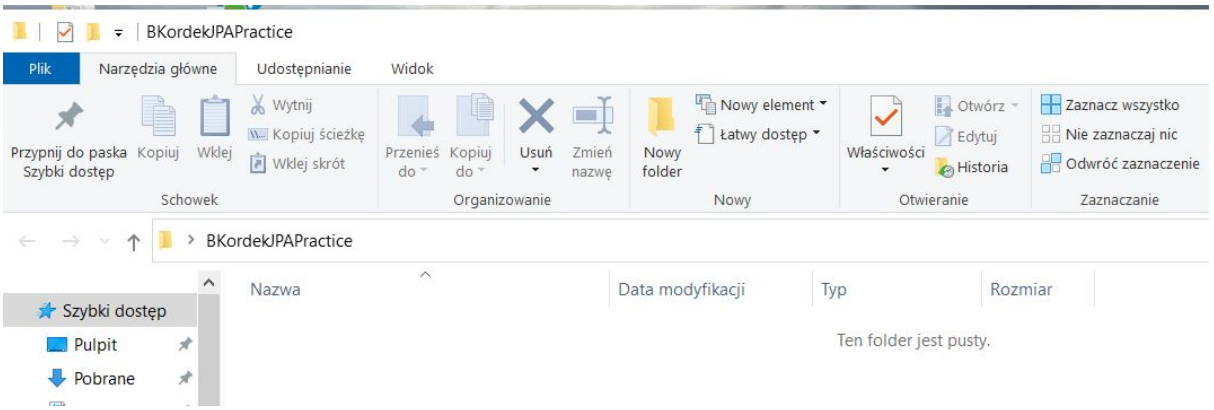
Bartosz Kordek

Bazy danych

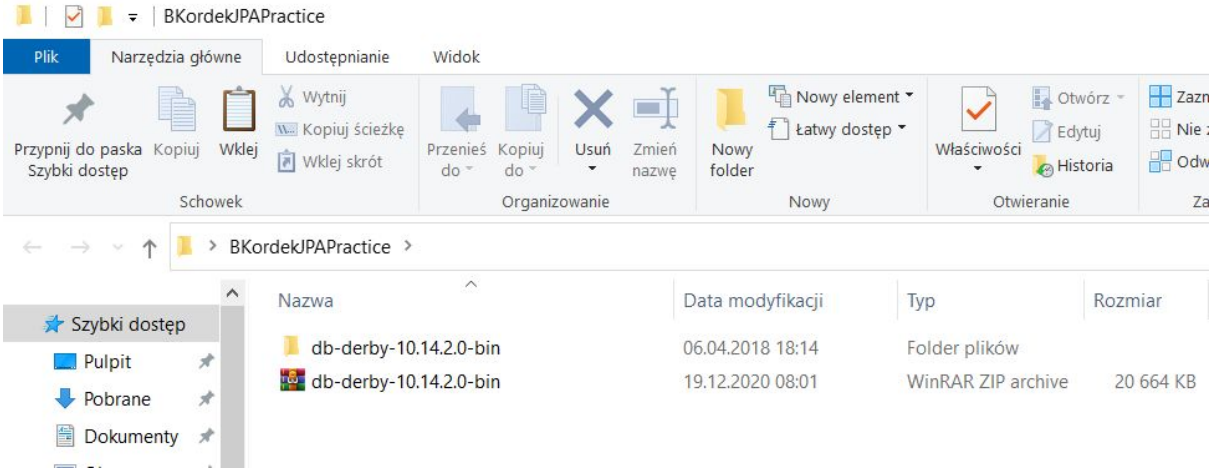
Laboratorium - Hibernate/JPA

I. Basics

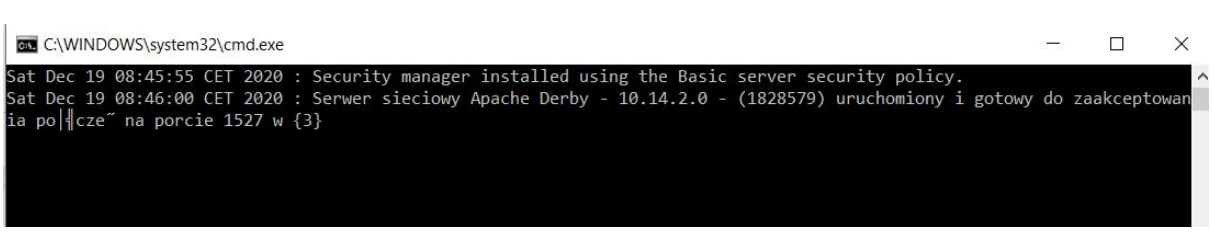
a. Stwórz folder INazwiskoJPAPractice



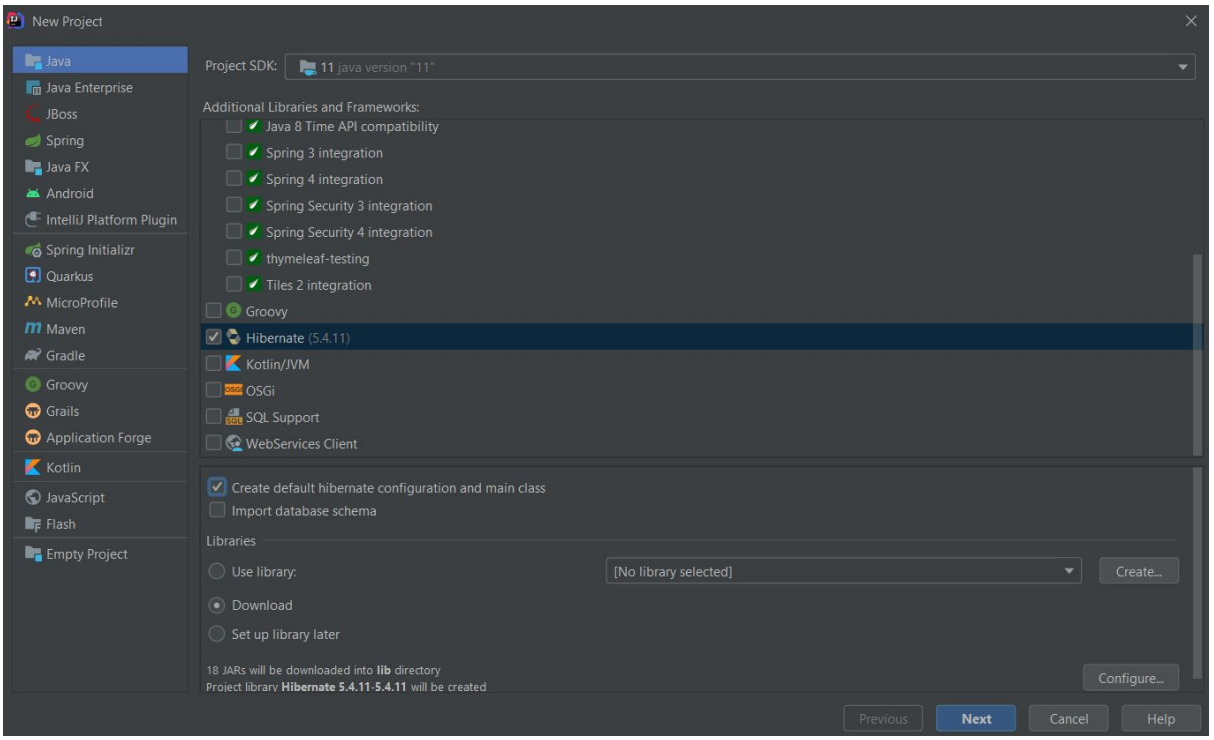
b. Sciagnij i rozpakuj w stworzonym folderze DBMS Derby https://wwweu.apache.org/dist//db/derby/db-derby-10.14.2.0/db-derby-10.14.2.0-bin.zip

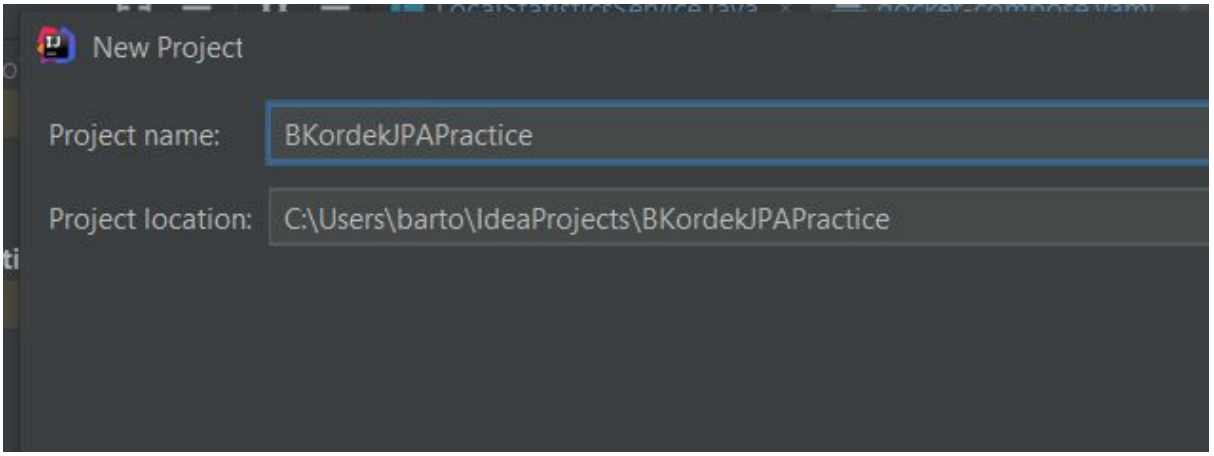


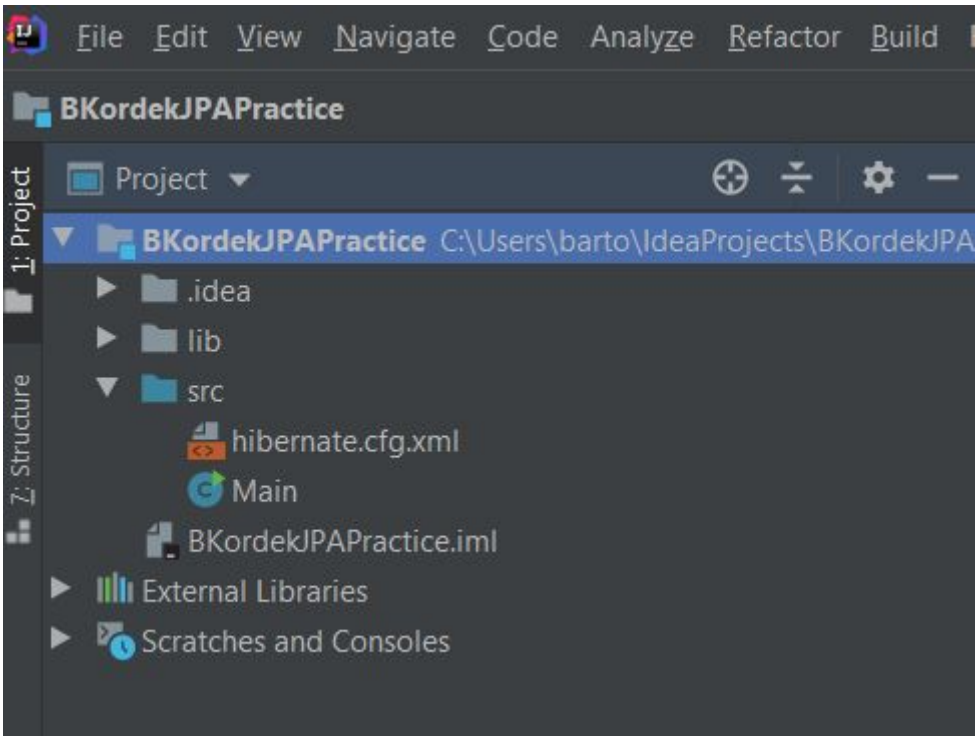
c. Uruchom serwer Derby (skrypt startNetworkSerwer z podkatalogu bin)



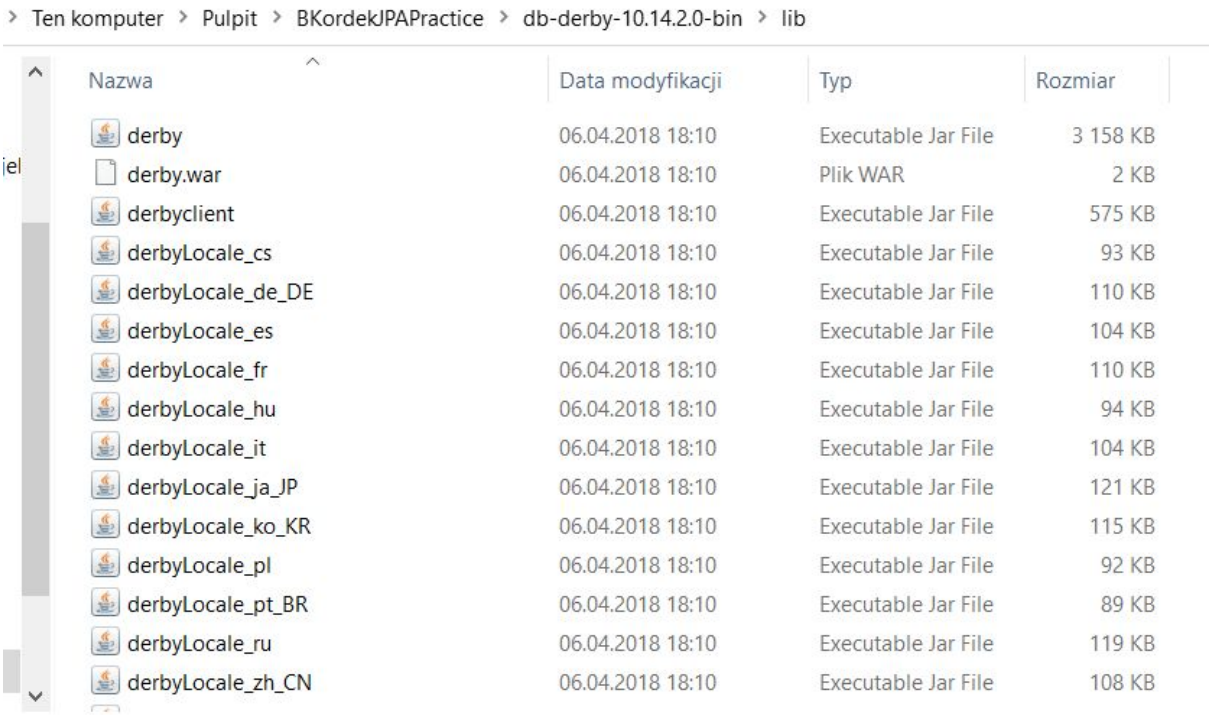
d. Stwórz projekt Javowy o nazwie InazwiskoJPAPractice

