klasy *Supplier*

```
//Address supplier1Address = new Address("Kraków", "30-226", "Kapelanka");
//Address supplier2Address = new Address("Kraków", "31-589", "Sołtysówka");

Supplier supplier1 = new Supplier(CompanyName: "Tesco", productSetForSupplier1,
                                 city: "Kraków", zipCode: "30-226", street: "Kapelanka");
Supplier supplier2 = new Supplier(CompanyName: "Żabka", productSetForSupplier2,
                                 city: "Kraków", zipCode: "31-589", street: "Sołtysówka");
```

## Uruchomienie programu

```
run: Main (1) ×

>      drop sequence hibernate_sequence restrict
| Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:

>      create table ADDRESS (
|         city varchar(255),
|         street varchar(255),
|         zipCode varchar(255),
|         SupplierID integer not null,
|         primary key (SupplierID)
|     )
sty 03, 2021 10:51:21 PM org.hibernate.resource.transaction.backend.jdbc.in
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hiber
Hibernate:

>      create table Category (
|         CategoryID integer not null,
|         Name varchar(255),
|         primary key (CategoryID)
|     )
Hibernate:

>      create table Invoice (
|         InvoiceID integer not null,
|         InvoiceNumber integer not null,
|         Quantity integer not null,
|         primary key (InvoiceID)
|     )
Hibernate:
```

```
Run: Main (1) ×

    create table Product (
        ProductID integer not null,
        ProductName varchar(255),
        UnitsInStock integer not null,
        CATEGORY_FK integer,
        SUPPLIER_FK integer,
        primary key (ProductID)
    )
Hibernate:

    create table Product_Invoice (
        products_ProductID integer not null,
        invoices_InvoiceID integer not null,
        primary key (products_ProductID, invoices_InvoiceID)
    )
Hibernate:

    create table Supplier (
        SupplierID integer not null,
        CompanyName varchar(255),
        primary key (SupplierID)
    )
Hibernate:

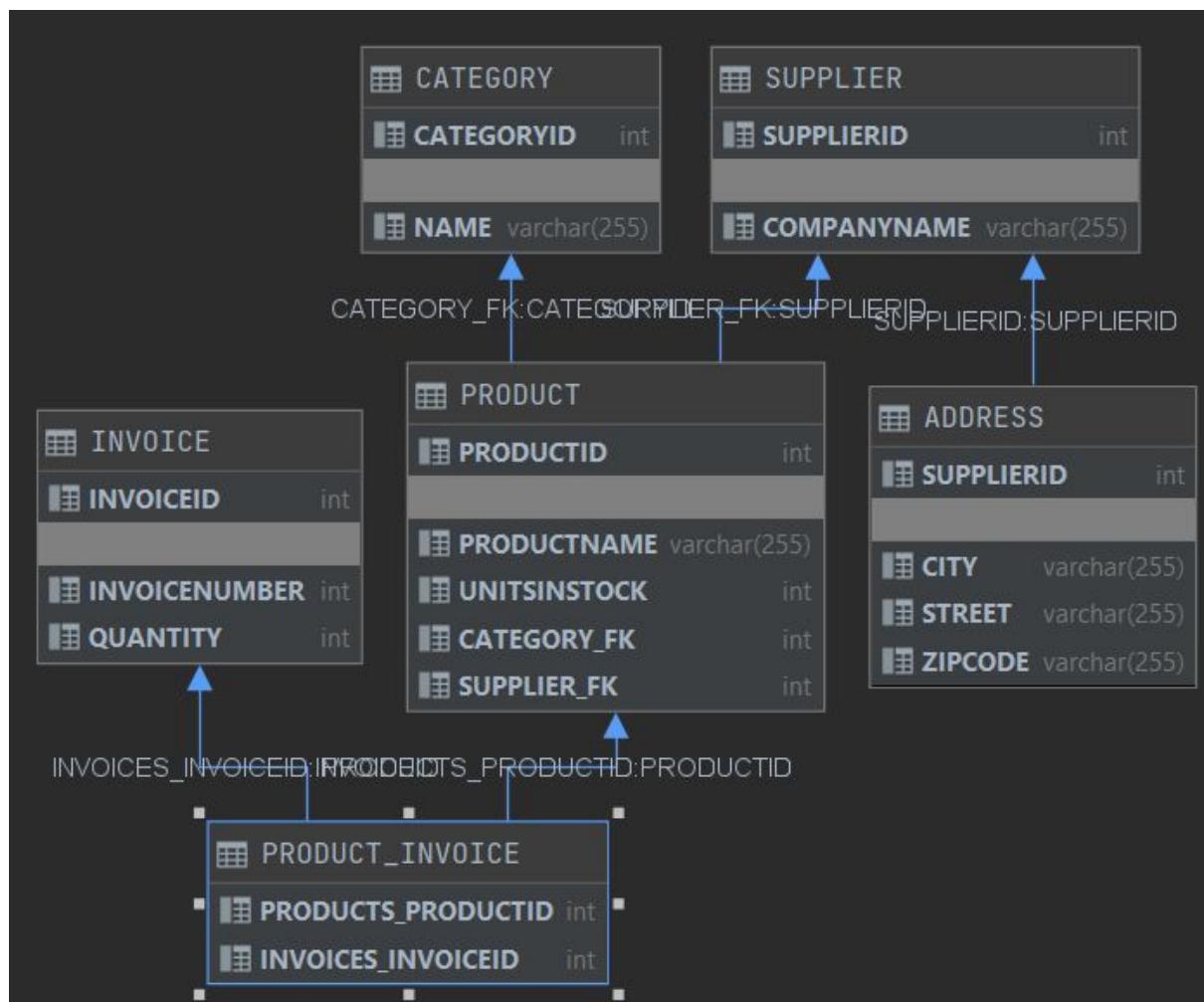
    alter table ADDRESS
        add constraint FK46ity49k53noswnl4quodoil
        foreign key (SupplierID)
        references Supplier
....
```

## Schemat bazy danych

Database jdbc:derby://127.0.0.1/BKordekJPA 1 of 11

APP

- ▼ ADDRESS
  - CITY VARCHAR(255)
  - STREET VARCHAR(255)
  - ZIPCODE VARCHAR(255)
  - SUPPLIERID INTEGER
  - SQL210103225121810 (SUPPLIERID)
  - FK46ITY49K53NOSWNL4AQUODO11 (SUPPLIERID) → SUPPLIER (SUPPLIERID)
  - SQL210103225121810 (SUPPLIERID) UNIQUE
  - SQL210103225121900 (SUPPLIERID)
- ▼ CATEGORY
  - CATEGORYID INTEGER
  - NAME VARCHAR(255)
  - SQL210103225121820 (CATEGORYID)
  - SQL210103225121820 (CATEGORYID) UNIQUE
- ▼ INVOICE
  - INVOICEID INTEGER
  - INVOICENUMBER INTEGER
  - QUANTITY INTEGER
  - SQL210103225121840 (INVOICEID)
  - SQL210103225121840 (INVOICEID) UNIQUE
- ▼ PRODUCT
  - PRODUCTID INTEGER
  - PRODUCTNAME VARCHAR(255)
  - UNITSINSTOCK INTEGER
  - CATEGORY\_FK INTEGER
  - SUPPLIER\_FK INTEGER
  - SQL210103225121850 (PRODUCTID)
  - FKEURY2HXL2J8URLKMW36585TKR (SUPPLIER\_FK) → SUPPLIER (SUPPLIERID)
  - FKPURGJJ563MV2AV0MGGDXEFD7 (CATEGORY\_FK) → CATEGORY (CATEGORYID)
  - SQL210103225121850 (PRODUCTID) UNIQUE
  - SQL210103225121910 (CATEGORY\_FK)
  - SQL210103225121920 (SUPPLIER\_FK)
- ▼ PRODUCT\_INVOICE
  - PRODUCTS\_PRODUCTID INTEGER
  - INVOICES\_INVOICEID INTEGER
  - SQL210103225121870 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID)
  - FK30TSIUDMV7Y4P1360MNQ4V7R (INVOICES\_INVOICEID) → INVOICE (INVOICEID)
  - FKMCMOHEDFHSUOR6LT120A304QD (PRODUCTS\_PRODUCTID) → PRODUCT (PRODUCTID)
  - SQL210103225121870 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID) UNIQUE
  - SQL210103225121940 (INVOICES\_INVOICEID)
  - SQL210103225121960 (PRODUCTS\_PRODUCTID)
- ▼ SUPPLIER
  - SUPPLIERID INTEGER
  - COMPANYNAME VARCHAR(255)
  - SQL210103225121880 (SUPPLIERID)
  - SQL210103225121880 (SUPPLIERID) UNIQUE