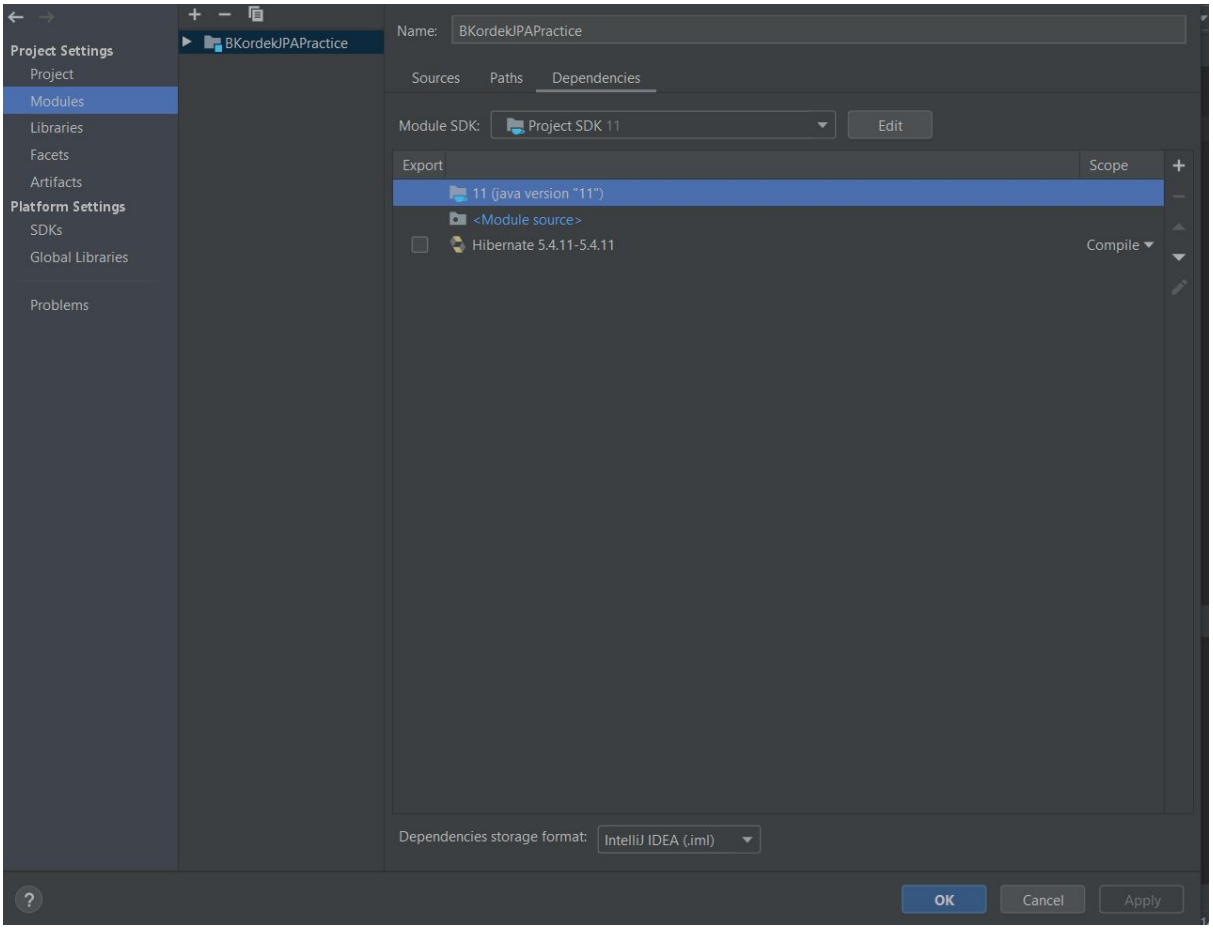


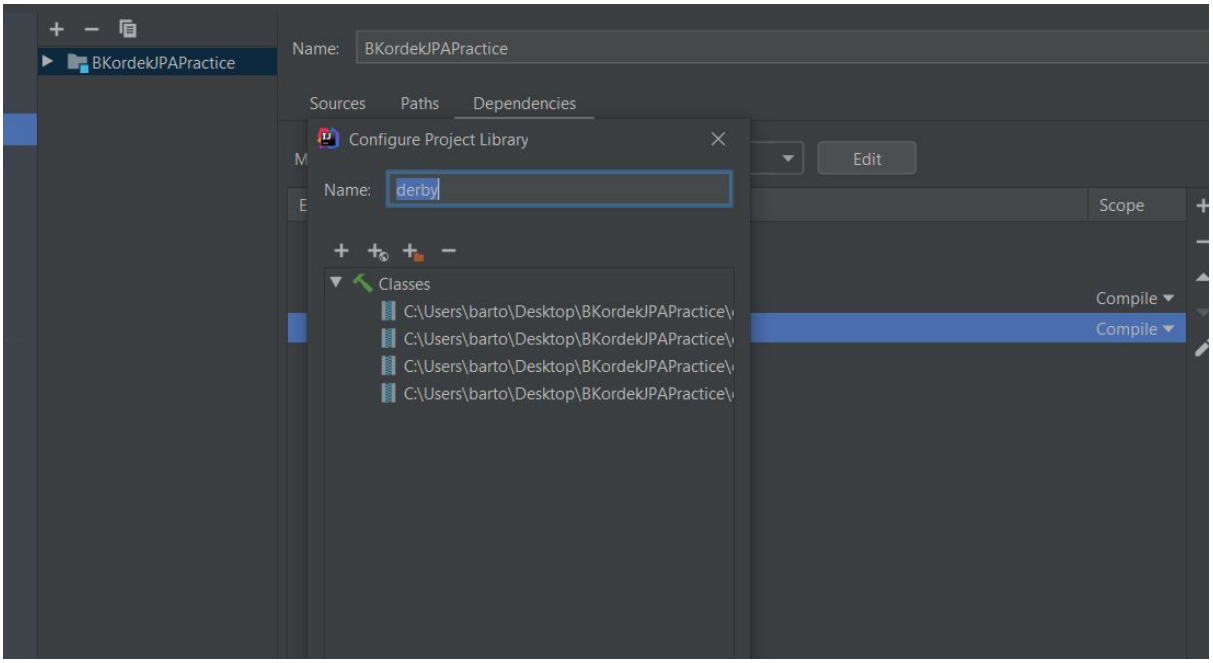


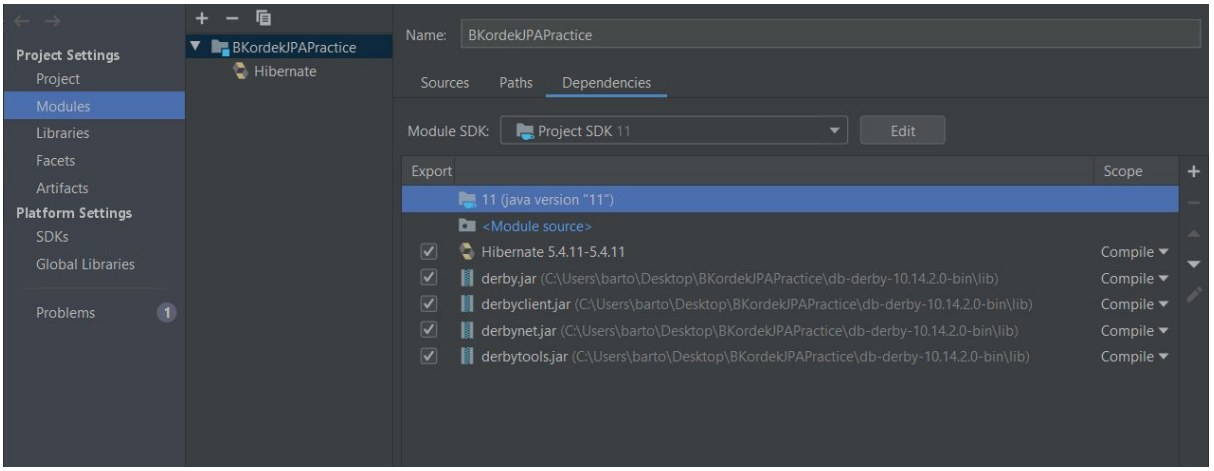


e. Dołącz do projektu (w IJ: File→Project Structure → Modules→ Dependencies) Jar-ki Związane z obsługą/komunikacją z Derby (*derby.jar*, *derbyclient.jar*, *derbynet.jar*, *derbytools.jar*).



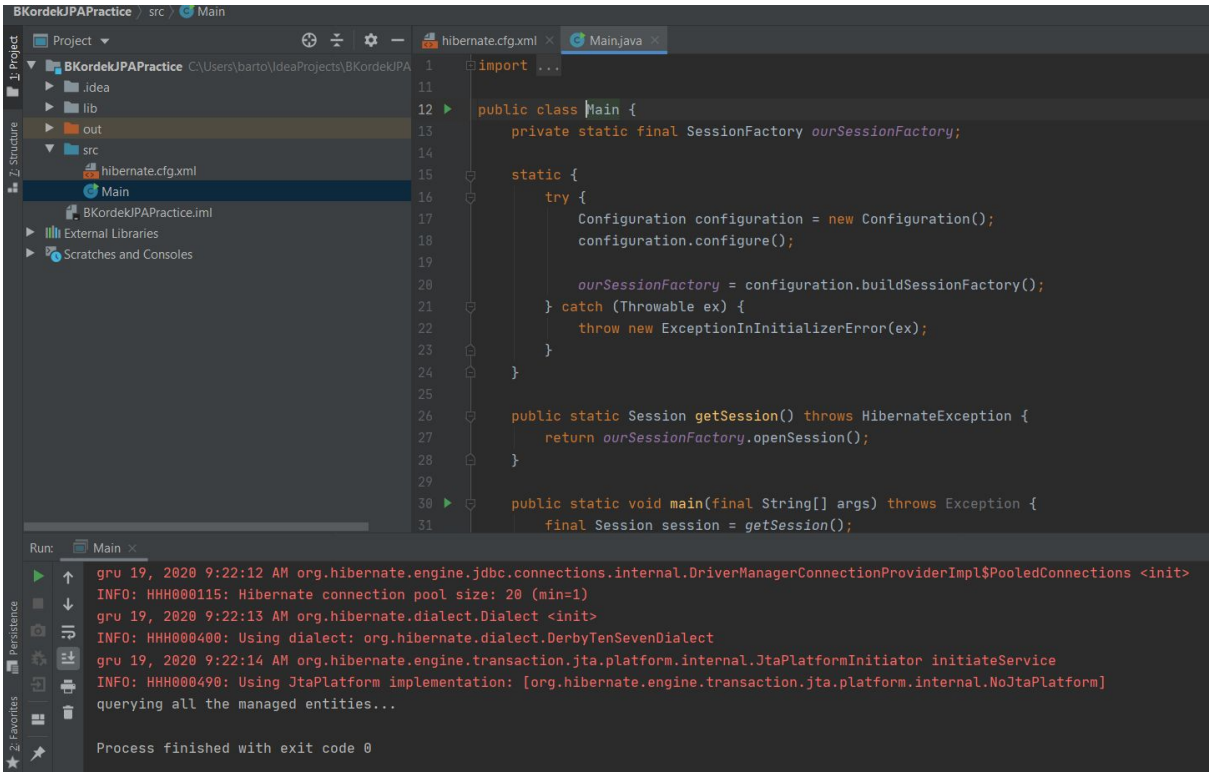

Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
BKordekJPAPractice
Name: BKordekJPAPractice
Sources
Paths
Dependencies
Module SDK: Project SDK 11
Edit
Export
Scope
11 (java version "11")
<Module source>
Hibernate 5.4.11-5.4.11
Compile
Dependencies storage format: IntelliJ IDEA (.iml)
OK
Cancel
Apply

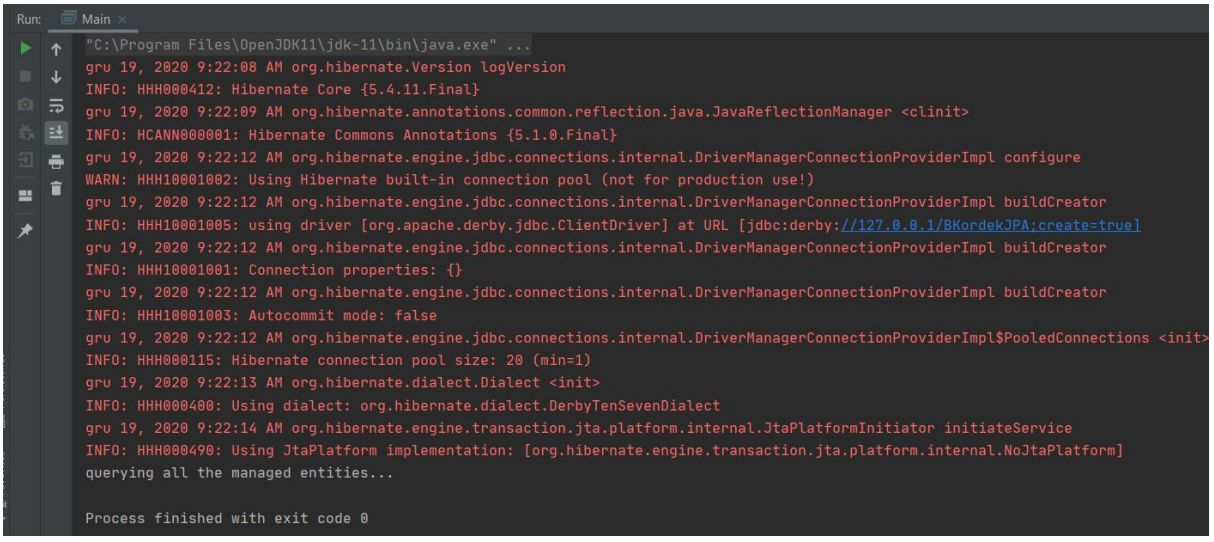

BKordekJPAPractice
Name: BKordekJPAPractice
Sources
Paths
Dependencies
Configure Project Library
Name: derby
Classes
C:\Users\barto\Desktop\BKordekJPAPractice\
C:\Users\barto\Desktop\BKordekJPAPractice\
C:\Users\barto\Desktop\BKordekJPAPractice\
C:\Users\barto\Desktop\BKordekJPAPractice\
Edit
Scope
Compile
Compile



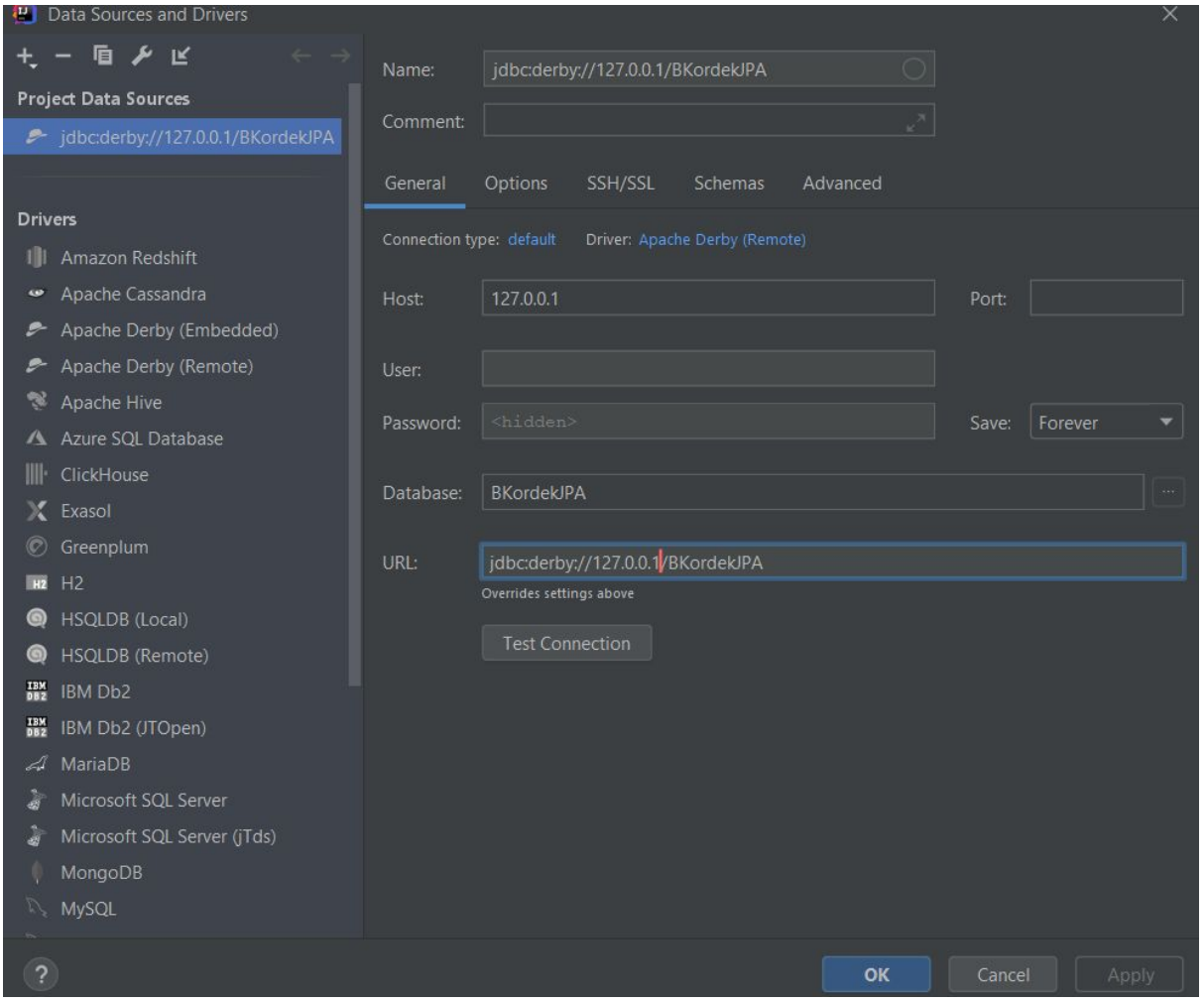
f. Uzupełnij wpisy w *hibernate.cfg.xml*