### Zawartość tabel

APP.ADDRESS [jdbc:derby://127.0.0.1/BKordekJPA]				
< 2 rows > Tx: Auto DDL Com				
Q <Filter Criteria>				
CITY	STREET	ZIPCODE	SUPPLIERID	
1 Kraków	Kapelanka	30-226		7
2 Kraków	Sołtysowka	31-589		8

APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA]	
< 2 rows > Tx: Auto	
Q <Filter Criteria>	
CATEGORYID	NAME
1	9 Mrożonki
2	10 Przekąski

APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA] ×

	INVOICEID	INVOICENUMBER	QUANTITY
1	2	123	10
2	5	125	10
3	6	124	10

APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] ×

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_FK	SUPPLIER_FK
1	1	Frytki	20	9	7
2	3	Pizza	10	9	7
3	4	Chipsy	50	10	8

APP.PRODUCT\_INVOICE [jdbc:derby://127.0.0.1/BKordekJPA] ×

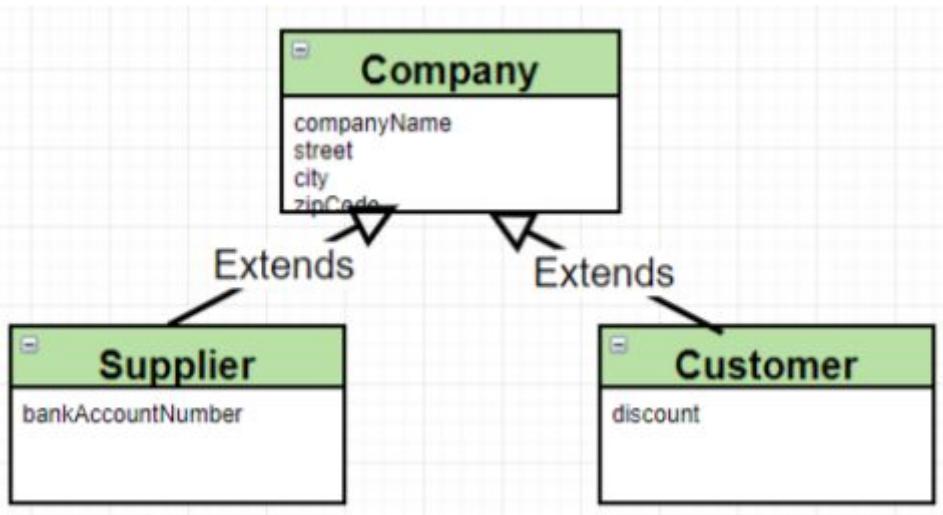
	PRODUCTS_PRODUCTID	INVOICES_INVOICEID
1	1	2
2	1	6
3	3	2
4	4	2
5	4	5
6	4	6

APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] ×

	SUPPLIERID	COMPANYNAME
1	7	Tesco
2	8	Żabka

Z powyższego widać, że została utworzona osobne tabele *ADDRESS* i *SUPPLIER*.

### XIII. Dziedziczenie

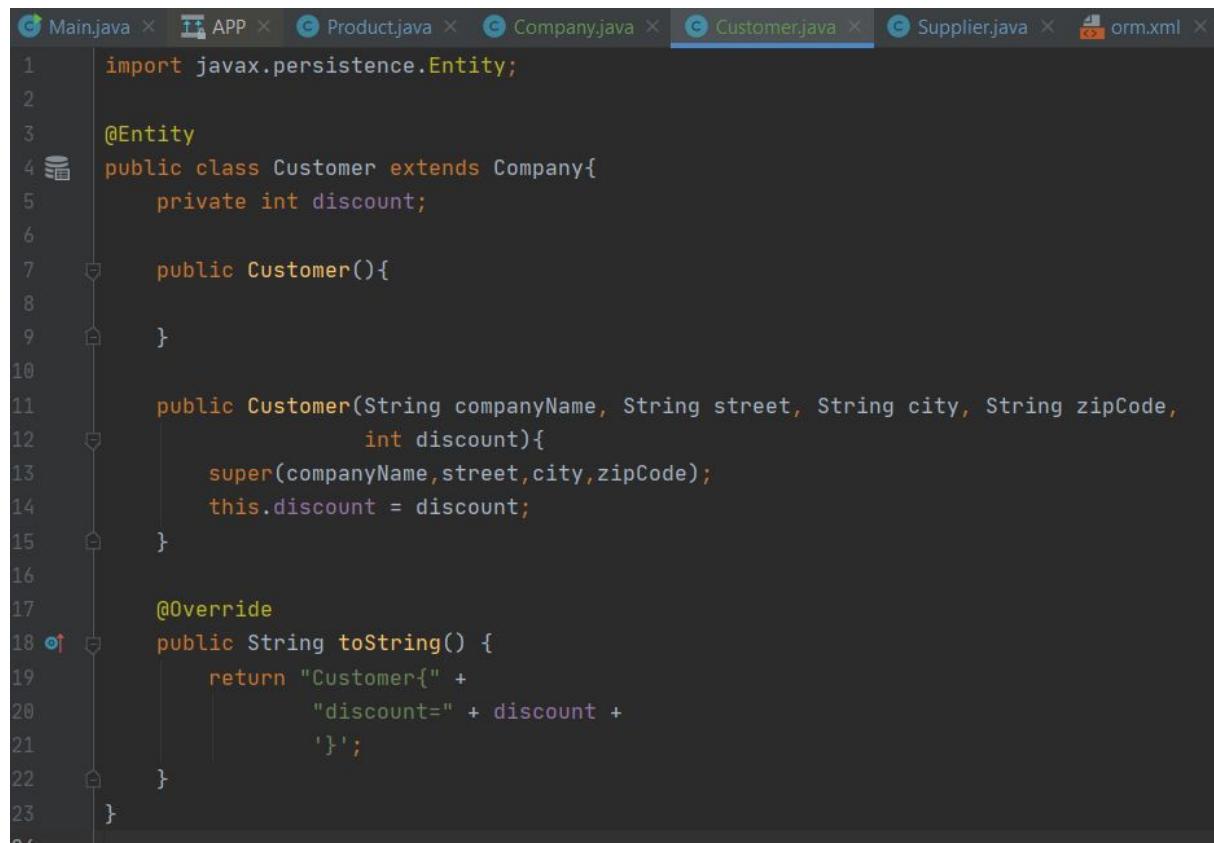


#### a) Single Table

Klasa Company

```
>Main.java × APP × Product.java × Company.java × Customer.java × Supplier.java × orm.xml ×
3   @Entity
4     @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
5       public class Company {
6         @Id
7           @GeneratedValue(strategy = GenerationType.AUTO)
8             private int companyId;
9             private String companyName;
10            private String street;
11            private String city;
12            private String zipCode;
13
14          public Company(){
15
16        }
17
18        public Company(String companyName, String street, String city, String zipCode){
19          this.companyName = companyName;
20          this.street = street;
21          this.city = city;
22          this.zipCode = zipCode;
23        }
24
25        @Override
26        public String toString() {
27          return "Company{" +
28              "companyId=" + companyId +
29              ", companyName='" + companyName + '\'' +
30              ", street='" + street + '\'' +
31              ", city='" + city + '\'' +
32              ", zipCode='" + zipCode + '\'' +
33              '}';
34      }
```

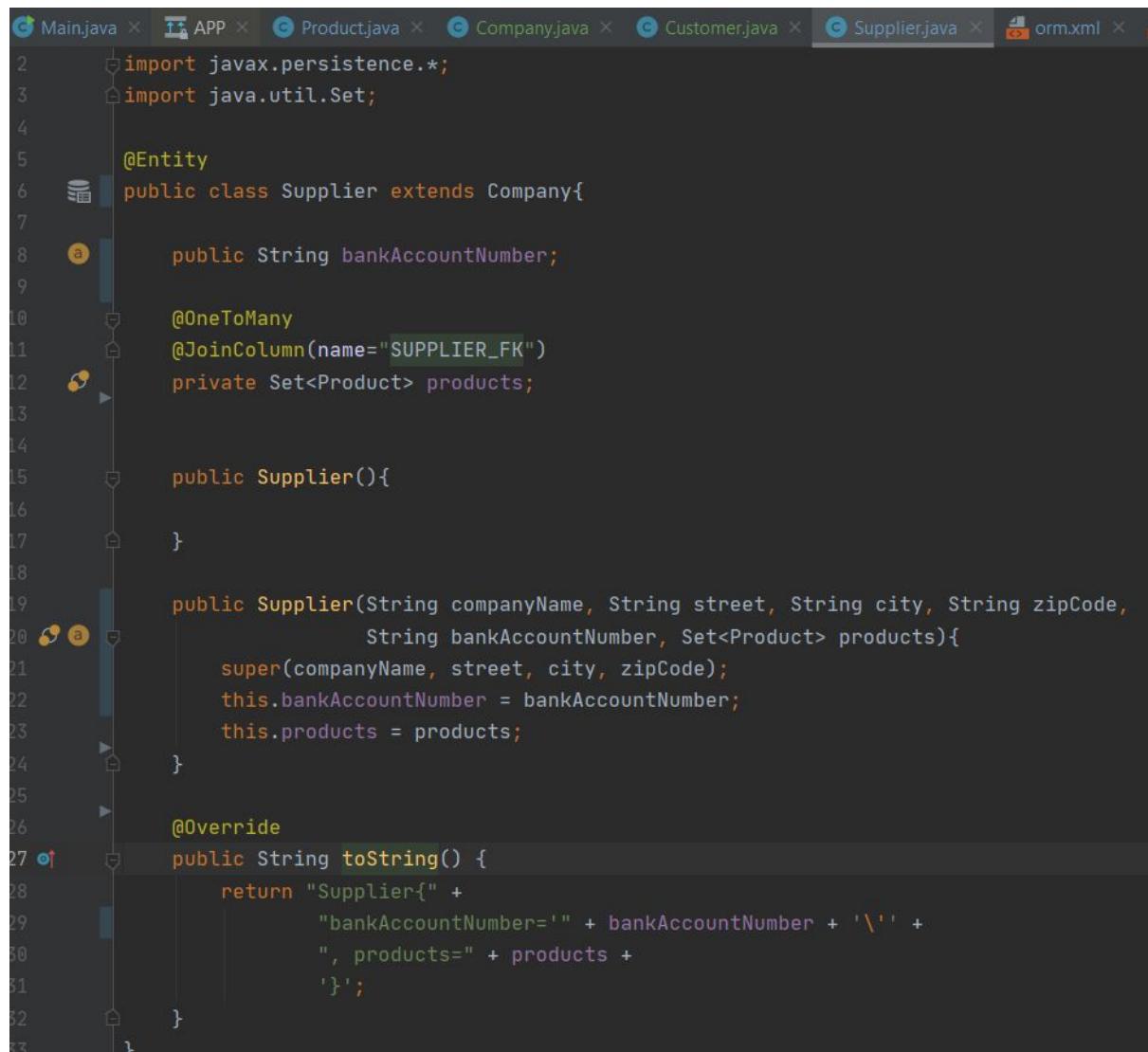
## Klasa Customer



The screenshot shows a Java code editor with the tab bar at the top containing files: Main.java, APP, Product.java, Company.java, Customer.java (which is the active tab), Supplier.java, and orm.xml. The code in the Customer.java editor is as follows:

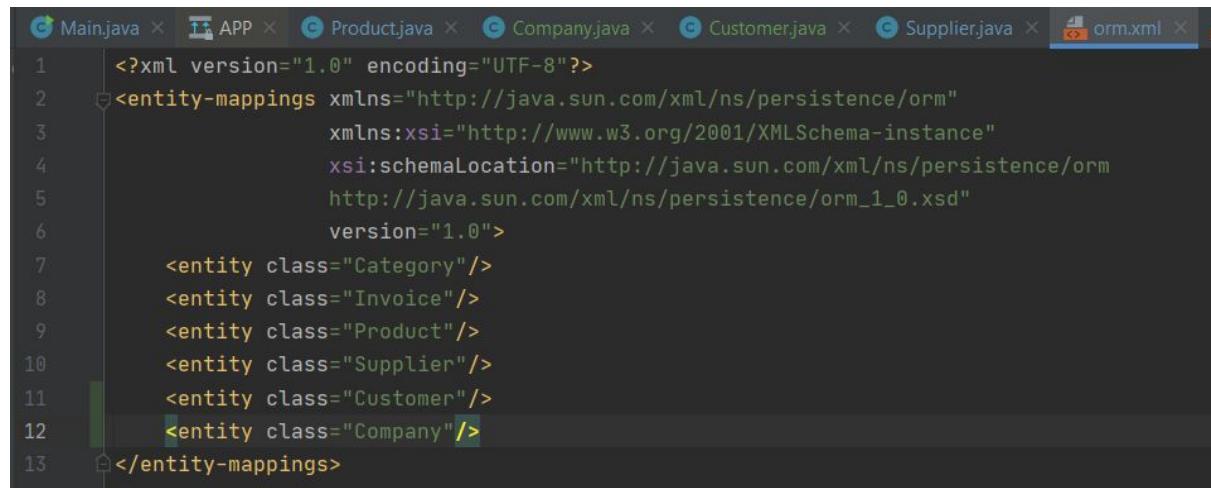
```
1 import javax.persistence.Entity;
2
3 @Entity
4 public class Customer extends Company{
5     private int discount;
6
7     public Customer(){
8
9     }
10
11    public Customer(String companyName, String street, String city, String zipCode,
12                    int discount){
13        super(companyName,street,city,zipCode);
14        this.discount = discount;
15    }
16
17    @Override
18    public String toString() {
19        return "Customer{" +
20               "discount=" + discount +
21               '}';
22    }
23}
```

## Klasa Supplier



```
2 import javax.persistence.*;
3 import java.util.Set;
4
5 @Entity
6 public class Supplier extends Company{
7
8     public String bankAccountNumber;
9
10    @OneToMany
11    @JoinColumn(name="SUPPLIER_FK")
12    private Set<Product> products;
13
14
15    public Supplier(){
16
17    }
18
19    public Supplier(String companyName, String street, String city, String zipCode,
20                    String bankAccountNumber, Set<Product> products){
21        super(companyName, street, city, zipCode);
22        this.bankAccountNumber = bankAccountNumber;
23        this.products = products;
24    }
25
26    @Override
27    public String toString() {
28        return "Supplier{" +
29                "bankAccountNumber='" + bankAccountNumber + '\'' +
30                ", products=" + products +
31                '}';
32    }
33}
```

## Modyfikacja pliku konfiguracyjnego - dodanie mapowania klasy Company



```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
                  version="1.0">
    <entity class="Category"/>
    <entity class="Invoice"/>
    <entity class="Product"/>
    <entity class="Supplier"/>
    <entity class="Customer"/>
    <entity class="Company"/>
</entity-mappings>
```