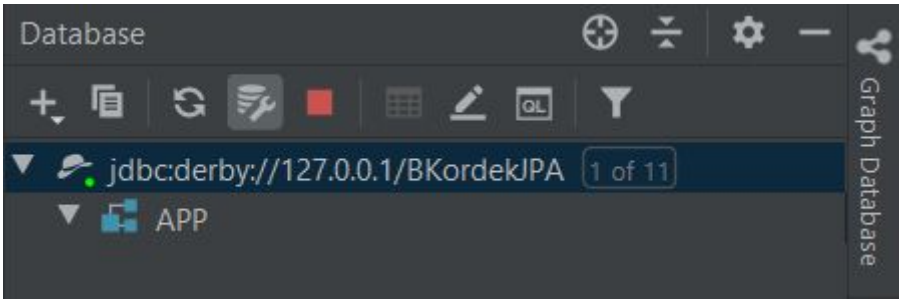
g. Uruchom projekt



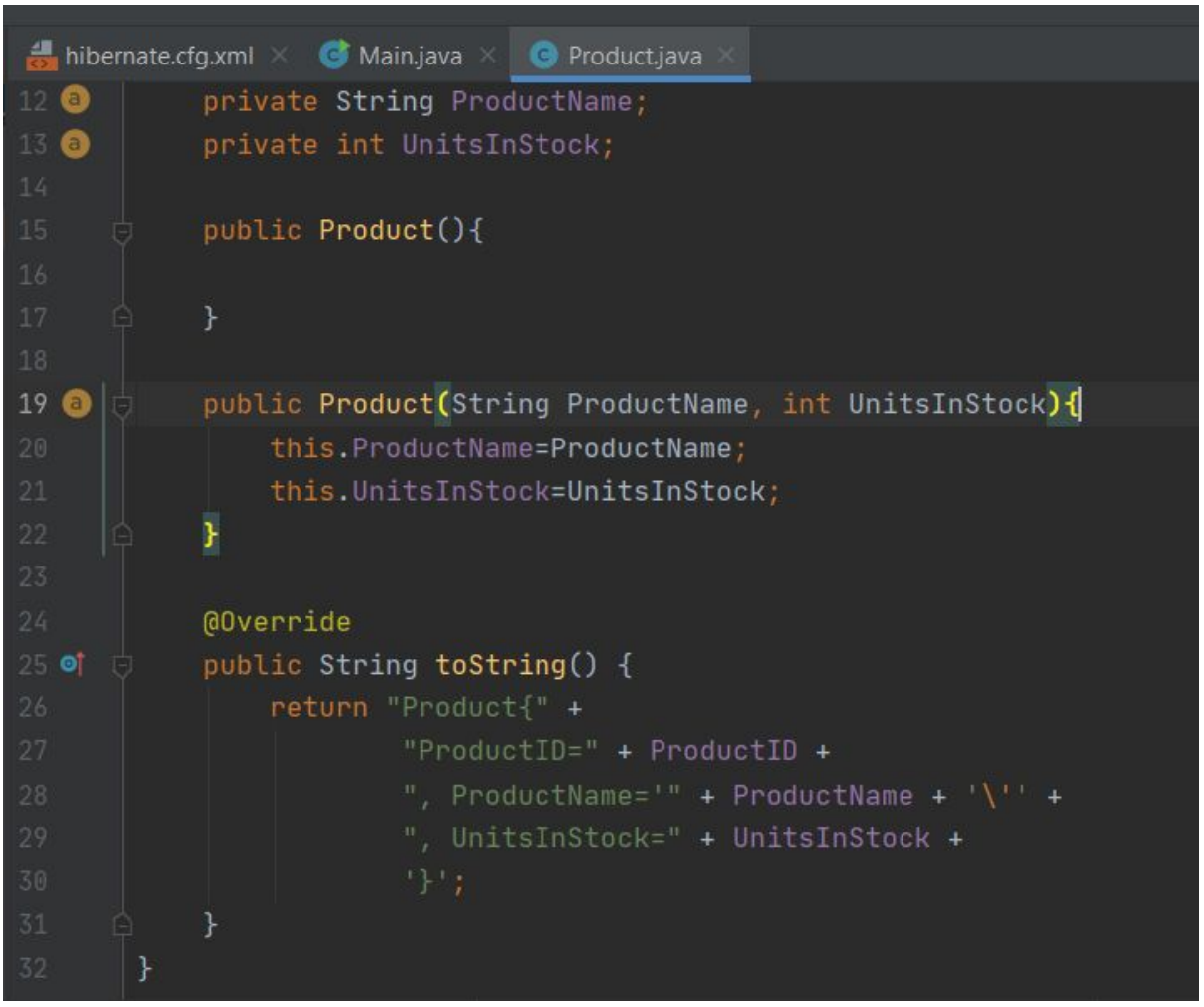


h. Schemat APP – na razie pusty

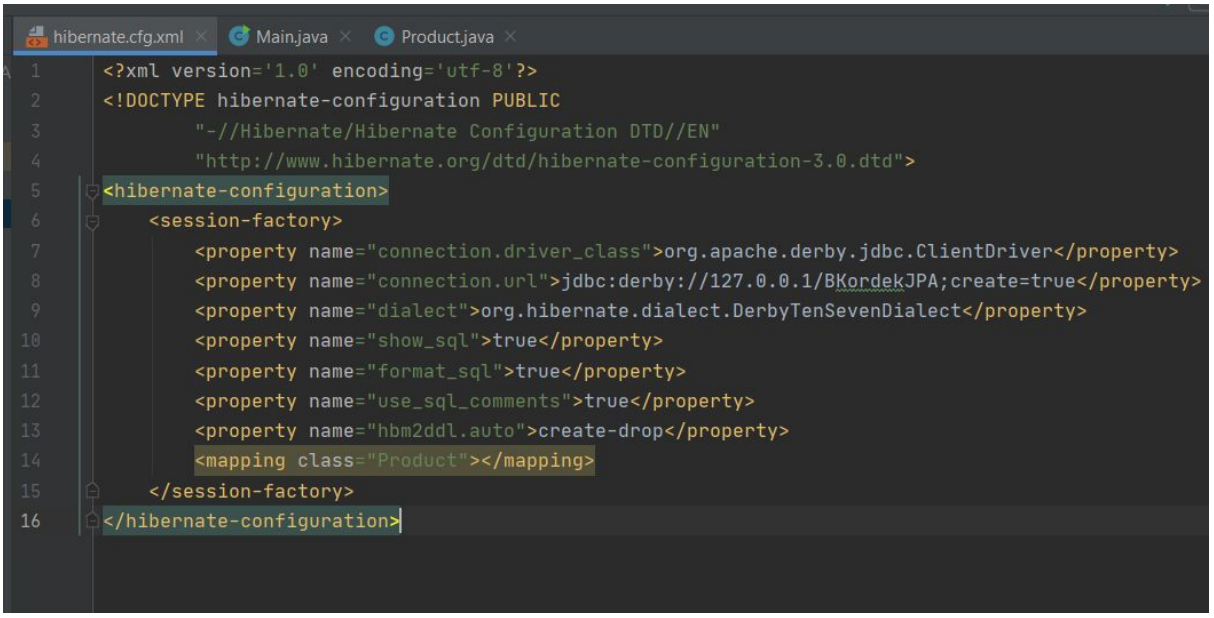




i. Stwórz klasę produktu z polami *ProductName*, *UnitsOnStock*. Zapewnij Mapowanie klasy do tabeli *Products*

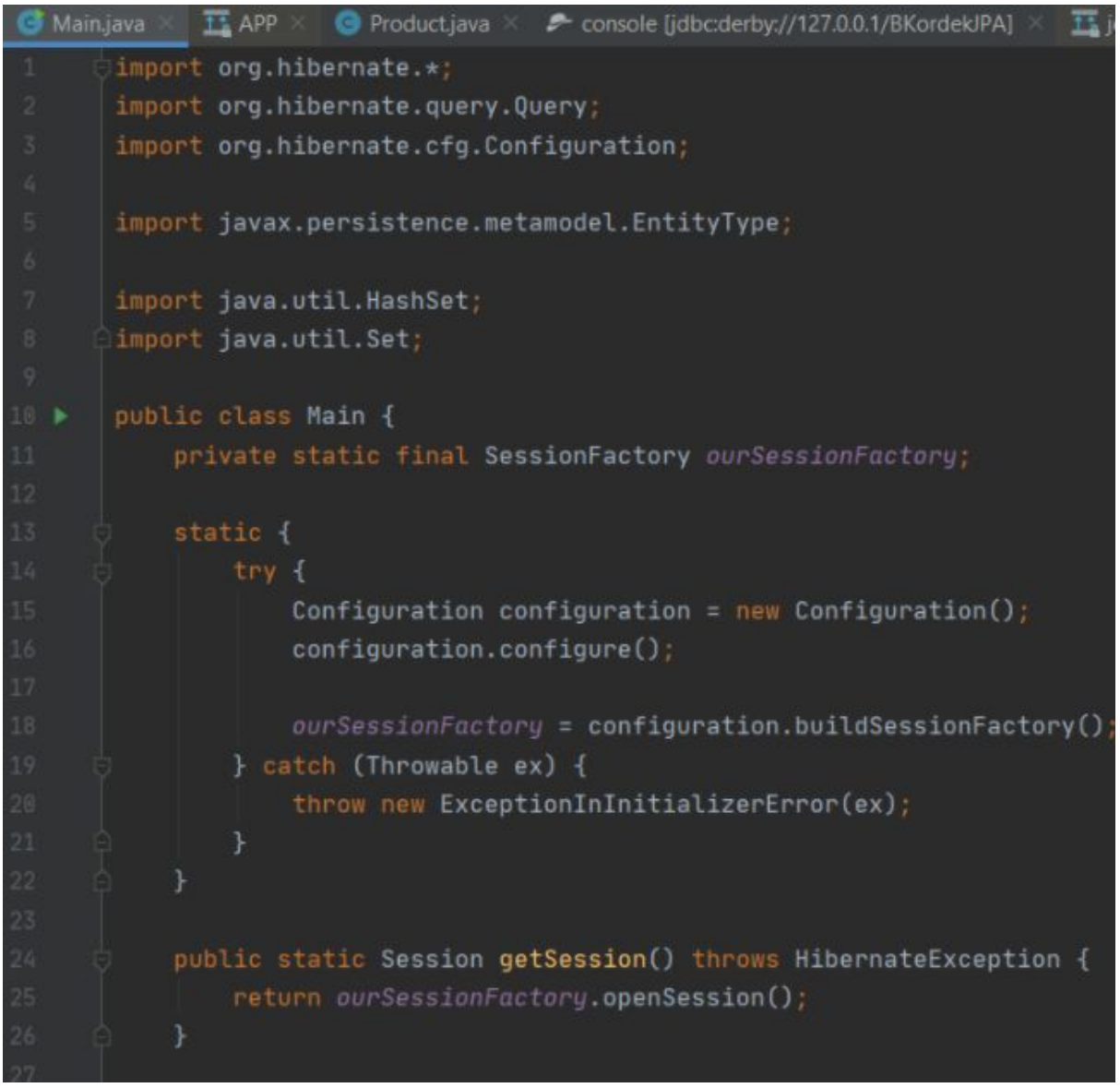


j. Uzupełnij w projekcie elementy potrzebne do zmapowania klasy do bazy danych

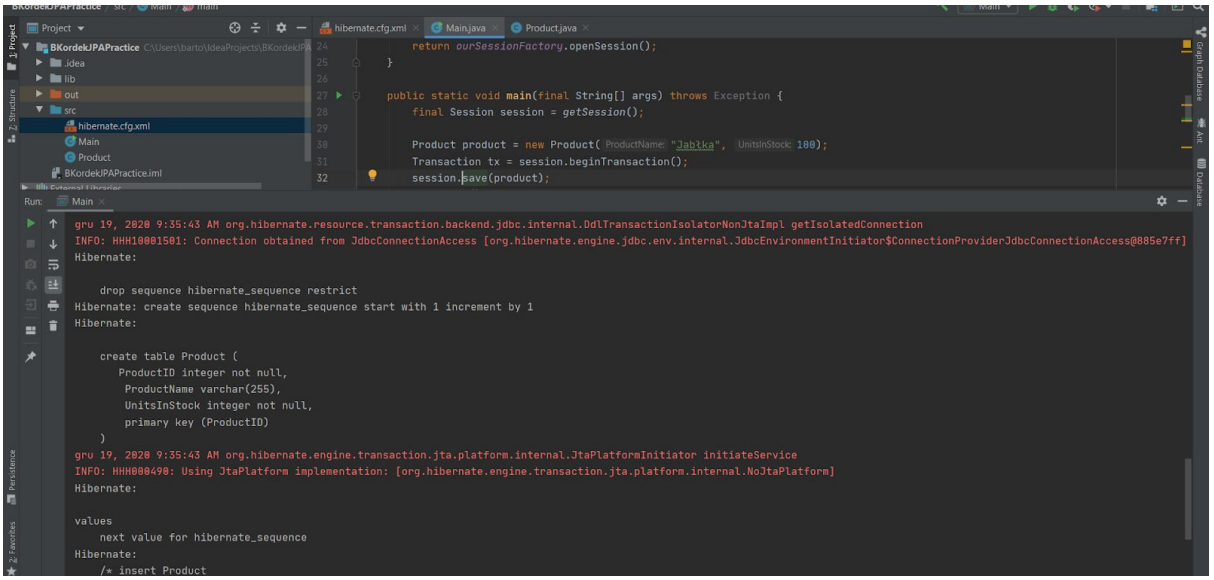


k. Jeżeli nie wygenerowałeś maina z tworzeniem sesji na etapie zakładania projektu w pliku głównym zapewnij tworzenie sesji i jej otwieranie.

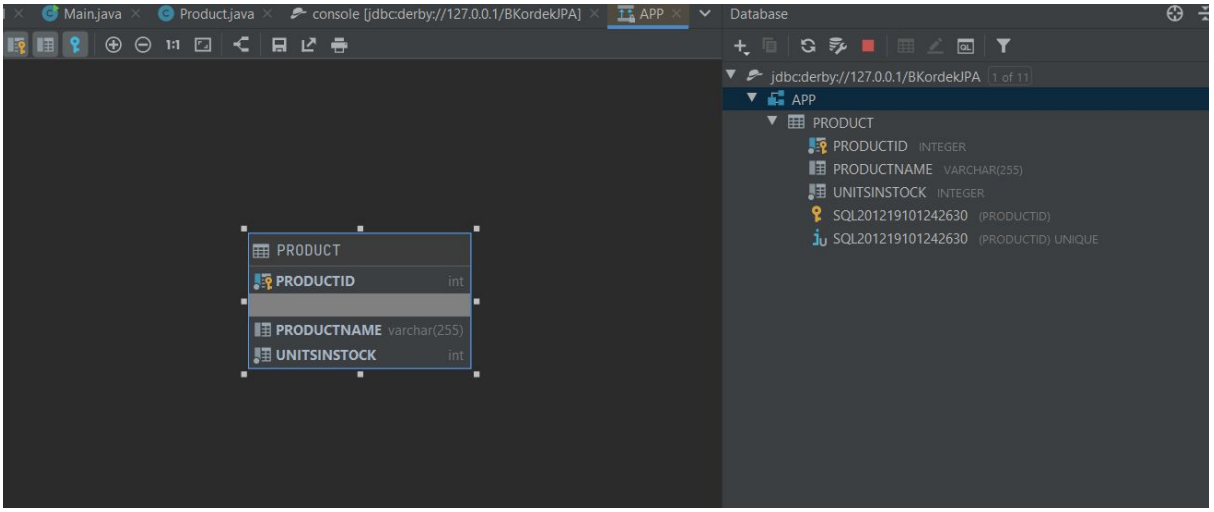
l.



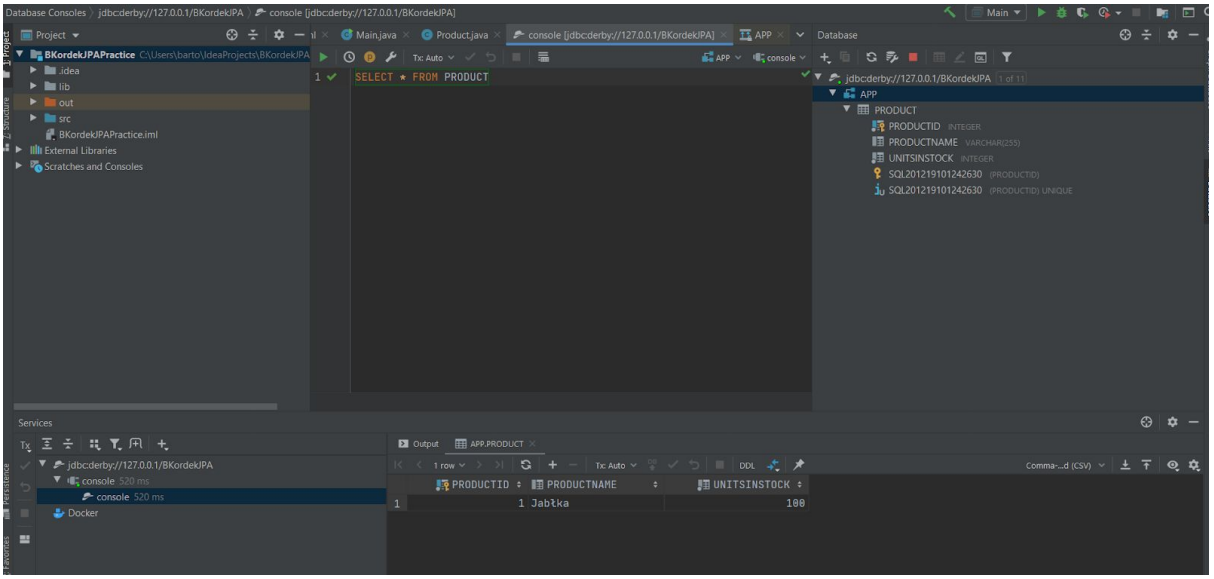
m. Uruchomienie projektu



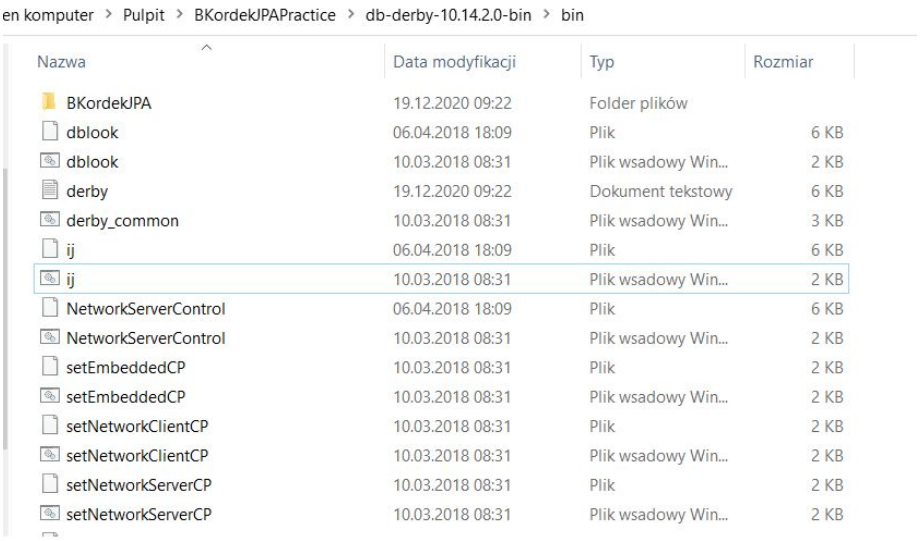
n. Widok schematu bazy danych z wykorzystaniem Intellij



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

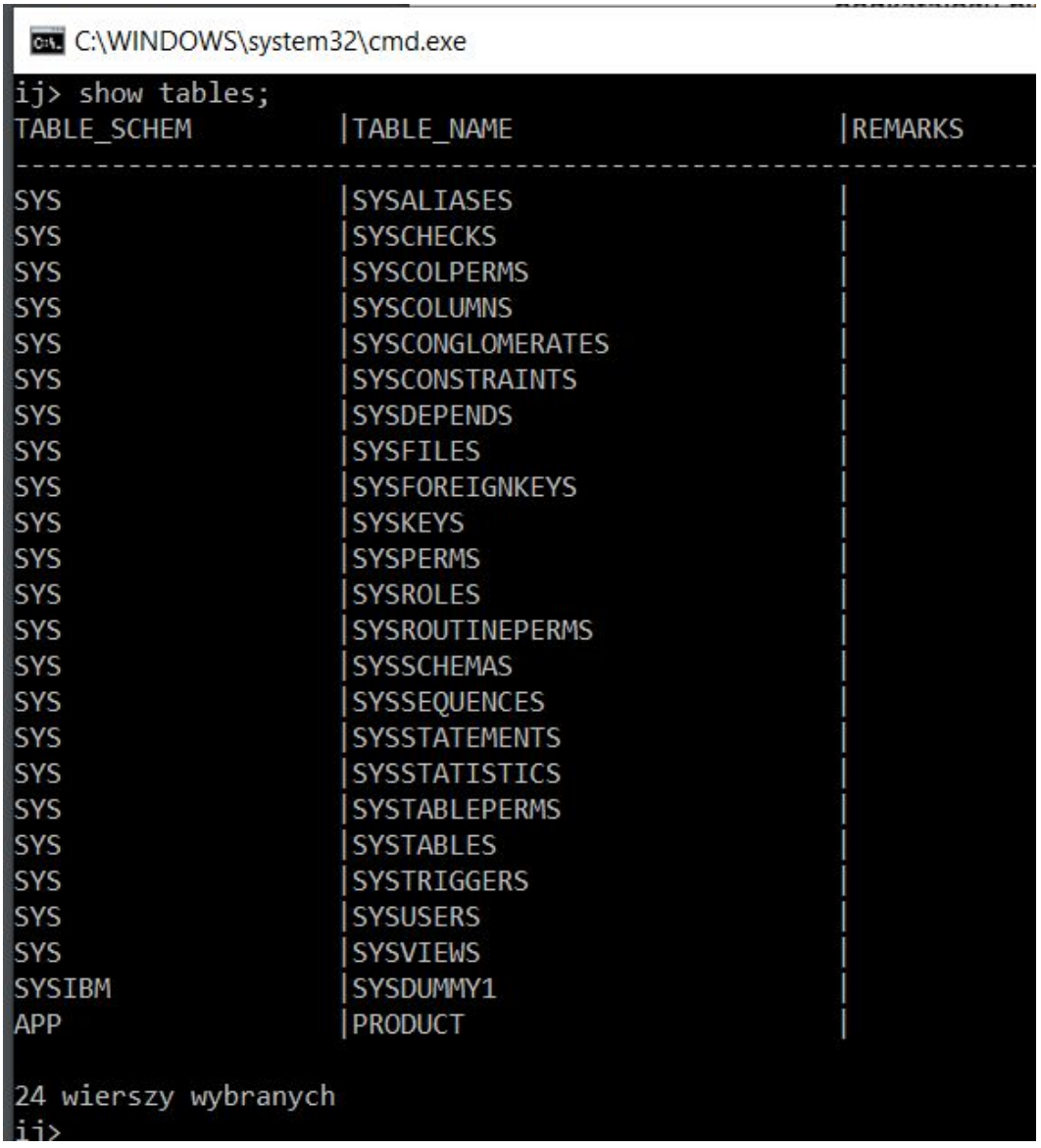


p. Widok tabel w bazie danych z wykorzystaniem shell’a serwera derby



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	
dblook	06.04.2018 18:09	Plik	6 KB
dblook	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              |SYSDUMMY1                       |
APP                 |PRODUCT                         |