Modyfikacja klasy sterującej *Main* - modyfikacja obiektów dostawcy, stworzenie obiektów klienta

```
36  
37     Supplier supplier1 = new Supplier( companyName: "Tesco", street: "Kapelanka", city: "Kraków", zipCode: "30-226",  
38             bankAccountNumber: "340727370997063", productSetForSupplier1);  
39  
40     Supplier supplier2 = new Supplier( companyName: "Żabka", street: "Soltysowska", city: "Kraków", zipCode: "31-589",  
41             bankAccountNumber: "371989964645795", productSetForSupplier2);  
42  
43     Customer customer1 = new Customer( companyName: "Offices", street: "Długa", city: "Kraków", zipCode: "31-146",  
44             discount: 10);  
45  
46     Customer customer2 = new Customer( companyName: "Company", street: "Szeroka", city: "Kraków", zipCode: "30-146",  
47             discount: 5);  
48
```

Dodanie do bazy danych obiektów klienta

```
86  
87         em.persist(supplier1);  
88         em.persist(supplier2);  
89  
90         em.persist(customer1);  
91         em.persist(customer2);  
92
```

Uruchomienie programu

Run: Main (1) ×

Hibernate:

```
create table Company (
    DTYPENAME varchar(31) not null,
    companyID integer not null,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    zipCode varchar(255),
    bankAccountNumber varchar(255),
    discount integer,
    primary key (companyID)
)
```

Hibernate:

```
create table Invoice (
    InvoiceID integer not null,
    InvoiceNumber integer not null,
    Quantity integer not null,
    primary key (InvoiceID)
)
```

Hibernate:

```
create table Product (
    ProductID integer not null,
    ProductName varchar(255),
    UnitsInStock integer not null,
    CATEGORY_FK integer,
    SUPPLIER_FK integer,
    primary key (ProductID)
)
```

## Schemat bazy danych

Database

jdbc:derby://127.0.0.1/BKordekPA 1 of 11

APP

CATEGORY

- CATEGORYID INTEGER
- NAME VARCHAR(255)
- SQL210104204027050 (CATEGORYID)
- SQL210104204027050 (CATEGORYID) UNIQUE

COMPANY

- DTYPE VARCHAR(31)
- COMPANYID INTEGER
- CITY VARCHAR(255)
- COMPANYNAME VARCHAR(255)
- STREET VARCHAR(255)
- ZIPCODE VARCHAR(255)
- BANKACCOUNTNUMBER VARCHAR(255)
- DISCOUNT INTEGER
- SQL210104204027070 (COMPANYID)
- SQL210104204027070 (COMPANYID) UNIQUE

INVOICE

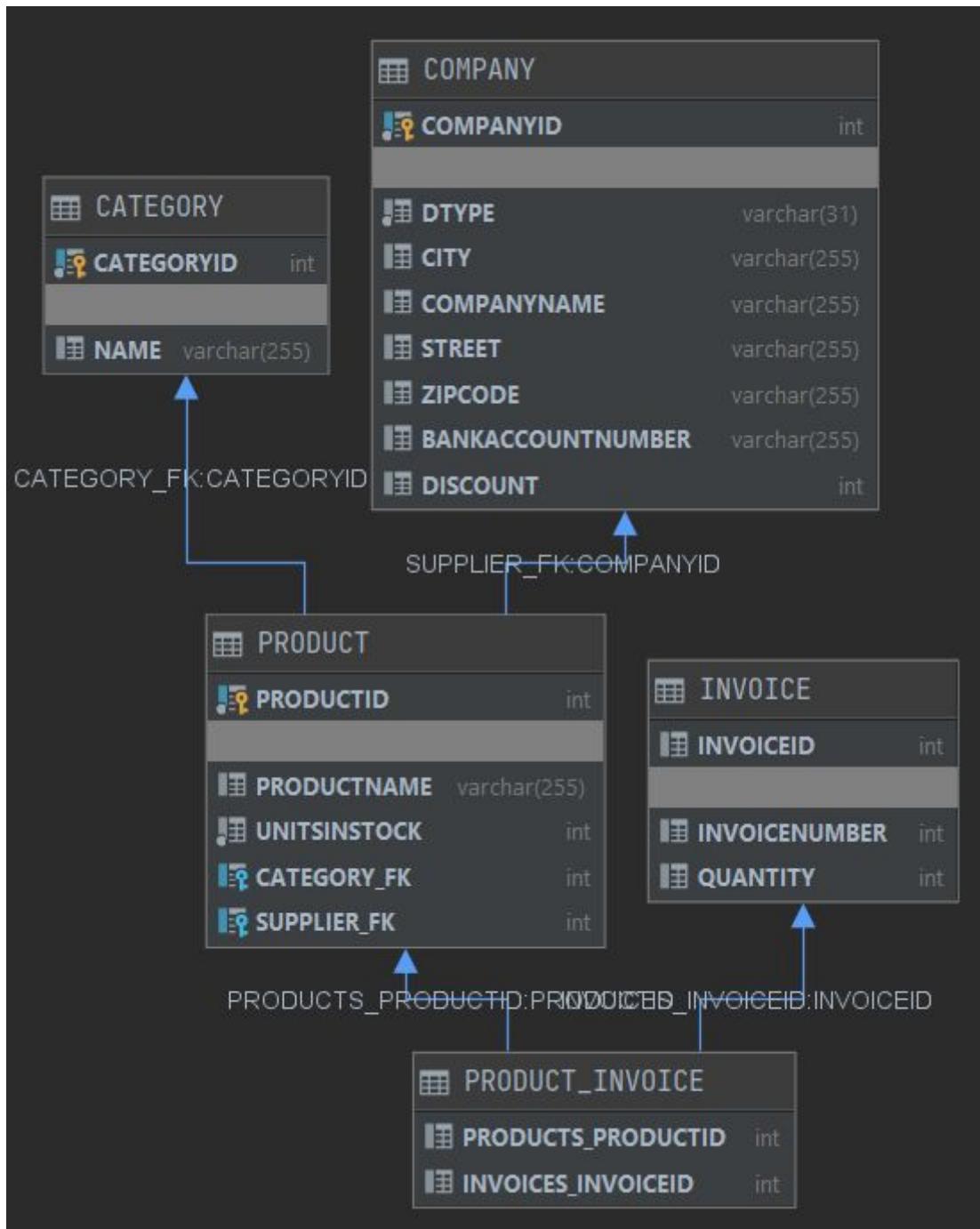
- INVOICEID INTEGER
- INVOICENUMBER INTEGER
- QUANTITY INTEGER
- SQL210104204027080 (INVOICEID)
- SQL210104204027080 (INVOICEID) UNIQUE

PRODUCT

- PRODUCTID INTEGER
- PRODUCTNAME VARCHAR(255)
- UNITSINSTOCK INTEGER
- CATEGORY\_FK INTEGER
- SUPPLIER\_FK INTEGER
- SQL210104204027100 (PRODUCTID)
- FKBHQ810AOUC906V8K7PXG1H290 (SUPPLIER\_FK) → COMPANY (COMPANYID)
- FKPURGGJ563MV2VAV0MGGDXEFD7 (CATEGORY\_FK) → CATEGORY (CATEGORYID)
- SQL210104204027100 (PRODUCTID) UNIQUE
- SQL210104204027140 (CATEGORY\_FK)
- SQL210104204027160 (SUPPLIER\_FK)

PRODUCT\_INVOICE

- PRODUCTS\_PRODUCTID INTEGER
- INVOICES\_INVOICEID INTEGER
- SQL210104204027110 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID)
- FK30TSIIUDMV7Y4P1360MNQ4V7R (INVOICES\_INVOICEID) → INVOICE (INVOICEID)
- FKCMCOHEDFHSUOR6LTI20A304QD (PRODUCTS\_PRODUCTID) → PRODUCT (PRODUCTID)
- SQL210104204027110 (PRODUCTS\_PRODUCTID, INVOICES\_INVOICEID) UNIQUE
- SQL210104204027170 (INVOICES\_INVOICEID)
- SQL210104204027190 (PRODUCTS\_PRODUCTID)



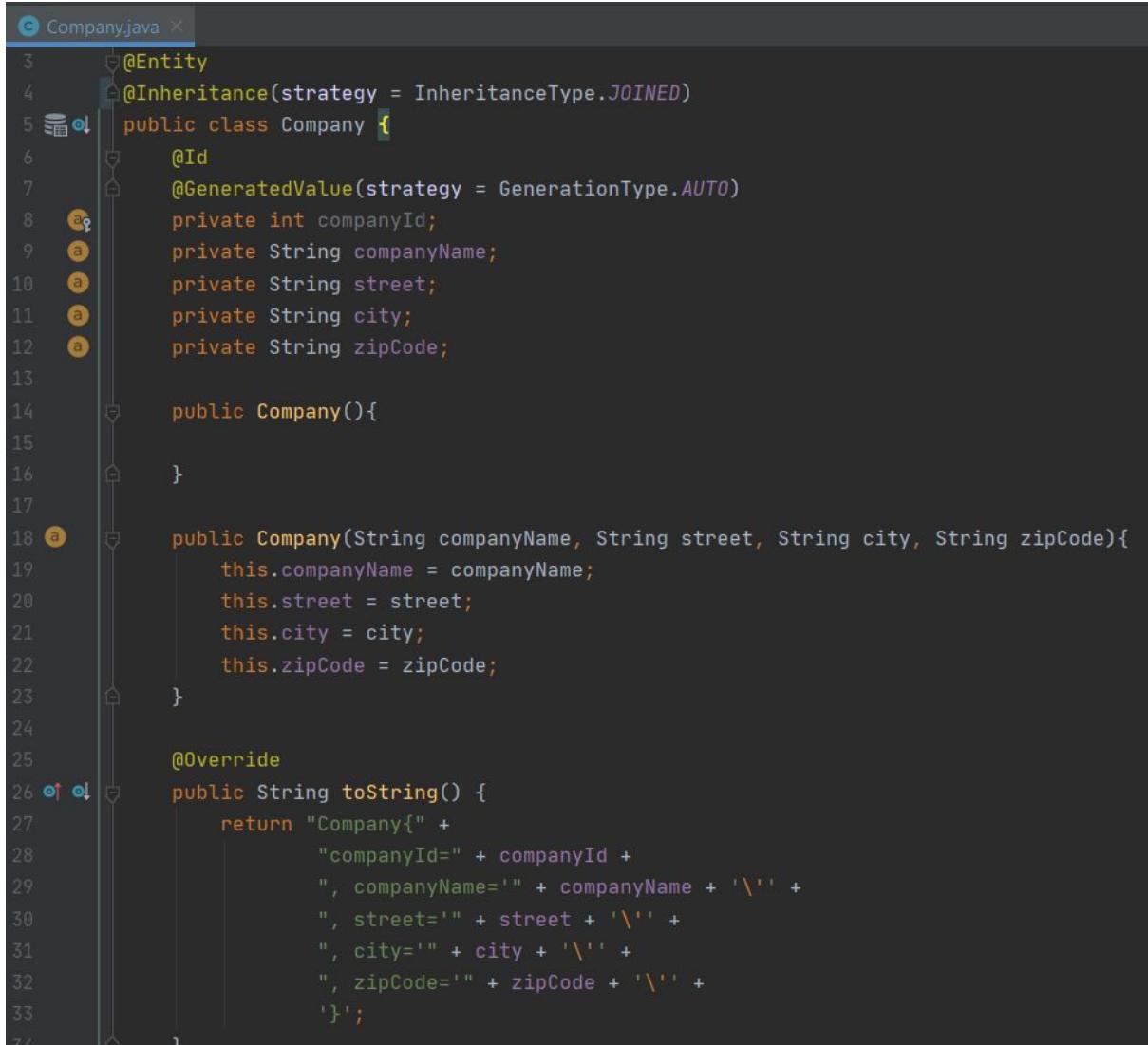
### Zawartość tabeli COMPANY

	DTYPE	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNTNUMBER	DISCOUNT
1	Supplier	7	Kraków	Tesco	Kapelanka	30-226	340727370997063	<null>
2	Supplier	8	Kraków	Żabka	Sołtysowska	31-589	371989964645795	<null>
3	Customer	9	Kraków	Offices	Długa	31-146	<null>	10
4	Customer	10	Kraków	Company	Szeroka	30-146	<null>	5

Została utworzona jedna wspólna tabela COMPANY dla klientów i dostawców.

## b. Joined

Modyfikacja klasy *Company* - zmiana typu dziedziczenia na **JOINED**



```
Company.java
3  @Entity
4  @Inheritance(strategy = InheritanceType.JOINED)
5  public class Company {
6      @Id
7      @GeneratedValue(strategy = GenerationType.AUTO)
8      private int companyId;
9      private String companyName;
10     private String street;
11     private String city;
12     private String zipCode;
13
14     public Company(){
15
16     }
17
18     public Company(String companyName, String street, String city, String zipCode){
19         this.companyName = companyName;
20         this.street = street;
21         this.city = city;
22         this.zipCode = zipCode;
23     }
24
25     @Override
26     public String toString() {
27         return "Company{" +
28             "companyId=" + companyId +
29             ", companyName='" + companyName + '\'' +
30             ", street='" + street + '\'' +
31             ", city='" + city + '\'' +
32             ", zipCode='" + zipCode + '\'' +
33             '}';
34     }
35 }
```

Uruchomienie programu

```
Run: Main (1) ×

    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 (
            discount integer not null,
            companyId integer not null,
            primary key (companyId)
        )
    Hibernate:

        create table Invoice (
            InvoiceID integer not null,
            InvoiceNumber integer not null,
            Quantity integer not null,
            primary key (InvoiceID)
        )
    Hibernate:

        create table Product (
            ProductID integer not null,
            ProductName varchar(255),
            UnitsInStock integer not null,
            CATEGORY_FK integer,
```

```
Hibernate:
```