24 wierszy wybranych
ij>
```

q. Dodanie produktu do bazy z wykorzystaniem klasy Scanner

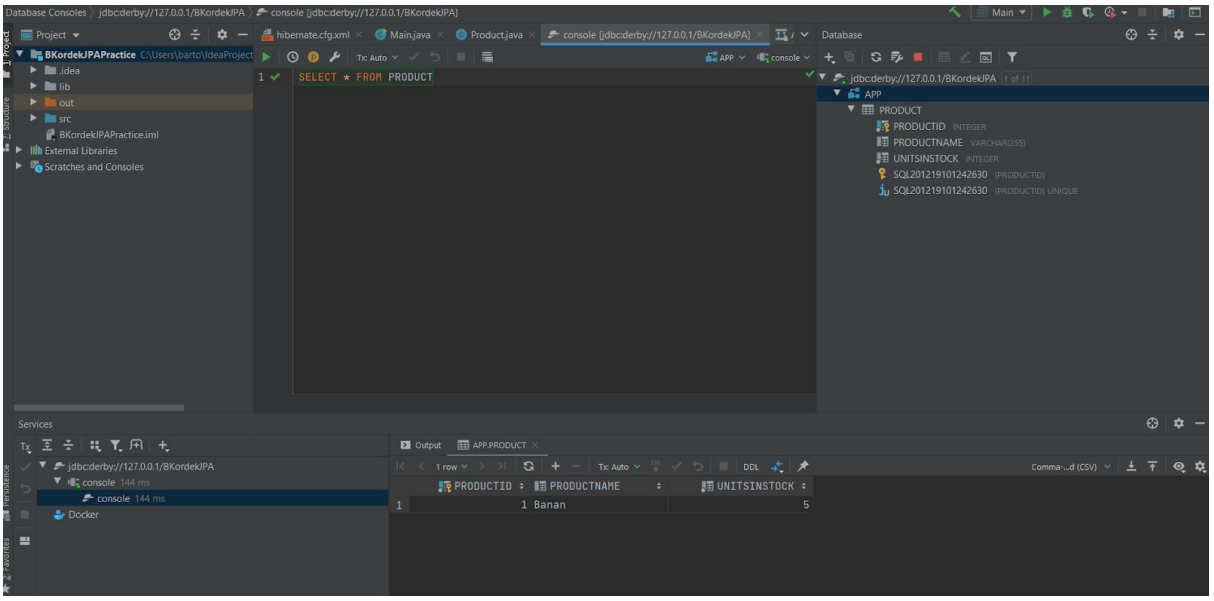
```
public static void main(final String[] args) throws Exception {

    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();
```

```
INFO: HHH000490: Using JtaPlatform implementation: [org.h
Podaj nazwę produktu:
Banan
Podaj ilość:
5
Hibernate:
```

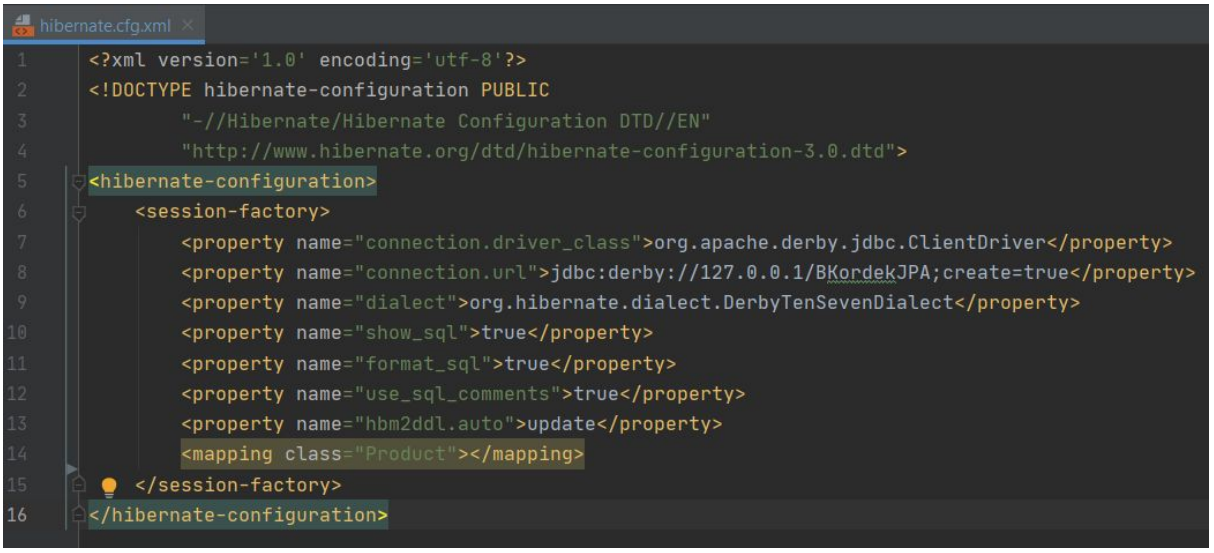
```
Run:  Main
            (ProductName, UnitsInStock, ProductID)
        values
            (?, ?, ?)
querying all the managed entities...
executing: from Product
Hibernate:
    /*
from
    Product */ select
        product0_.ProductID as producti1_0_,
        product0_.ProductName as productn2_0_,
        product0_.UnitsInStock as unitsins3_0_
    from
        Product product0_
  Product{ProductID=1, ProductName='Banan', UnitsInStock=5}

Process finished with exit code 0

6: TODO  4: Run  Graph Database Console  Terminal  8: Services  Java Enterprise  0:
```

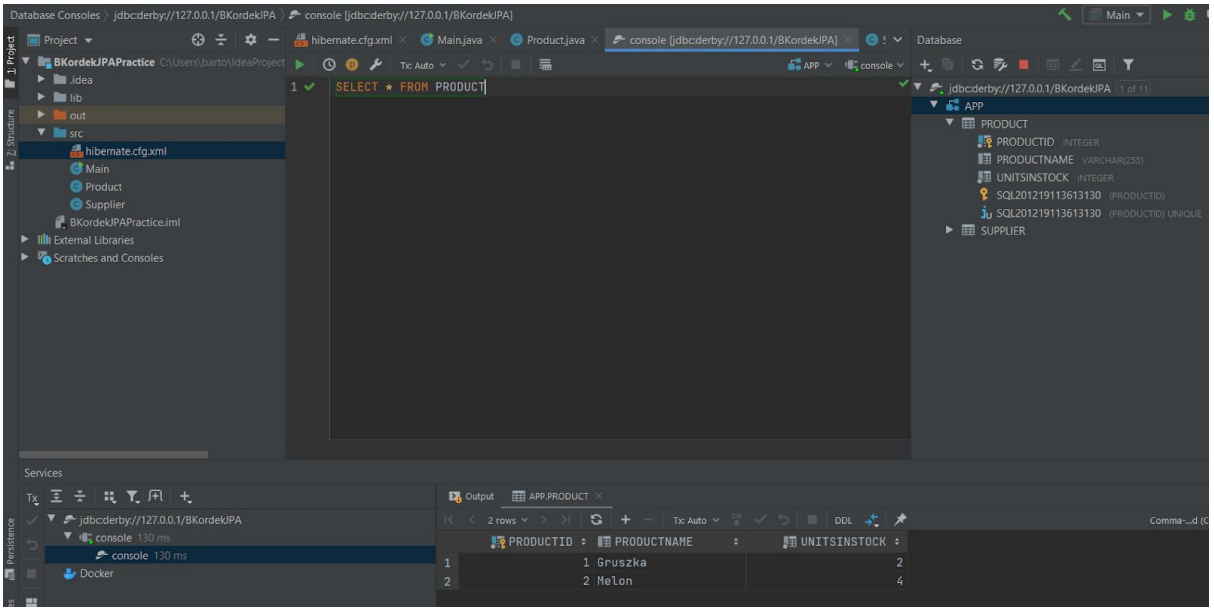
r. Sprawdzenie stanu magazynowego

- W przypadku właściwości create-drop w pliku konfiguracyjnym



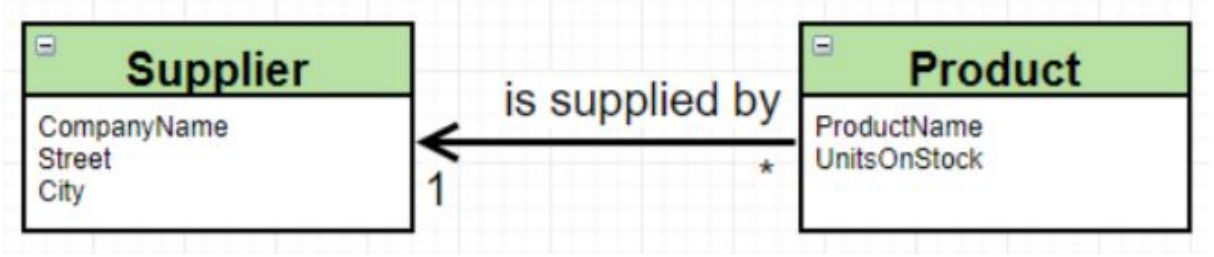
- W przypadku właściwości update w pliku konfiguracyjnym





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

IV. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej:

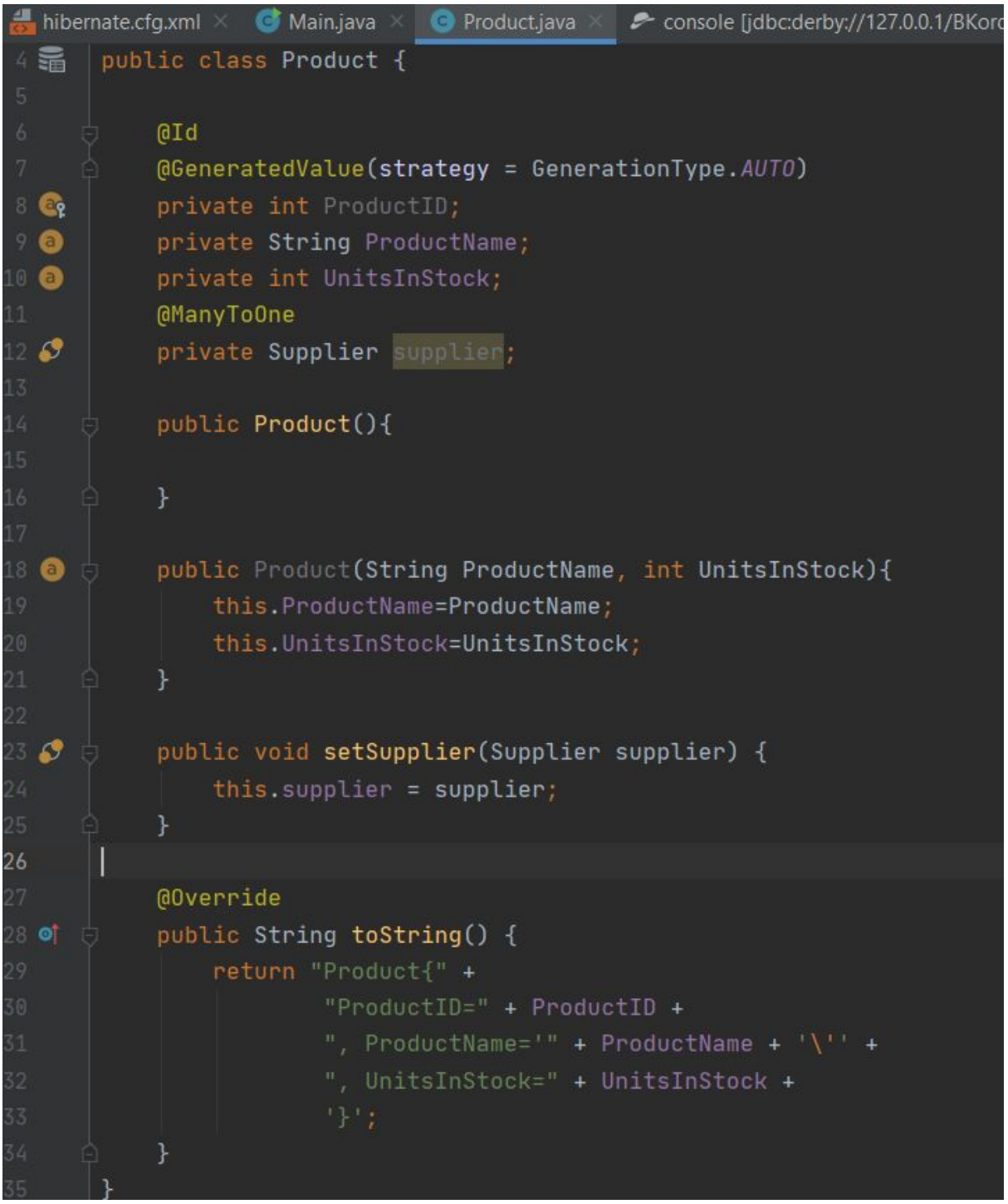


Dodanie mapowania klasy *Supplier* w pliku konfiguracyjnym *hibernate.cfg.xml*

```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*

Klasa *Product*



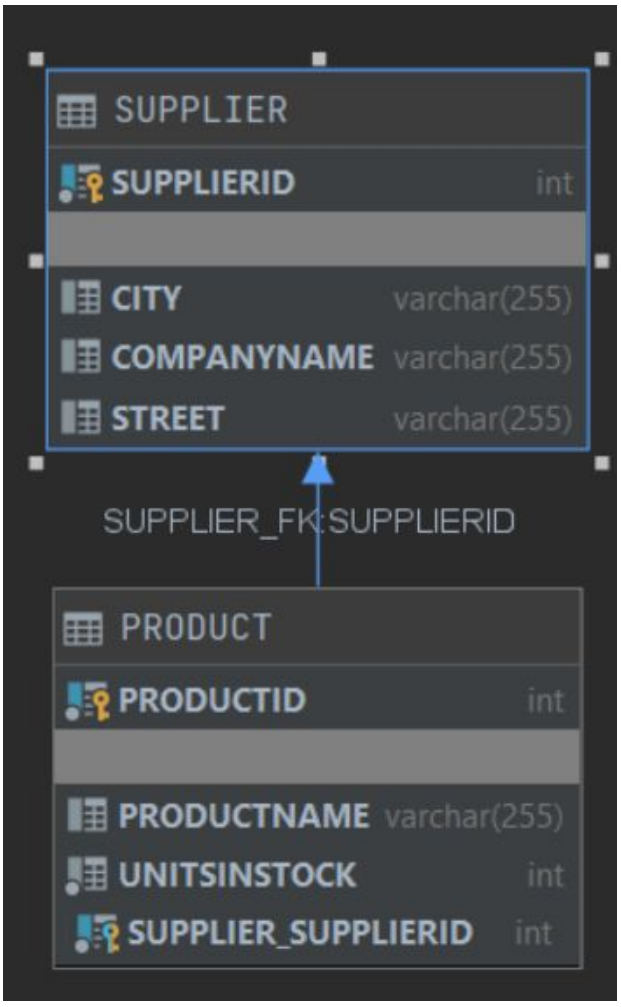
Klasa sterująca *Main* - stworzenie i dodanie dostawcy do bazy danych

```
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();
```