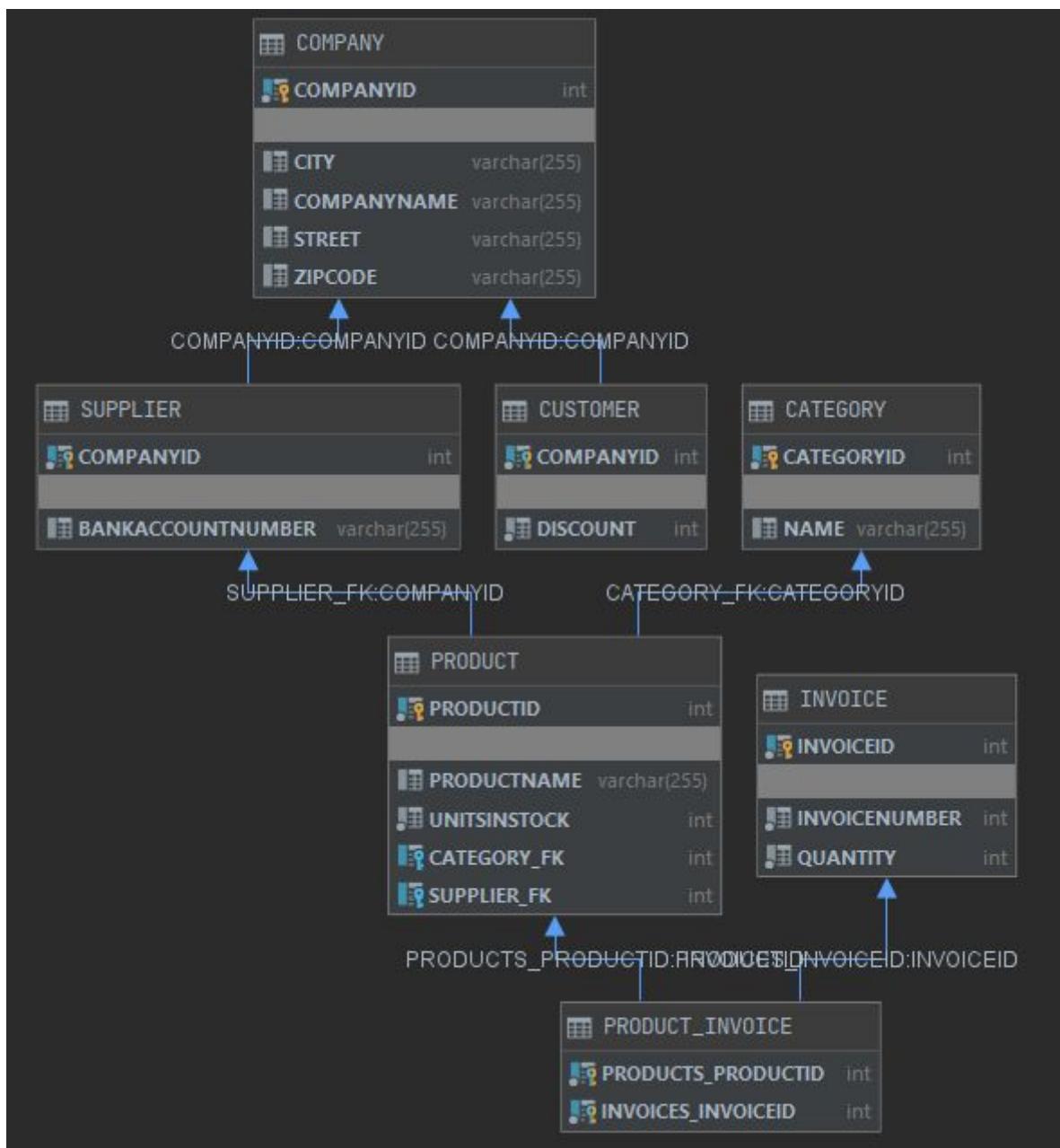
```
create table Supplier (
    bankAccountNumber varchar(255),
    companyId integer not null,
    primary key (companyId)
)
```

```
Hibernate:
```

```
alter table Customer
    add constraint FKq5b9xcmqp3uih4u11r37l6229
        foreign key (companyId)
        references Company
```

## Schemat bazy danych

jdbc:derby://127.0.0.1/BKordekJPA 1 of 11	
▼	APP
▼	CATEGORY
■	CATEGORYID INTEGER
■	NAME VARCHAR(255)
🔑	SQL210104210147200 (CATEGORYID)
↳	SQL210104210147200 (CATEGORYID) UNIQUE
▼	COMPANY
■	COMPANYID INTEGER
■	CITY VARCHAR(255)
■	COMPANYNAME VARCHAR(255)
■	STREET VARCHAR(255)
■	ZIPCODE VARCHAR(255)
🔑	SQL210104205915290 (COMPANYID)
↳	SQL210104205915290 (COMPANYID) UNIQUE
▼	CUSTOMER
■	DISCOUNT INTEGER
■	COMPANYID INTEGER
🔑	SQL210104210147230 (COMPANYID)
↳	FKQ5B9XCMQP3UIH4U11R37L6229 (COMPANYID) → COMPANY (COMPANYID)
↳	SQL210104210147230 (COMPANYID) UNIQUE
↳	SQL210104210147290 (COMPANYID)
▼	INVOICE
■	INVOICEID INTEGER
■	INVOICENUMBER INTEGER
■	QUANTITY INTEGER
🔑	SQL210104210147240 (INVOICEID)
↳	SQL210104210147240 (INVOICEID) UNIQUE
▼	PRODUCT
■	PRODUCTID INTEGER
■	PRODUCTNAME VARCHAR(255)
■	UNITSINSTOCK INTEGER
■	CATEGORY_FK INTEGER
■	SUPPLIER_FK INTEGER
🔑	SQL210104210147250 (PRODUCTID)
↳	FKEURY2HXL2J8URLKMW36585TKR (SUPPLIER_FK) → SUPPLIER (COMPANYID)
↳	FKPURGJJ563MV2AV0MGGDXEFD7 (CATEGORY_FK) → CATEGORY (CATEGORYID)
↳	SQL210104210147250 (PRODUCTID) UNIQUE
↳	SQL210104210147291 (CATEGORY_FK)
↳	SQL210104210147300 (SUPPLIER_FK)
▼	PRODUCT_INVOICE
■	PRODUCTS_PRODUCTID INTEGER
■	INVOICES_INVOICEID INTEGER
🔑	SQL210104210147260 (PRODUCTS_PRODUCTID, INVOICES_INVOICEID)
↳	FK30TSIUDMV7Y4P1360MNQ4V7R (INVOICES_INVOICEID) → INVOICE (INVOICEID)
↳	FKMCMOHEDFHJSUOR6LTI20A304QD (PRODUCTS_PRODUCTID) → PRODUCT (PRODUCTID)
↳	SQL210104210147260 (PRODUCTS_PRODUCTID, INVOICES_INVOICEID) UNIQUE
↳	SQL210104210147310 (INVOICES_INVOICEID)
↳	SQL210104210147320 (PRODUCTS_PRODUCTID)
▼	SUPPLIER
■	BANKACCOUNTNUMBER VARCHAR(255)
■	COMPANYID INTEGER
🔑	SQL210104210147270 (COMPANYID)
↳	FK43EKYFATN1CY2AW2U5FGM8ETW (COMPANYID) → COMPANY (COMPANYID)
↳	SQL210104210147270 (COMPANYID) UNIQUE
↳	SQL210104210147330 (COMPANYID)



### Zawartość tabeli COMPANY

APP.COMPANY [jdbc:derby://127.0.0.1/BKordekJPA] ×				
	4 rows	< >	< >	Tx: Auto
<i>Q▼ &lt;Filter Criteria&gt;</i>				
COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	7 Kraków	Tesco	Kapelanka	30-226
2	8 Kraków	Żabka	Sołtysowska	31-589
3	9 Kraków	Offices	Długa	31-146
4	10 Kraków	Company	Szeroka	30-146

Zawartość tabeli *CUSTOMER*

APP.CUSTOMER [jdbc:derby://127.0.0.1/BKordekJPA]		
<	2 rows	>
Q <Filter Criteria>		
	DISCOUNT	COMPANYID
1	10	9
2	5	10