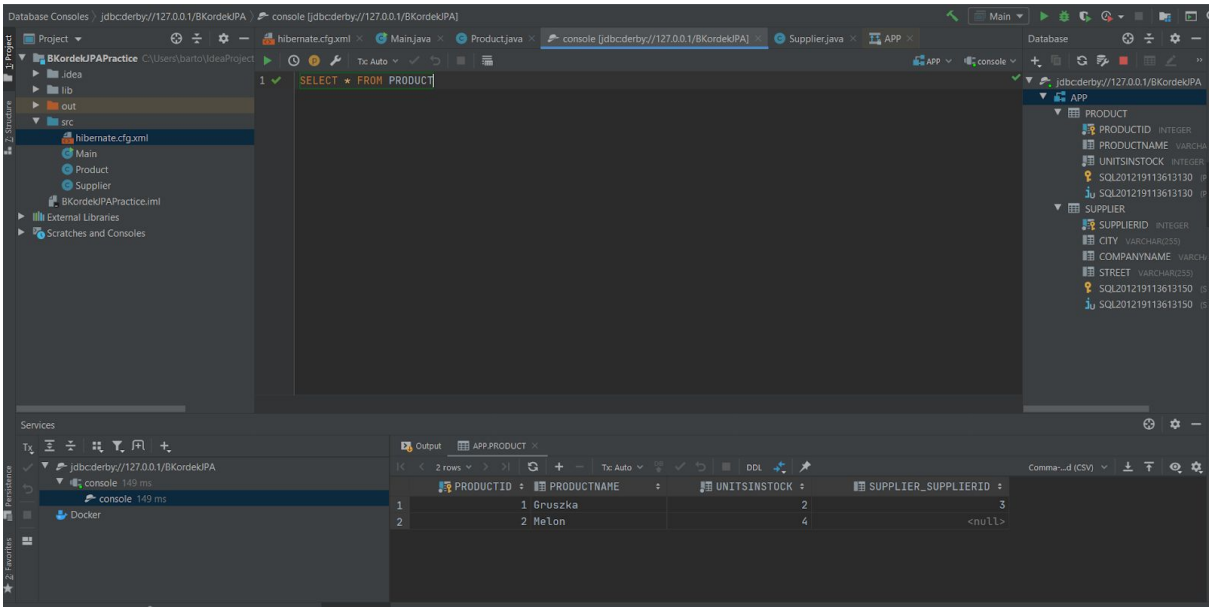
Wynik działania programu

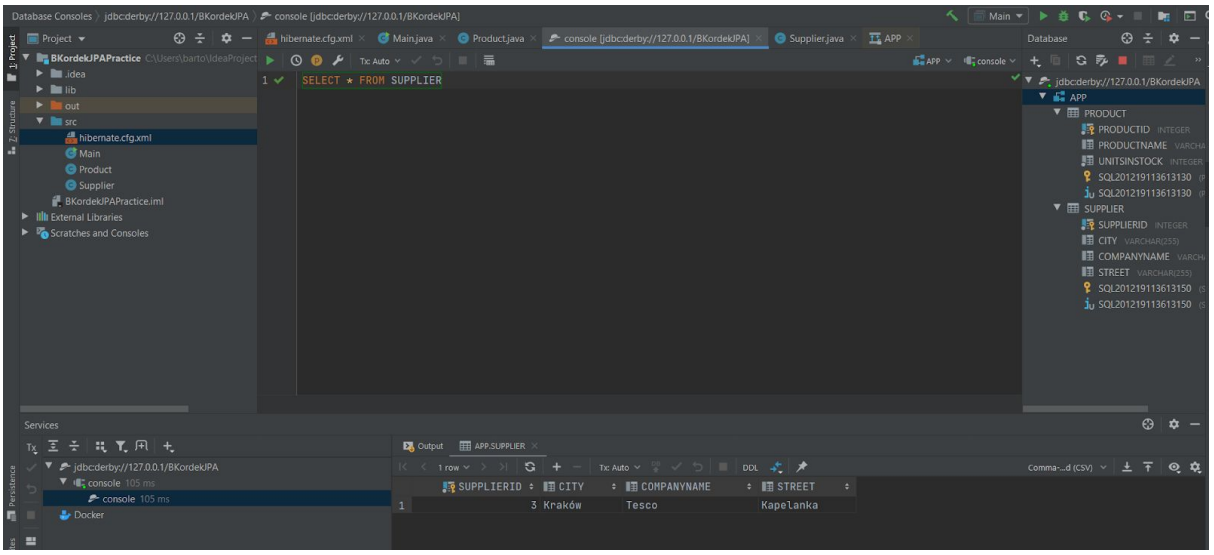
```
Run:  Main
    Supplier */ select
        supplier0_.SupplierID as supplier1_1_,
        supplier0_.City as city2_1_,
        supplier0_.CompanyName as companyn3_1_,
        supplier0_.Street as street4_1_
    from
        Supplier supplier0_
  Supplier{SupplierID=3, CompanyName='Tesco', Street='Kapelanka', City='Krakó
executing: from Product
Hibernate:
    /*
from
    Product */ select
        product0_.ProductID as producti1_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



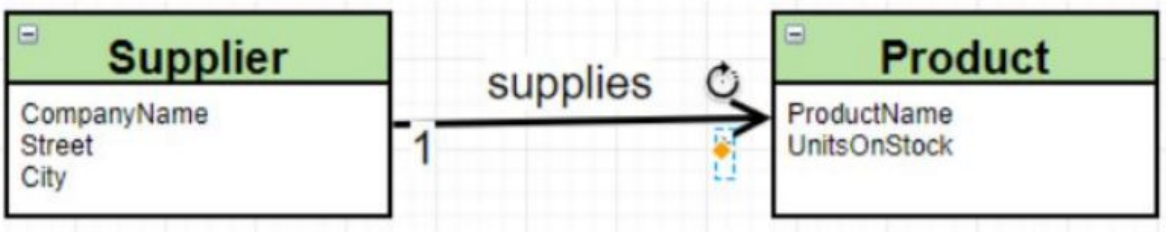
Zawartość bazy danych




Database Consoles
jdbc:derby://127.0.0.1/BKordekJPA
console [jdbc:derby://127.0.0.1/BKordekJPA]
Main
Project
BKordekJPAPractice C:\Users\barto\IdeaProject
.idea
lib
out
src
hibernate.cfg.xml
Main
Product
Supplier
BKordekJPAPractice.iml
External Libraries
Scratches and Consoles
hibernate.cfg.xml
Main.java
Product.java
console [jdbc:derby://127.0.0.1/BKordekJPA]
Supplier.java
APP
Tx: Auto
1 SELECT * FROM SUPPLIER
Database
jdbc:derby://127.0.0.1/BKordekJPA
APP
PRODUCT
PRODUCTID INTEGER
PRODUCTNAME VARCHA
UNITSINSTOCK INTEGER
SQL201219113613130
SQL201219113613130
SUPPLIER
SUPPLIERID INTEGER
CITY VARCHAR(255)
COMPANYNAME VARCH
STREET VARCHAR(255)
SQL201219113613150
SQL201219113613150
Services
jdbc:derby://127.0.0.1/BKordekJPA
console 105 ms
console 105 ms
Docker
Output
APP.SUPPLIER
1 row
Tx: Auto
DDL
Comma-...d (CSV)
SUPPLIERID CITY COMPANYNAME STREET
1 3 Kraków Tesco Kapelanka

V. Odwróć relacje zgodnie z poniższym schematem

ODWRÓCONA RELACJA



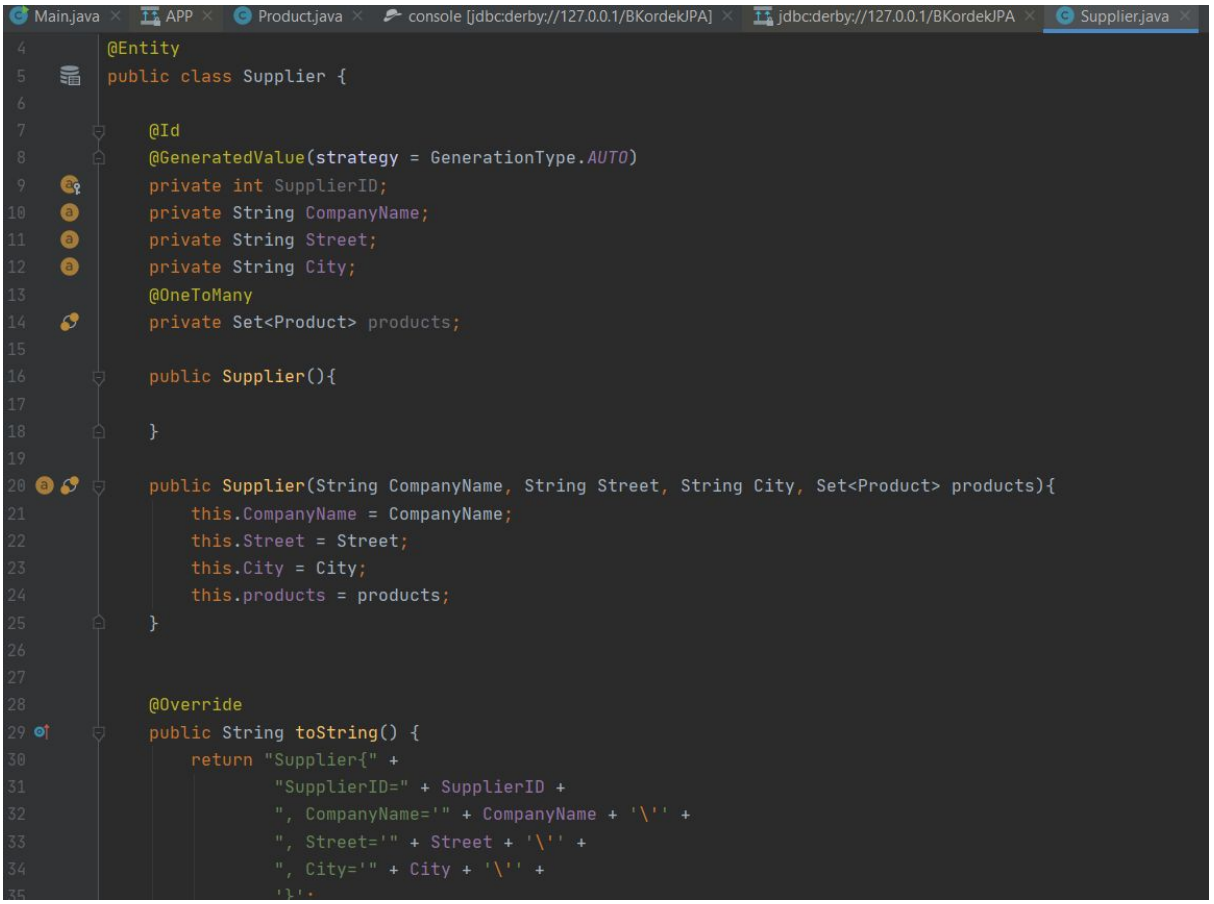
a) Wariant z tabelą łącznikową

Klasa *Product*

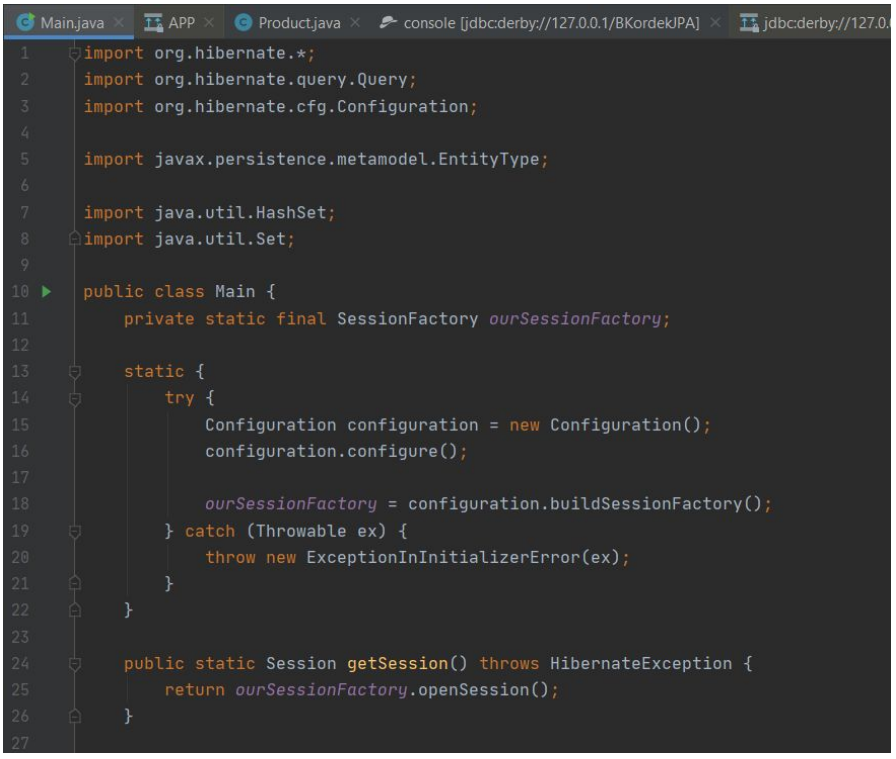
hibernate.cfg.xml | Main.java | APP | Product.java | console [jdbc:d

```java
import javax.persistence.*;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsInStock;

    public Product(){

    }

    public Product(String ProductName, int UnitsInStock){
        this.ProductName=ProductName;
        this.UnitsInStock=UnitsInStock;
    }


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

Klasa *Supplier*



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

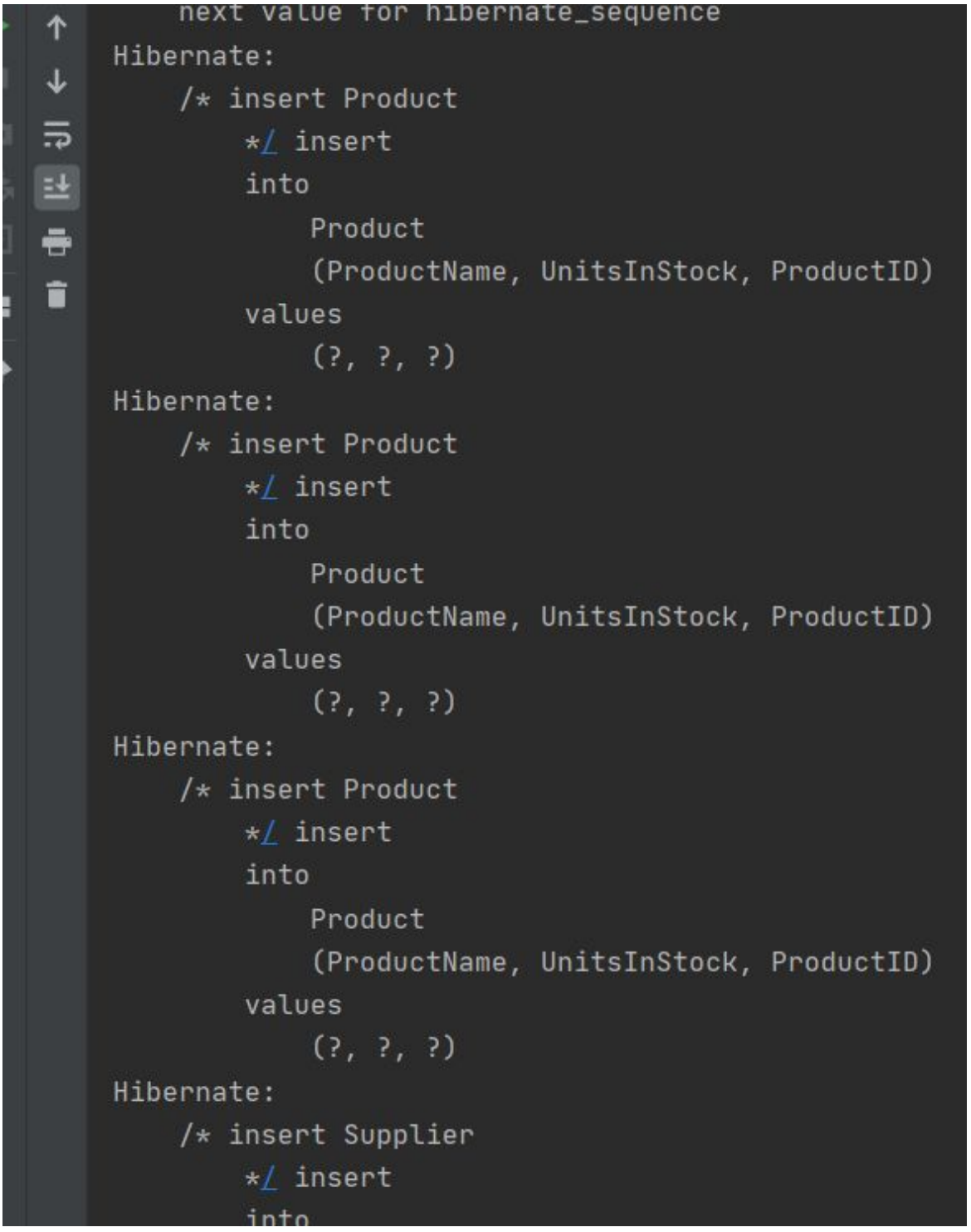


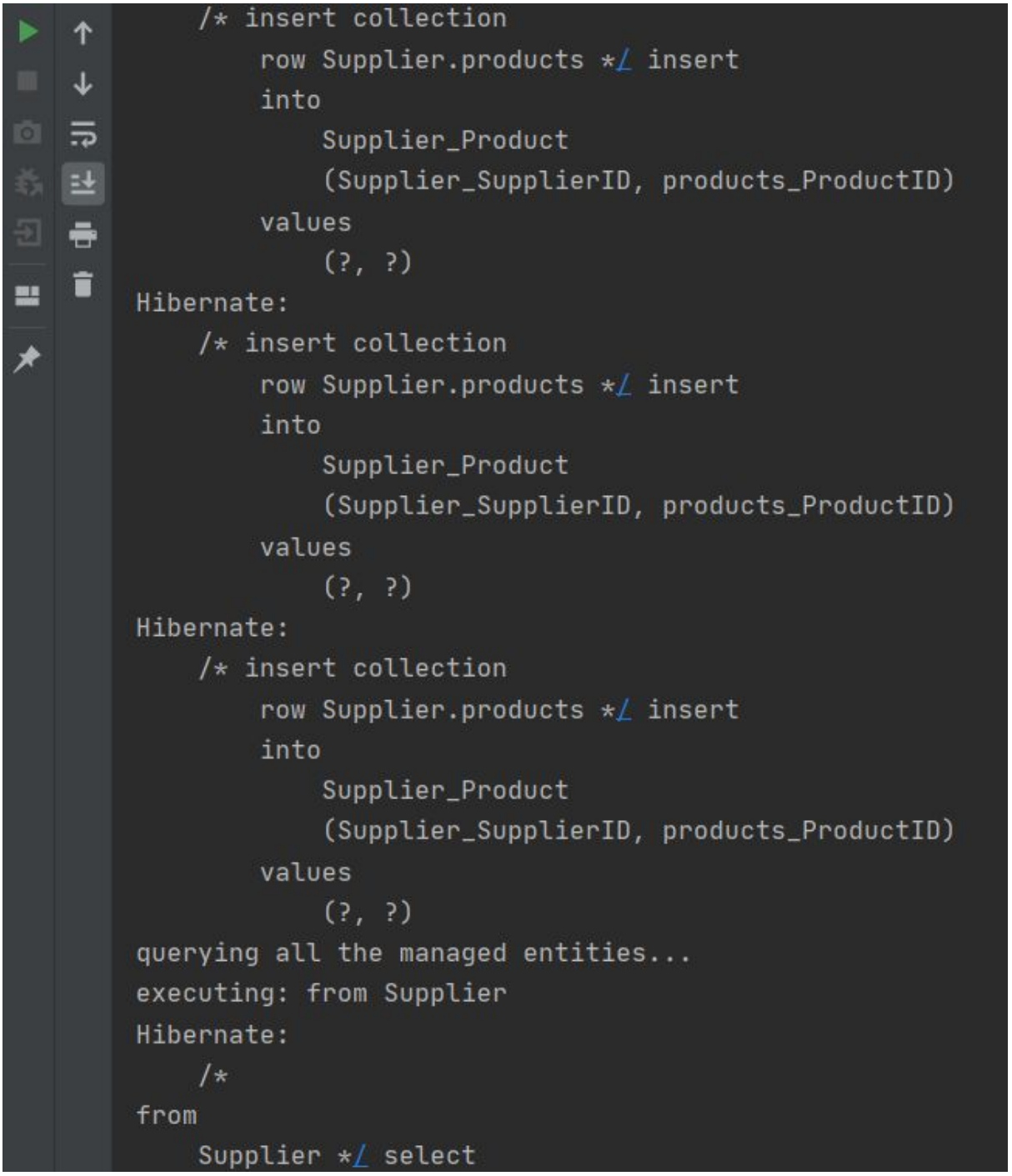
Main.java | APP | Product.java | console [jdbc:derby://127.0.0.1/BKordekJPA] | jdbc:derby://127.0.0.1/BKordekJPA | Supplier.java

```
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", productSetFo
42        Supplier supplier2 = new Supplier( CompanyName: "Żabka",  Street: "Sołtysowka",  City: "Kraków", productSetF
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
54        tx.commit();
55        //session.close();
56
57        try {
58            System.out.println("querying all the managed entities...");
```

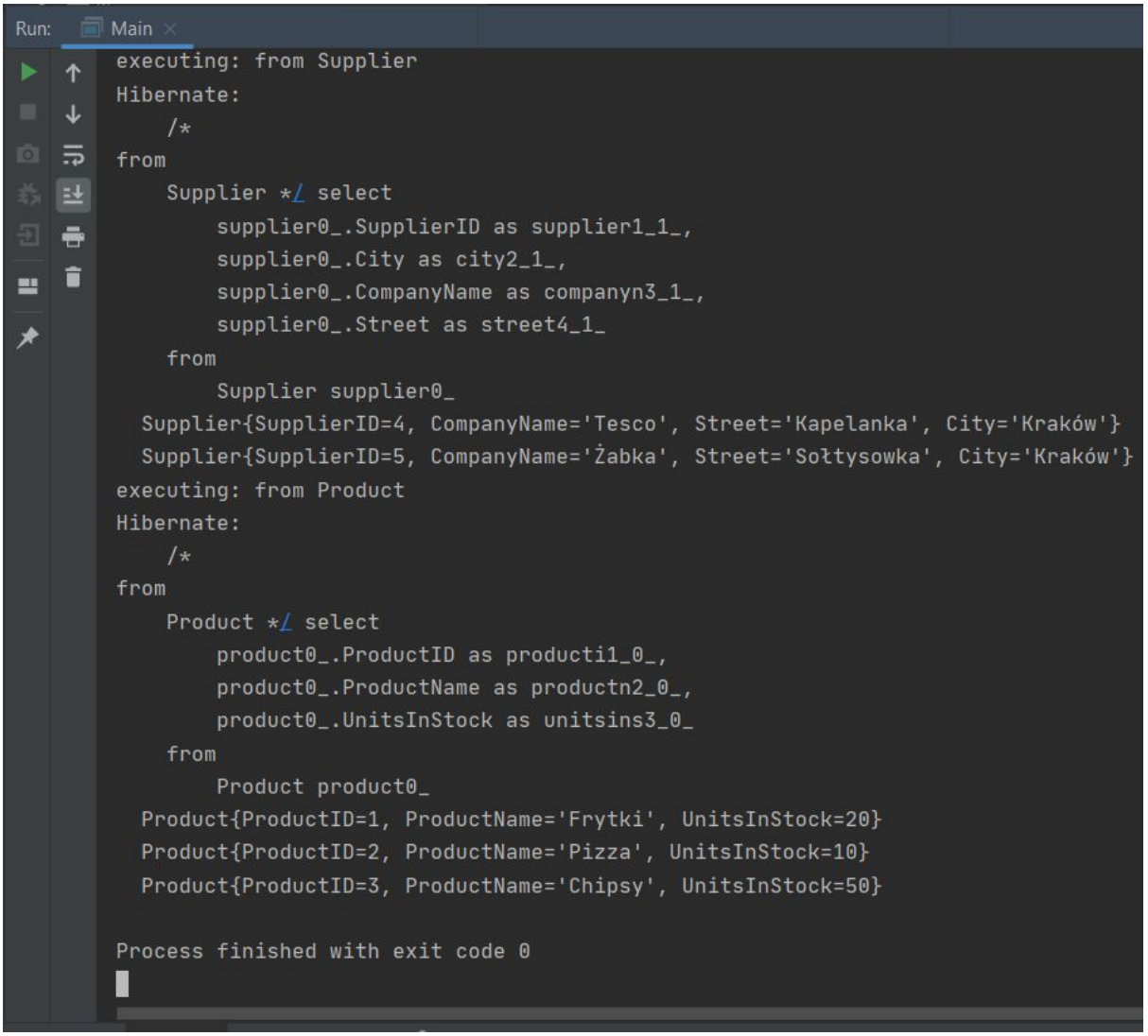
```
56
57        try {
58            System.out.println("querying all the managed entities...");
59            final Metamodel metamodel = session.getSessionFactory().getMetamodel();
60            for (EntityType<?> entityType : metamodel.getEntities()) {
61                final String entityName = entityType.getName();
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

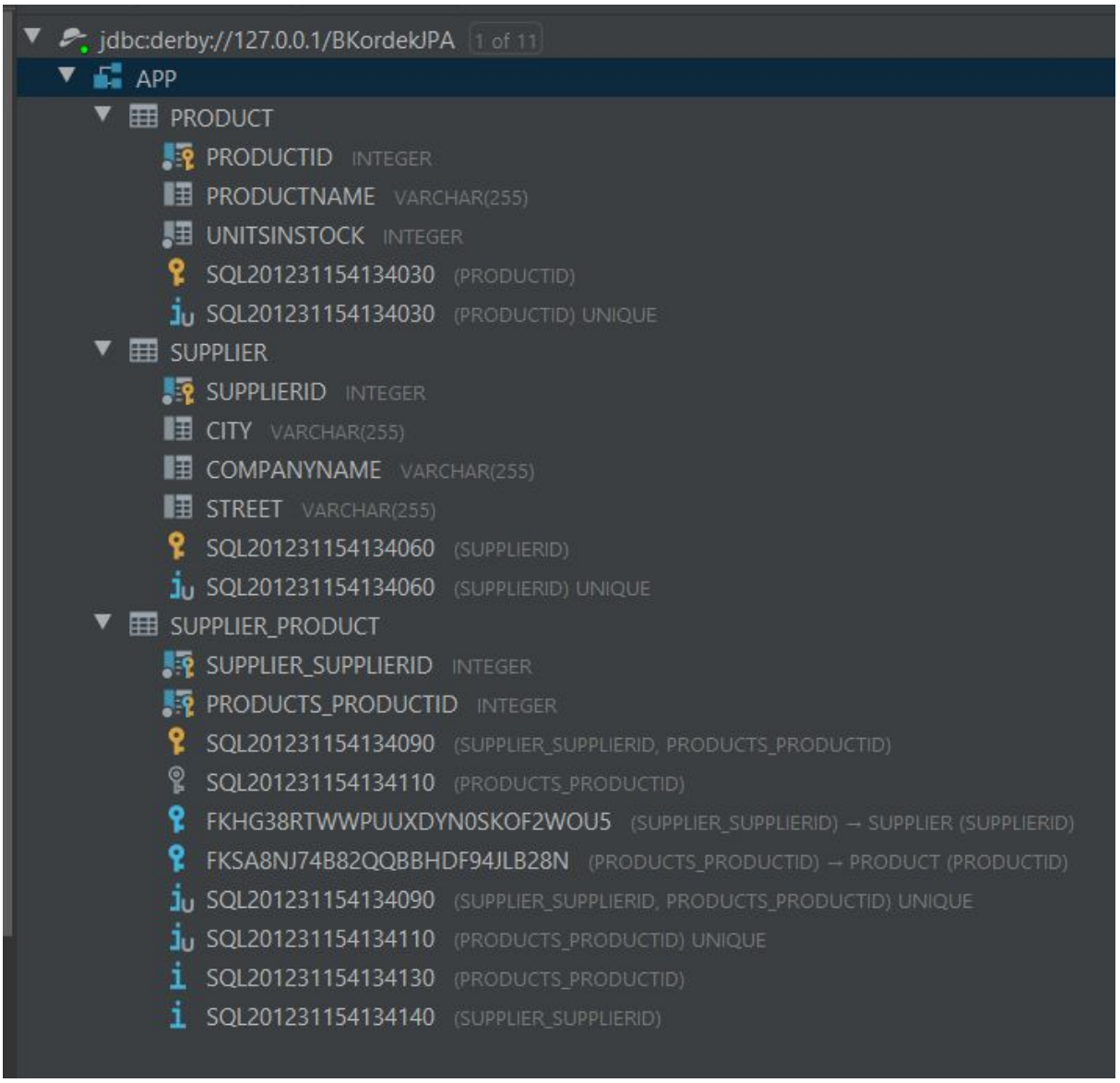


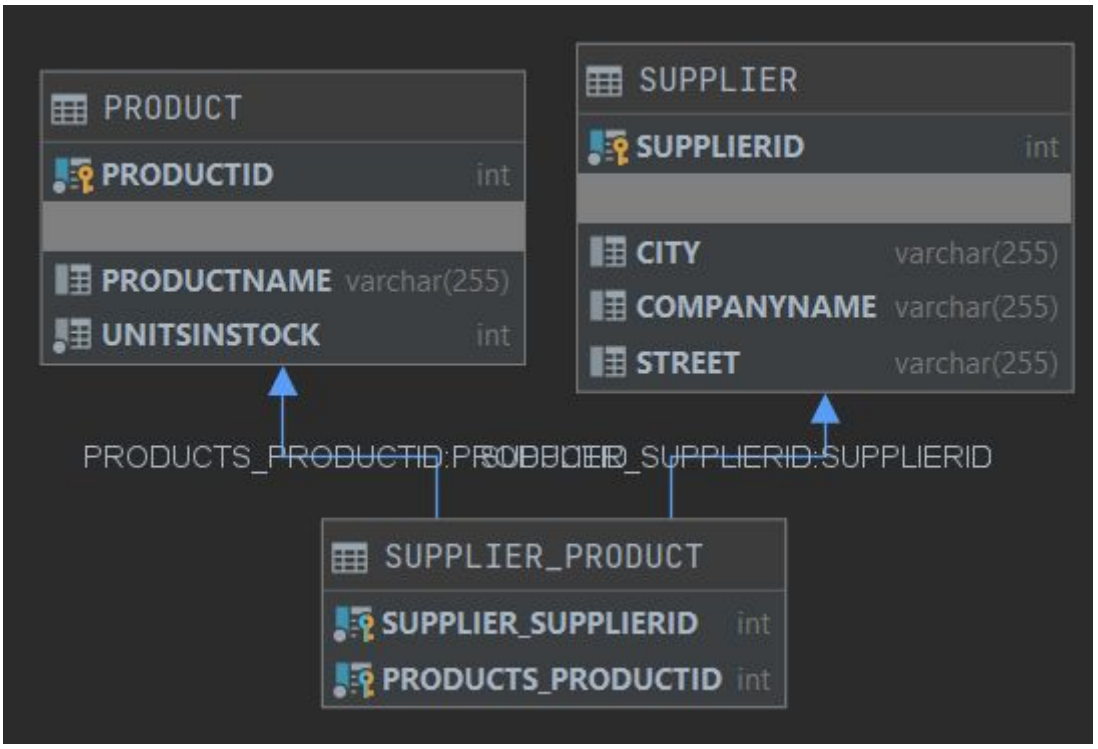