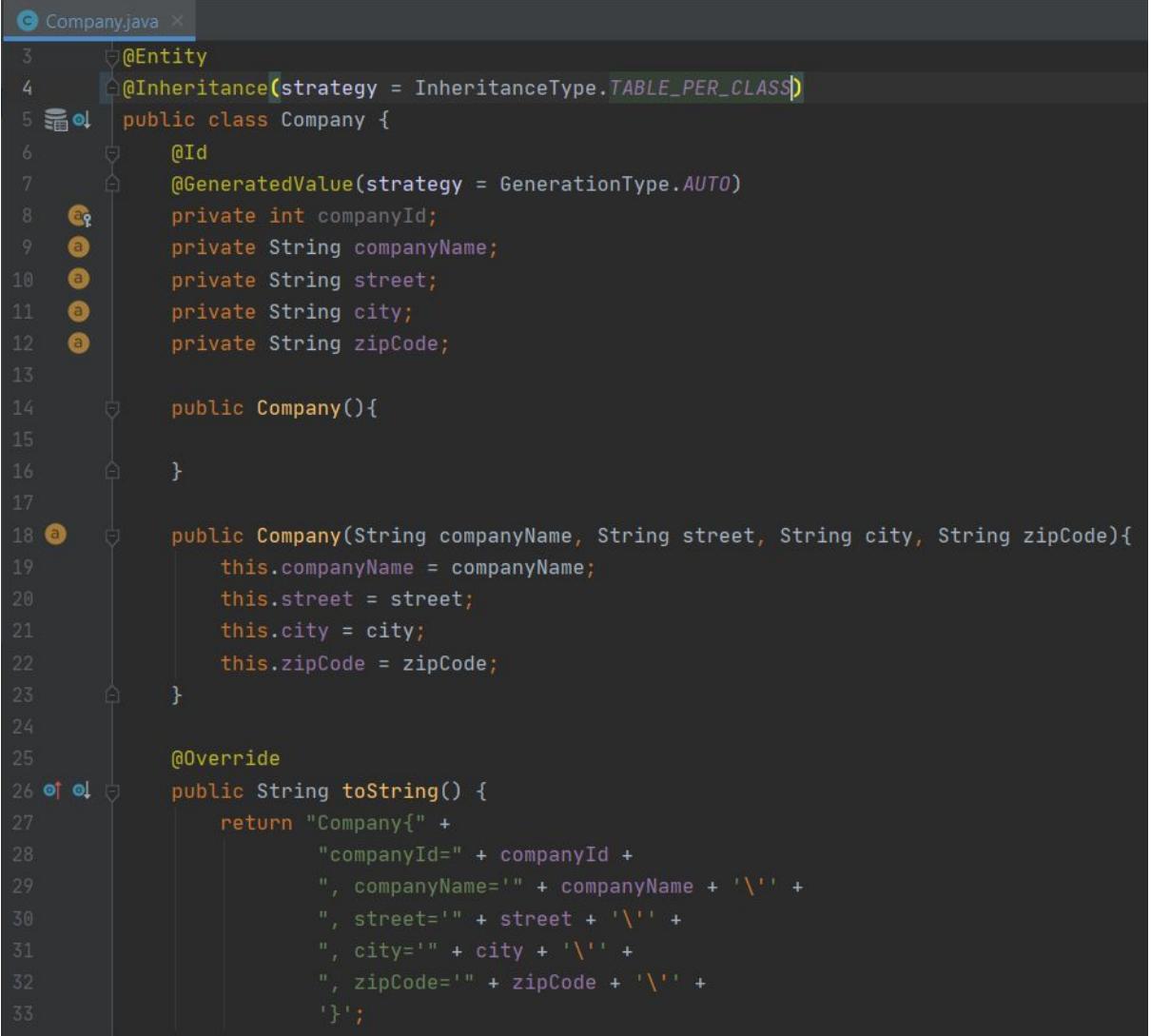
Zawartość tabeli *SUPPLIER*

APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA]		
<	2 rows	>
Q <Filter Criteria>		
	BANKACCOUNTNUMBER	COMPANYID
1	340727370997063	7
2	371989964645795	8

Z powyższego widać, że zostały utworzone 3 tabele *CUSTOMER*, *SUPPLIER* i tabela z której dziedziczą dane klientów i dostawców *COMPANY*. W tabeli *CUSTOMER* znajdują się rekordy zarówno dostawców jak i klientów z polami wspólnymi dla obu.

### c. Table per Class

Modyfikacja klasy *Company* - zmiana typu dziedziczenia na **TABLE\_PER\_CLASS**



The screenshot shows a code editor window for a Java file named `Company.java`. The code defines a class `Company` annotated with `@Entity` and `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)`. The class has four private fields: `companyName`, `street`, `city`, and `zipCode`, each annotated with `@Id` and `@GeneratedValue(strategy = GenerationType.AUTO)`. It includes a constructor that initializes these fields and an overridden `toString()` method that returns a string representation of the object's state.

```
3  @Entity
4  @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
5  public class Company {
6      @Id
7      @GeneratedValue(strategy = GenerationType.AUTO)
8      private int companyId;
9      private String companyName;
10     private String street;
11     private String city;
12     private String zipCode;
13
14     public Company(){
15     }
16
17
18     public Company(String companyName, String street, String city, String zipCode){
19         this.companyName = companyName;
20         this.street = street;
21         this.city = city;
22         this.zipCode = zipCode;
23     }
24
25     @Override
26     public String toString() {
27         return "Company{" +
28             "companyId=" + companyId +
29             ", companyName='" + companyName + '\'' +
30             ", street='" + street + '\'' +
31             ", city='" + city + '\'' +
32             ", zipCode='" + zipCode + '\'' +
33             '}';
34     }
35 }
```