```
    /* insert collection
        row Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_SupplierID, products_ProductID)
        values
            (?, ?)
Hibernate:
    /* insert collection
        row Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_SupplierID, products_ProductID)
        values
            (?, ?)
Hibernate:
    /* insert collection
        row Supplier.products */ insert
        into
            Supplier_Product
            (Supplier_SupplierID, products_ProductID)
        values
            (?, ?)
querying all the managed entities...
executing: from Supplier
Hibernate:
    /*
from
    Supplier */ select
```

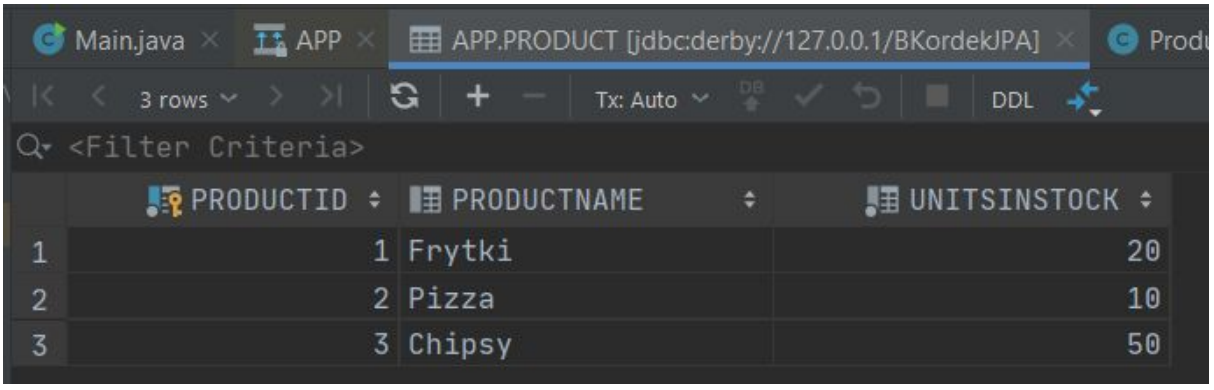

Run:
Main
executing: from Supplier
Hibernate:
/*
from
Supplier */ select
supplier0_.SupplierID as supplier1_1_,
supplier0_.City as city2_1_,
supplier0_.CompanyName as companyn3_1_,
supplier0_.Street as street4_1_
from
Supplier supplier0_
Supplier{SupplierID=4, CompanyName='Tesco', Street='Kapelanka', City='Kraków'}
Supplier{SupplierID=5, CompanyName='Żabka', Street='Sołtysowka', City='Kraków'}
executing: from Product
Hibernate:
/*
from
Product */ select
product0_.ProductID as 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

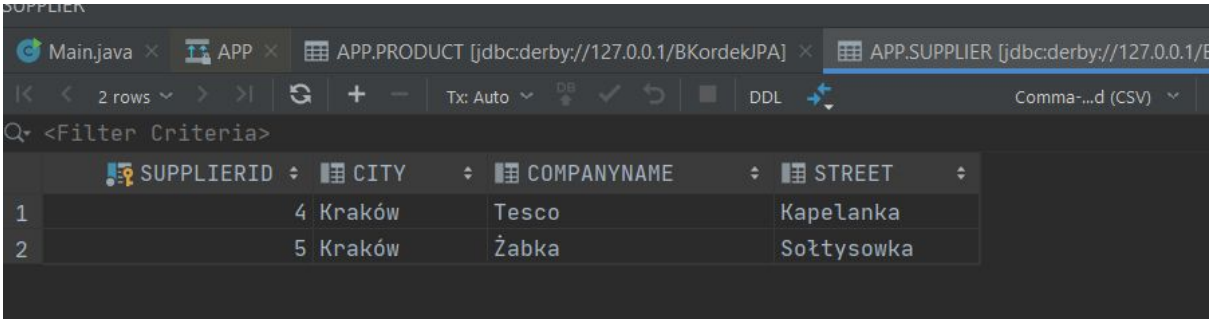
Schemat bazy danych

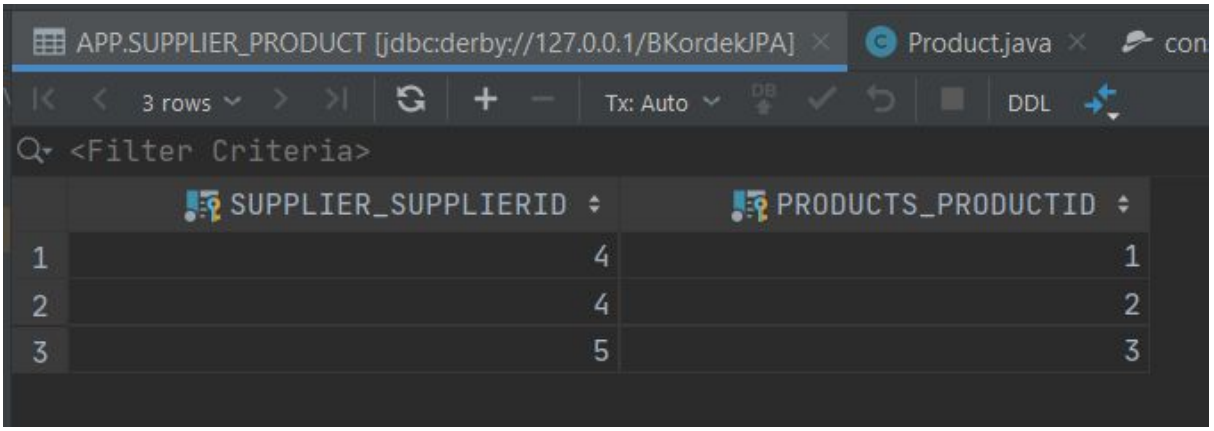




Zawartość tabel



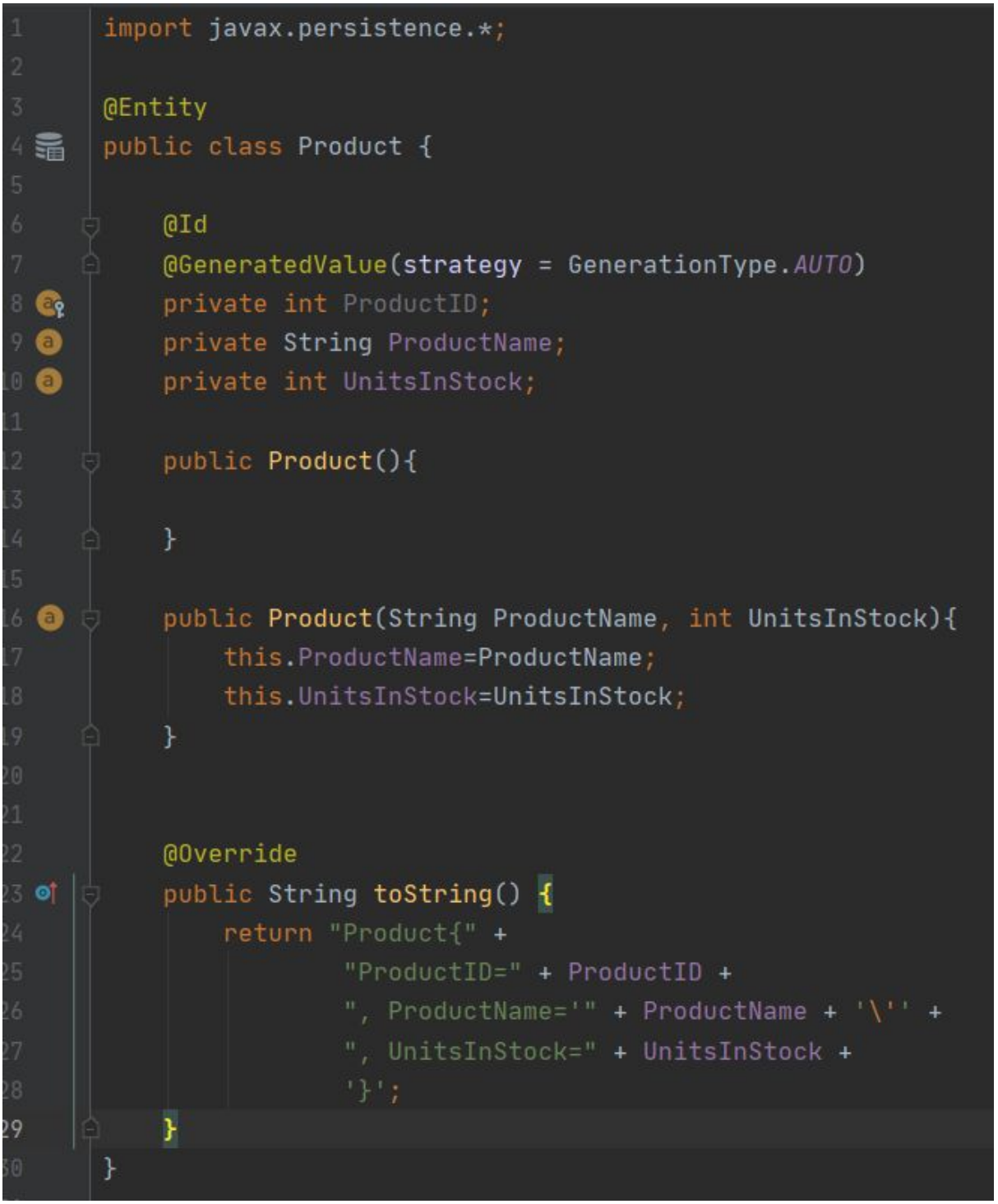




Jak widać powyżej, została stworzona tabela łącznikowa *SUPPLIER_PRODUCT* zawierająca klucze obce *SupplierID* i przyporządkowane *ProductID*.

b) Wariant bez tabeli łącznikowej

Klasa *Product*

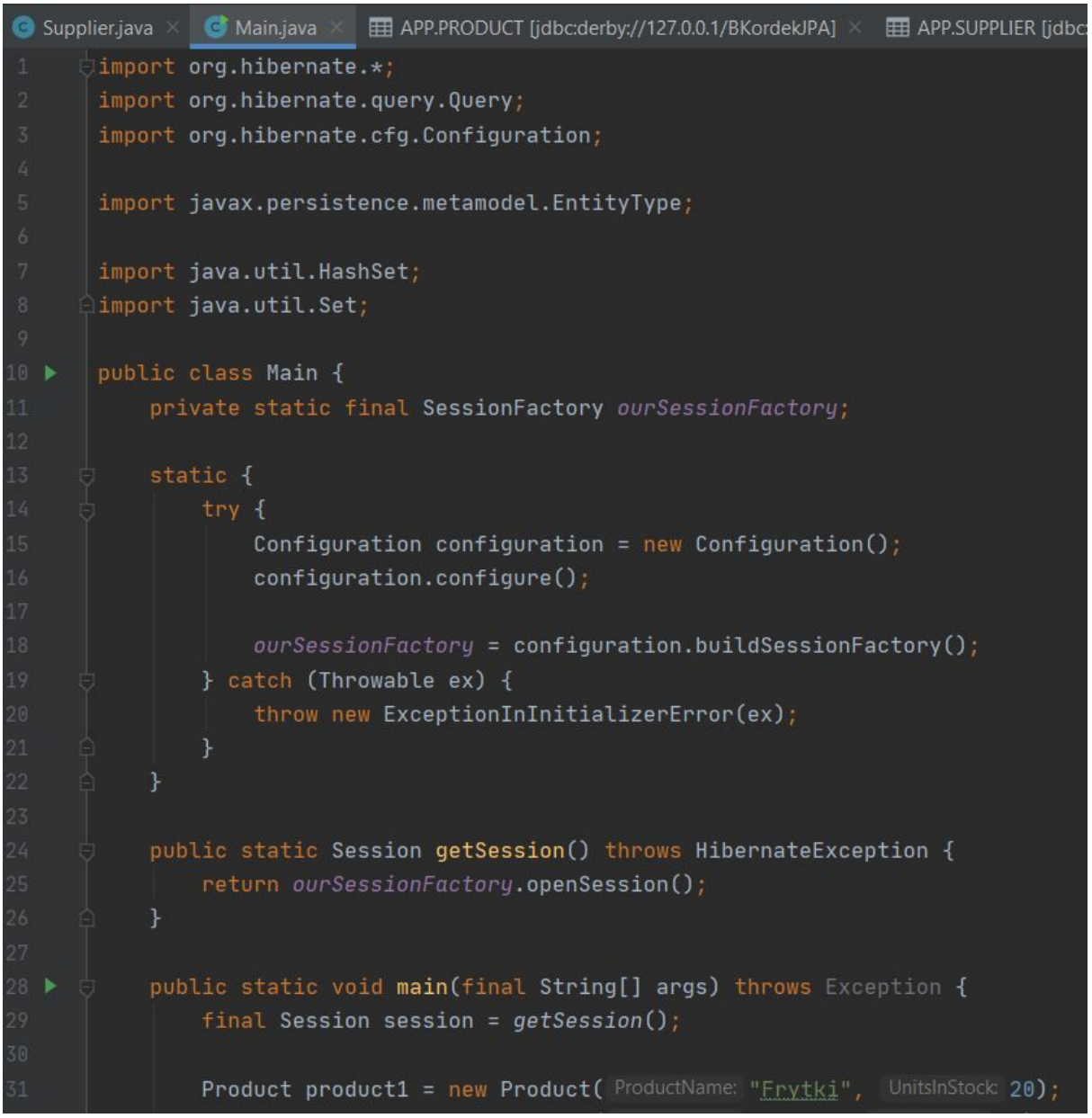


```java
import javax.persistence.*;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsInStock;

    public Product(){

    }

    public Product(String ProductName, int UnitsInStock){
        this.ProductName=ProductName;
        this.UnitsInStock=UnitsInStock;
    }


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

Klasa *Supplier*

```java
@Entity
public class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    @JoinColumn(name="SUPPLIER_FK")
    private Set<Product> products;

    public Supplier(){

    }

    public Supplier(String CompanyName, String Street, String City, Set<Product> products){
        this.CompanyName = CompanyName;
        this.Street = Street;
        this.City = City;
        this.products = products;
    }


    @Override
    public String toString() {
        return "Supplier{" +
                "SupplierID=" + SupplierID +
                ", CompanyName='" + CompanyName + '\'' +
                ", Street='" + Street + '\'' +
                ", City='" + City + '\'' +
```

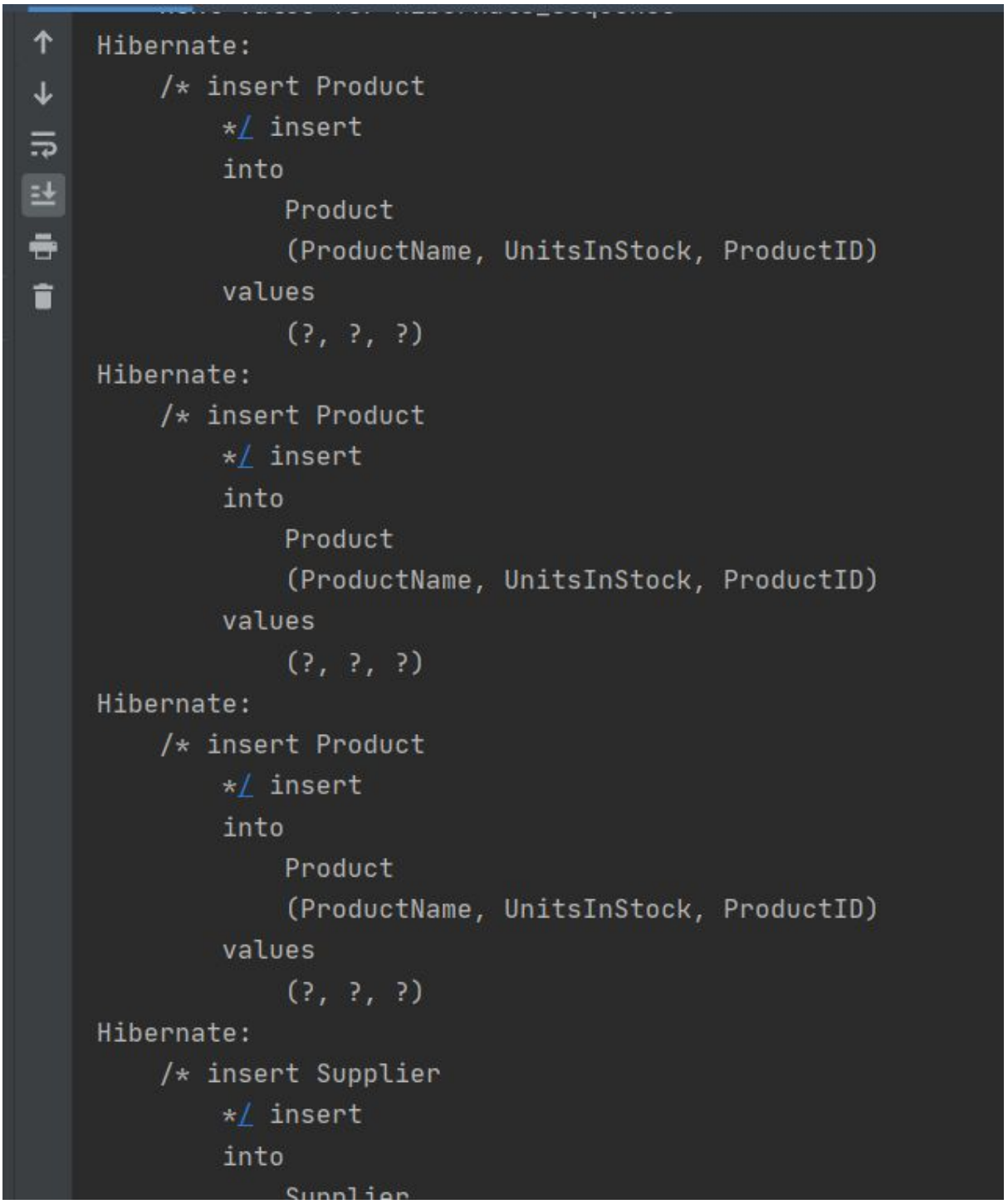
Klasa sterująca *Main* - stworzenie kilku produktów i dostarczenie ich przed dostawcę



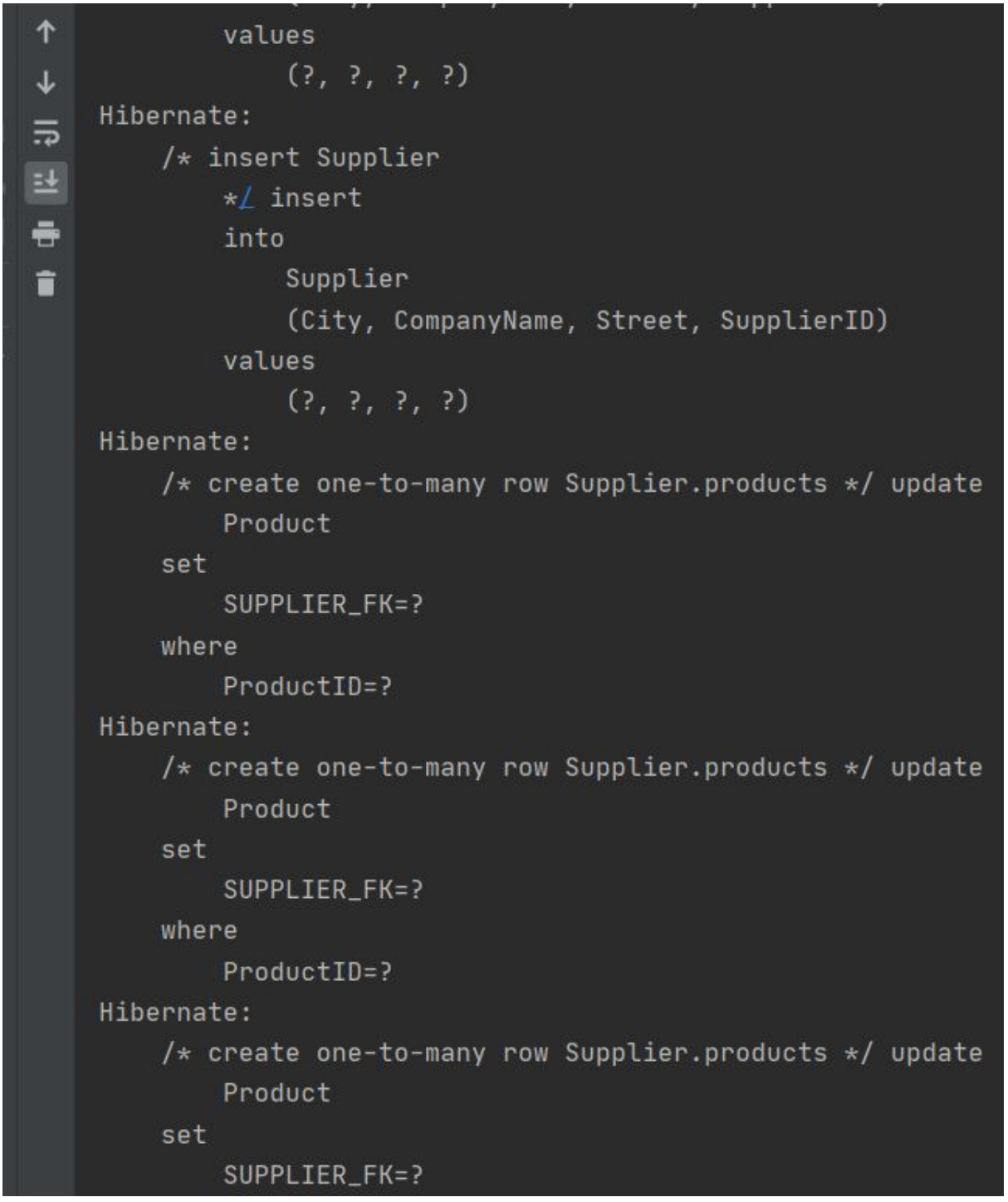
```
27
28      public static void main(final String[] args) throws Exception {
29          final Session session = getSession();
30
31          Product product1 = new Product( ProductName: "Frytki",  UnitsInStock: 20);
32          Product product2 = new Product( ProductName: "Pizza",  UnitsInStock: 10);
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", productSetForSupp
42          Supplier supplier2 = new Supplier( CompanyName: "Żabka",  Street: "Sołtysowka",  City: "Kraków", productSetForSup
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 {
```

```
56          try {
57              System.out.println("querying all the managed entities...");
58              final Metamodel metamodel = session.getSessionFactory().getMetamodel();
59              for (EntityType<?> entityType : metamodel.getEntities()) {
60                  final String entityName = entityType.getName();
61                  final Query query = session.createQuery( s: "from " + entityName);
62                  System.out.println("executing: " + query.getQueryString());
63                  for (Object o : query.list()) {
64                      System.out.println("  " + o);
65                  }
66              }
67          } finally {
68              session.close();
69          }
70      }
71  }
```