## Uruchomienie programu

```
Run: Main (1) ×

Hibernate:
create table Category (
    CategoryID integer not null,
    Name varchar(255),
    primary key (CategoryID)
)
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,
    city varchar(255),
    companyName varchar(255),
    street varchar(255),
    zipCode varchar(255),
    discount integer not null,
    primary key (companyId)
)
```

```
create table Invoice (
    InvoiceID integer not null,
    InvoiceNumber integer not null,
    Quantity integer not null,
    primary key (InvoiceID)
)
Hibernate:

create table Product (
    ProductID integer not null,
    ProductName varchar(255),
    UnitsInStock integer not null,
    CATEGORY_FK integer,
    SUPPLIER_FK integer,
    primary key (ProductID)
)
Hibernate:

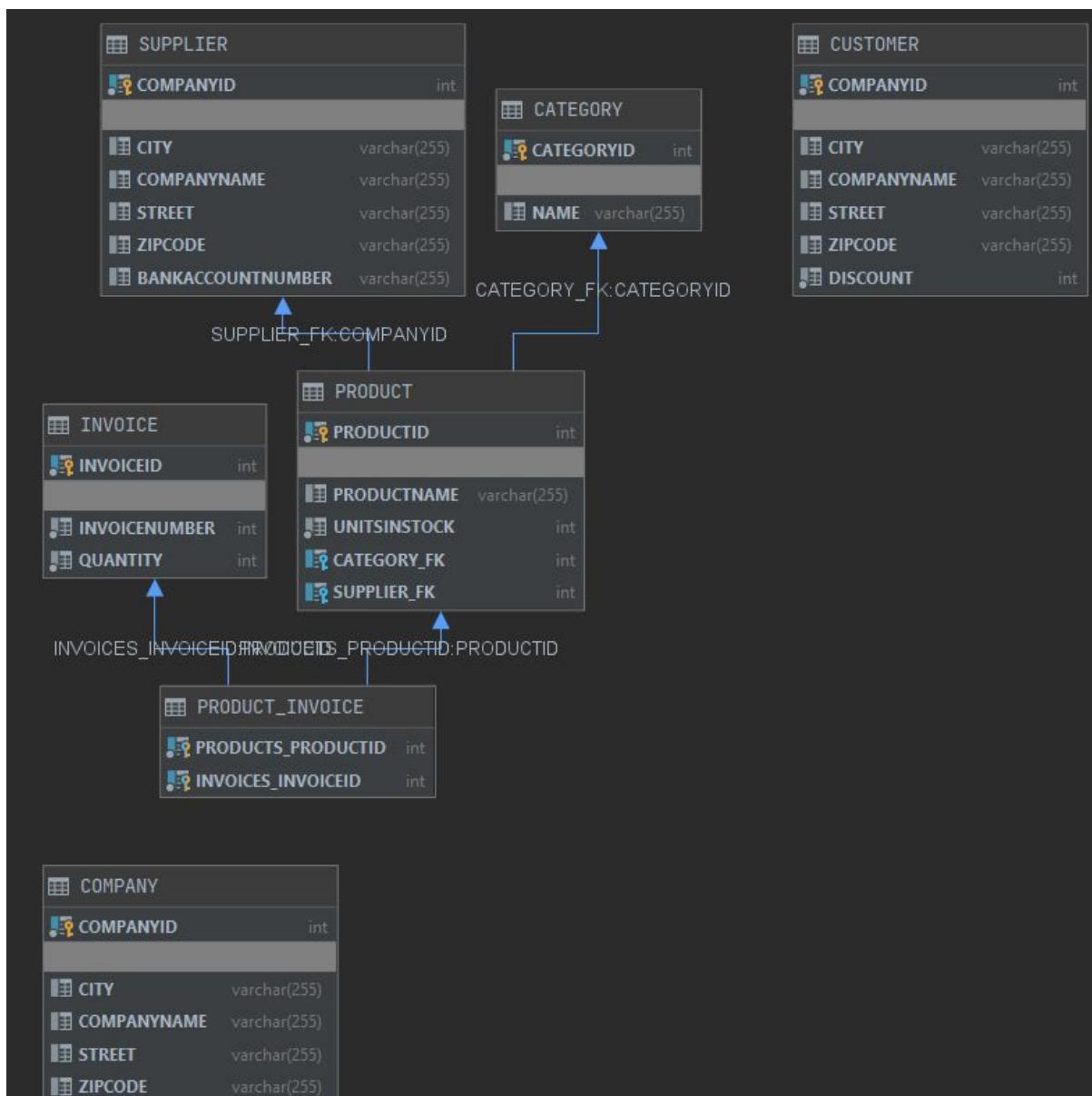
create table Product_Invoice (
    products_ProductID integer not null,
    invoices_InvoiceID integer not null,
    primary key (products_ProductID, invoices_InvoiceID)
)
```

```
Hibernate:

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

## Schemat bazy danych

jdbc:derby://127.0.0.1/BKordekJPA [1 of 11]	
▼	CATEGORY
	CATEGORYID INTEGER NAME VARCHAR(255) SQL210104211946270 (CATEGORYID) iu SQL210104211946270 (CATEGORYID) UNIQUE
▼	COMPANY
	COMPANYID INTEGER CITY VARCHAR(255) COMPANYNAME VARCHAR(255) STREET VARCHAR(255) ZIPCODE VARCHAR(255) SQL210104211946280 (COMPANYID) iu SQL210104211946280 (COMPANYID) UNIQUE
▼	CUSTOMER
	COMPANYID INTEGER CITY VARCHAR(255) COMPANYNAME VARCHAR(255) STREET VARCHAR(255) ZIPCODE VARCHAR(255) DISCOUNT INTEGER SQL210104211946300 (COMPANYID) iu SQL210104211946300 (COMPANYID) UNIQUE
▼	INVOICE
	INVOICEID INTEGER INVOICENUMBER INTEGER QUANTITY INTEGER SQL210104211946310 (INVOICEID) iu SQL210104211946310 (INVOICEID) UNIQUE
▼	PRODUCT
	PRODUCTID INTEGER PRODUCTNAME VARCHAR(255) UNITSINSTOCK INTEGER CATEGORY_FK INTEGER SUPPLIER_FK INTEGER SQL210104211946320 (PRODUCTID) FKEURY2HXL2J8URLKMW36585TKR (SUPPLIER_FK) → SUPPLIER (COMPANYID) FKPURGJJ563MV2AV0MGGDXEF7 (CATEGORY_FK) → CATEGORY (CATEGORYID) iu SQL210104211946320 (PRODUCTID) UNIQUE i SQL210104211946390 (CATEGORY_FK) i SQL210104211946400 (SUPPLIER_FK)
▼	PRODUCT_INVOICE
	PRODUCTS_PRODUCTID INTEGER INVOICES_INVOICEID INTEGER SQL210104211946340 (PRODUCTS_PRODUCTID, INVOICES_INVOICEID) FK3OTSIUDMV7Y4P1360MNQ4V7R (INVOICES_INVOICEID) → INVOICE (INVOICEID) FKCMCOHEDFHSUOR6LT20A304QD (PRODUCTS_PRODUCTID) → PRODUCT (PRODUCTID) iu SQL210104211946340 (PRODUCTS_PRODUCTID, INVOICES_INVOICEID) UNIQUE i SQL210104211946420 (INVOICES_INVOICEID) i SQL210104211946430 (PRODUCTS_PRODUCTID)
▼	SUPPLIER
	COMPANYID INTEGER CITY VARCHAR(255) COMPANYNAME VARCHAR(255) STREET VARCHAR(255) ZIPCODE VARCHAR(255) BANKACCOUNTNUMBER VARCHAR(255) SQL210104211946350 (COMPANYID) iu SQL210104211946350 (COMPANYID) UNIQUE



Zawartość tabeli **COMPANY** - pusta

APP.COMPANY [jdbc:derby://127.0.0.1/BKordekJPA]						
<input type="button" value=" &lt;"/> <input type="button" value="&lt;"/> <input type="button" value="0 rows"/> <input type="button" value="&gt;"/> <input type="button" value=" &gt;"/> <input type="button" value="refresh"/> <input type="button" value="+"/> <input type="button" value="-"/> Tx: Auto <input type="button" value="DB"/> <input type="button" value="DDL"/> <input type="button" value="Commit"/> <input type="button" value="Cancel"/> Comma...d (CSV) <input type="button" value="Download"/>						
<b>Q&gt; &lt;Filter Criteria&gt;</b>						
COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT	

Zawartość tabeli **CUSTOMER**

APP.CUSTOMER [jdbc:derby://127.0.0.1/BKordekJPA]						
<input type="button" value=" &lt;"/> <input type="button" value="&lt;"/> <input type="button" value="2 rows"/> <input type="button" value="&gt;"/> <input type="button" value=" &gt;"/> <input type="button" value="refresh"/> <input type="button" value="+"/> <input type="button" value="-"/> Tx: Auto <input type="button" value="DB"/> <input type="button" value="DDL"/> Comma...d (CSV) <input type="button" value="Download"/> APP.CUSTOMER						
<b>Q&gt; &lt;Filter Criteria&gt;</b>						
COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT	
1	9	Kraków	Offices	Długa	31-146	10
2	10	Kraków	Company	Szeroka	30-146	5

## Zawartość tabeli *SUPPLIER*

COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	7	Kraków	Tesco	Kapelanka	30-226
2	8	Kraków	Żabka	Sołtysowska	31-589

Zostały utworzone osobne tabele dla klienta i dostawcy. Nawet przy wspólnych polach dla obu nie zapisują się one do tabeli COMPANY.

## Pobranie dostawców i klientów z bazy

Klasa Main - operacje odczytu

```
99
100     Supplier supplierId7 = em.find(Supplier.class, o: 7);
101     Supplier supplierId8 = em.find(Supplier.class, o: 8);
102     Customer customerId9 = em.find(Customer.class, o: 9);
103     Customer customerId10 = em.find(Customer.class, o: 10);
104
105     System.out.println(supplierId7.toString());
106     System.out.println(supplierId8.toString());
107     System.out.println(customerId9.toString());
108     System.out.println(customerId10.toString());
109 /*
```

## Wynik - metoda Single Table

```
values
next value for hibernate_sequence
Supplier{bankAccountNumber='340727370997063', products=[Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, category=11 Mrożonki}, Product{ProductID=3, ProductName='Pizza'
Supplier{bankAccountNumber='371989964645795', products=[Product{ProductID=4, ProductName='Chipsy', UnitsInStock=50, category=12 Przekąski}]
Customer{discount=10}
Customer{discount=5}
```

## Wynik - metoda Joined

```
values
next value for hibernate_sequence
Supplier{bankAccountNumber='340727370997063', products=[Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, category=11 Mrożonki}, Product{ProductID=6, ProductName='Pizza'
Supplier{bankAccountNumber='371989964645795', products=[Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, category=12 Przekąski}]
Customer{discount=10}
Customer{discount=5}
hibernate*
```

## Wynik - Metoda Table per Class

```
values
next value for hibernate_sequence
Supplier{bankAccountNumber='340727370997063', products=[Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, category=11 Mrożonki}, Product{ProductID=6, ProductName='Pizza'
Supplier{bankAccountNumber='371989964645795', products=[Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, category=12 Przekąski}]
Customer{discount=10}
Customer{discount=5}
```

Uzyskane wyniki odczytu z bazy są identyczne w przypadku każdej z metod.