Wynik działania programu



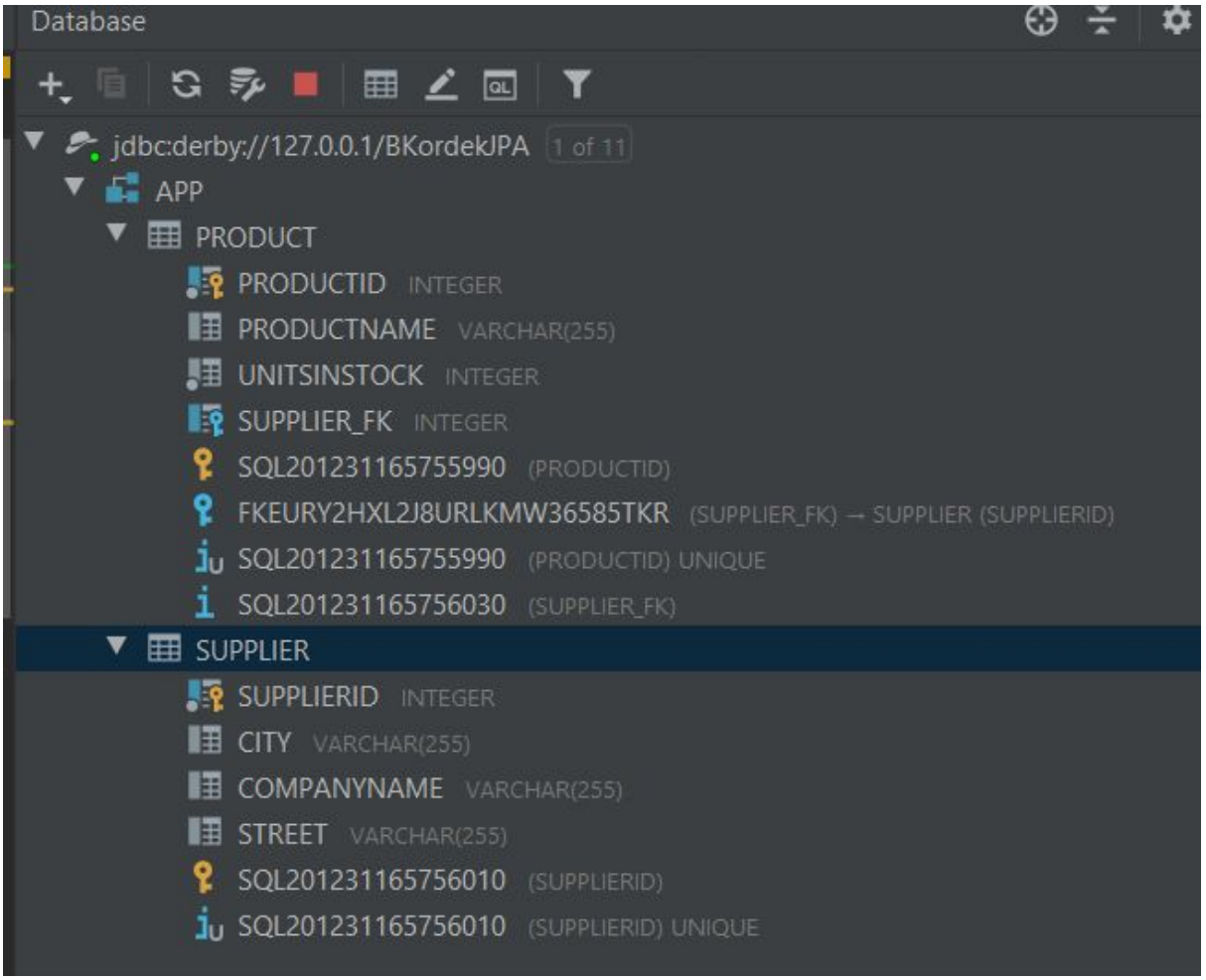
```
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitsInStock, ProductID)
        values
            (?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitsInStock, ProductID)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

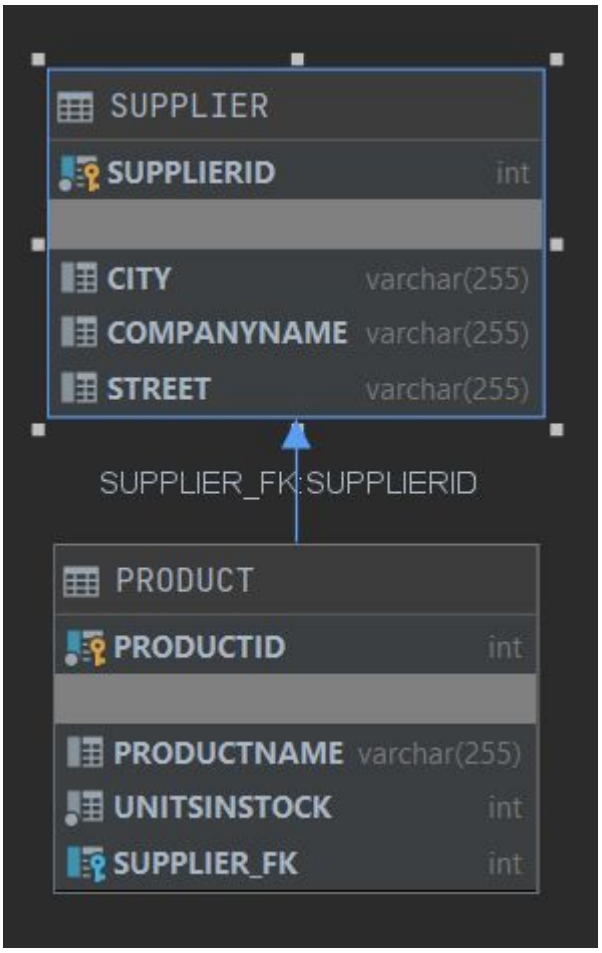


```
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
        */ insert
        into
            Supplier
            (City, CompanyName, Street, SupplierID)
        values
            (?, ?, ?, ?)
Hibernate:
    /* create one-to-many row Supplier.products */ update
        Product
    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=?
```

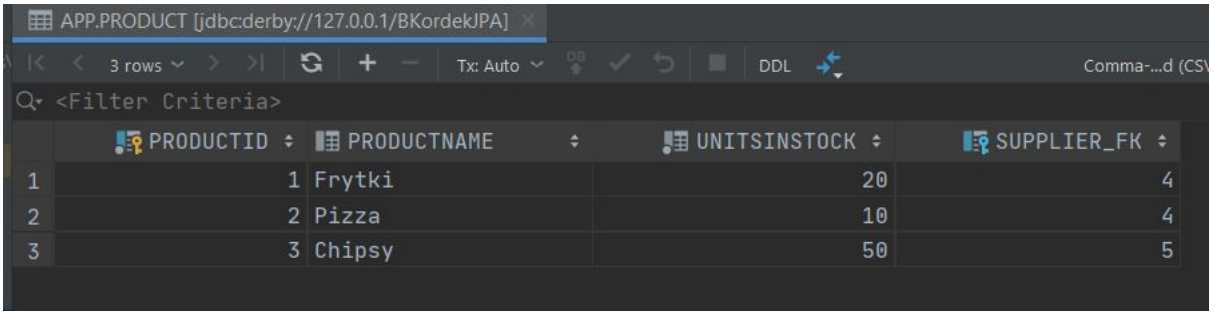
```
executing: from Supplier
Hibernate:
    /*
from
    Supplier */ select
        supplier0_.SupplierID as supplier1_1_,
        supplier0_.City as city2_1_,
        supplier0_.CompanyName as companyn3_1_,
        supplier0_.Street as street4_1_
    from
        Supplier supplier0_
  Supplier{SupplierID=4, 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
```

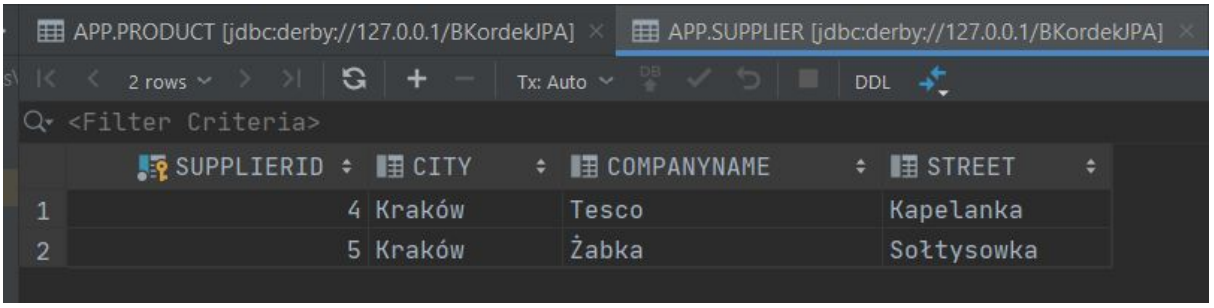
Schemat bazy danych





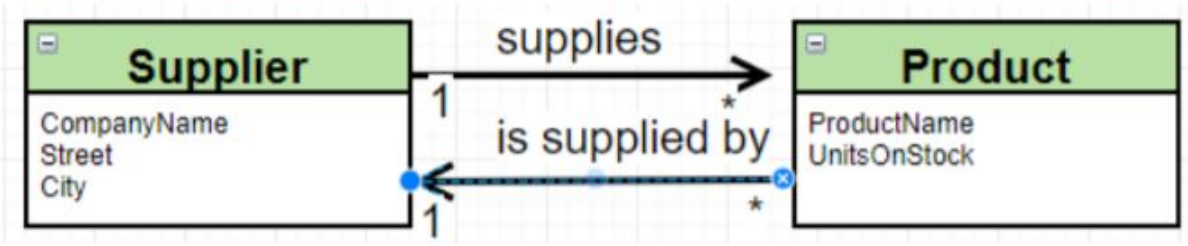
Zawartość tabel





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

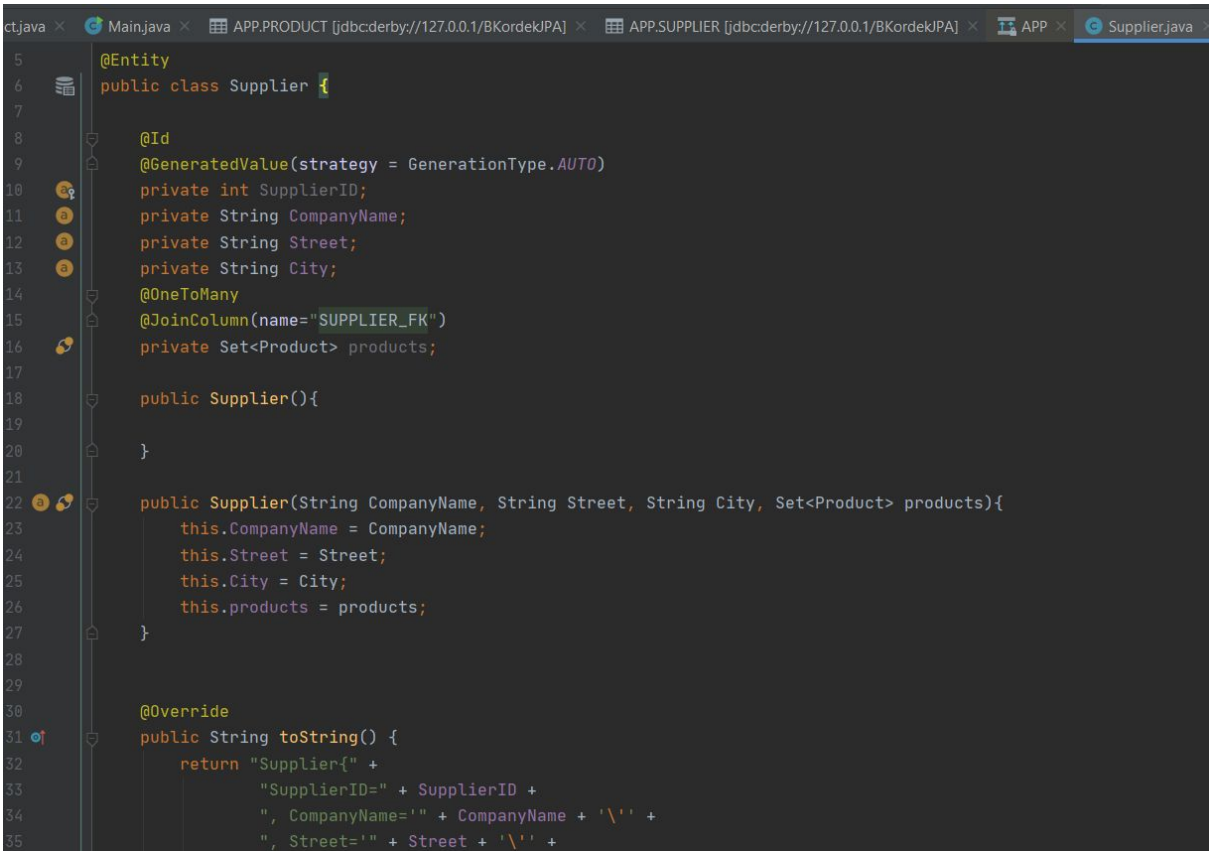


Klasa *Product*

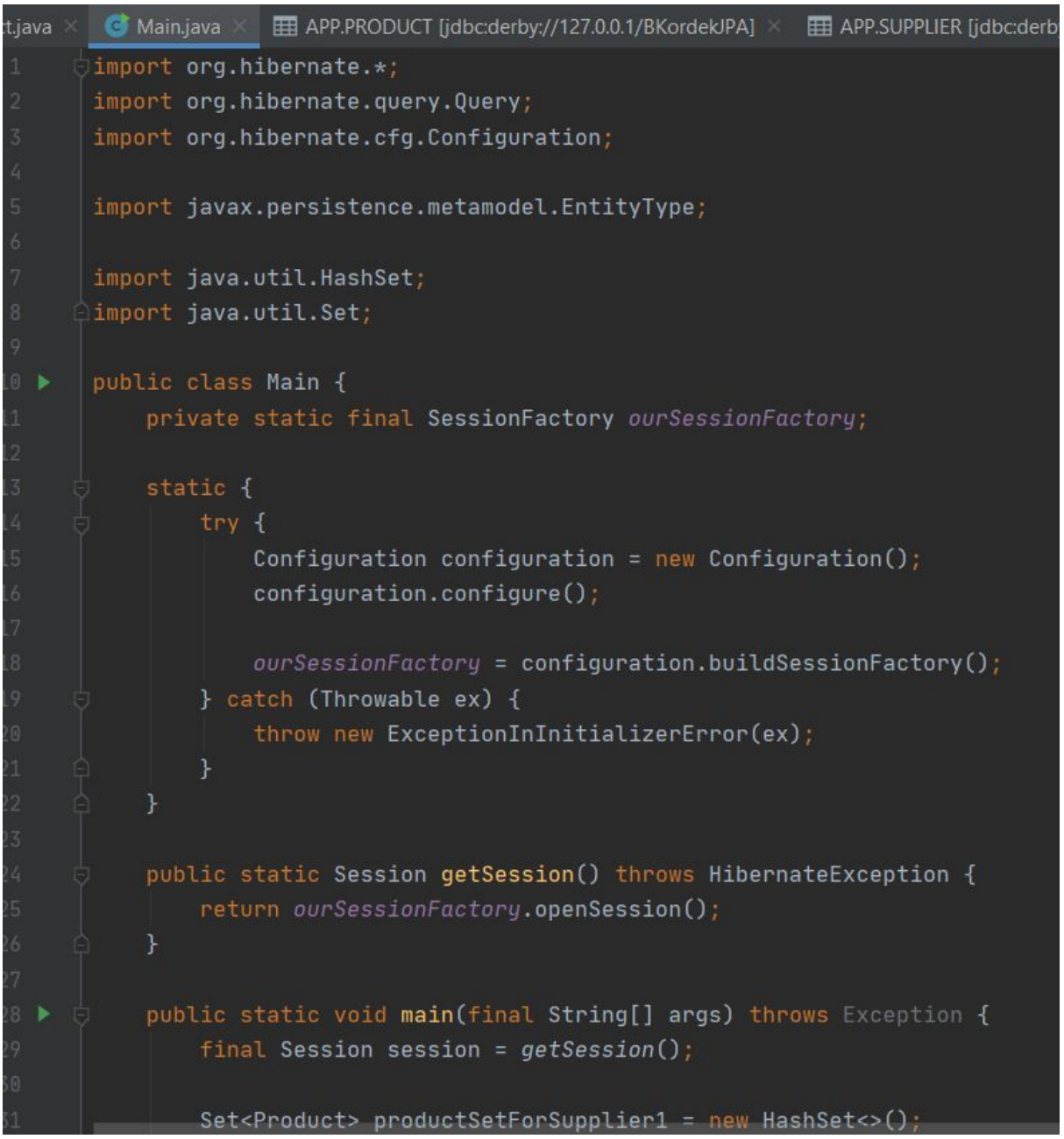
Product.java | Main.java | APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] | APP.SUPPLIER [jdbc:derby:/

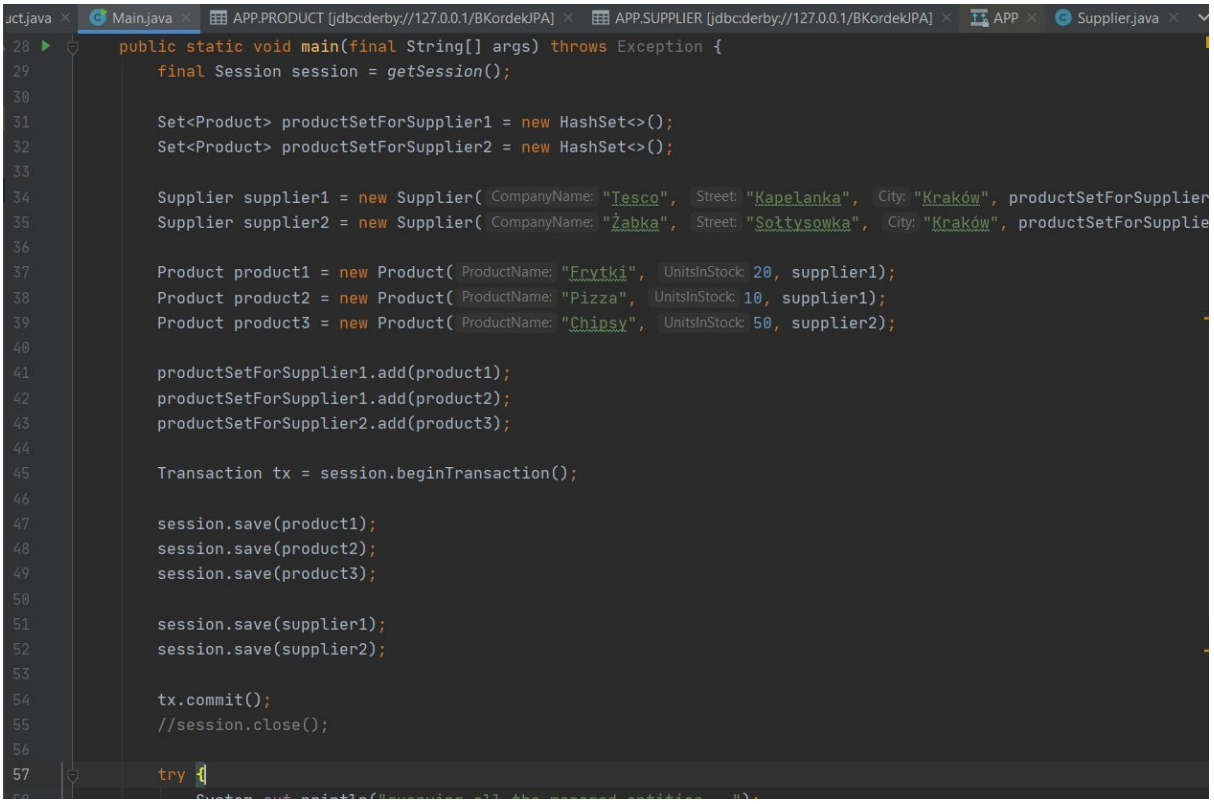
```java
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsInStock;
    @ManyToOne
    @JoinColumn(name="SUPPLIER_FK")
    private Supplier supplier;
    public Product(){

    }

    public Product(String ProductName, int UnitsInStock, Supplier supplier){
        this.ProductName=ProductName;
        this.UnitsInStock=UnitsInStock;
        this.supplier=supplier;
    }


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

Klasa *Supplier*



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

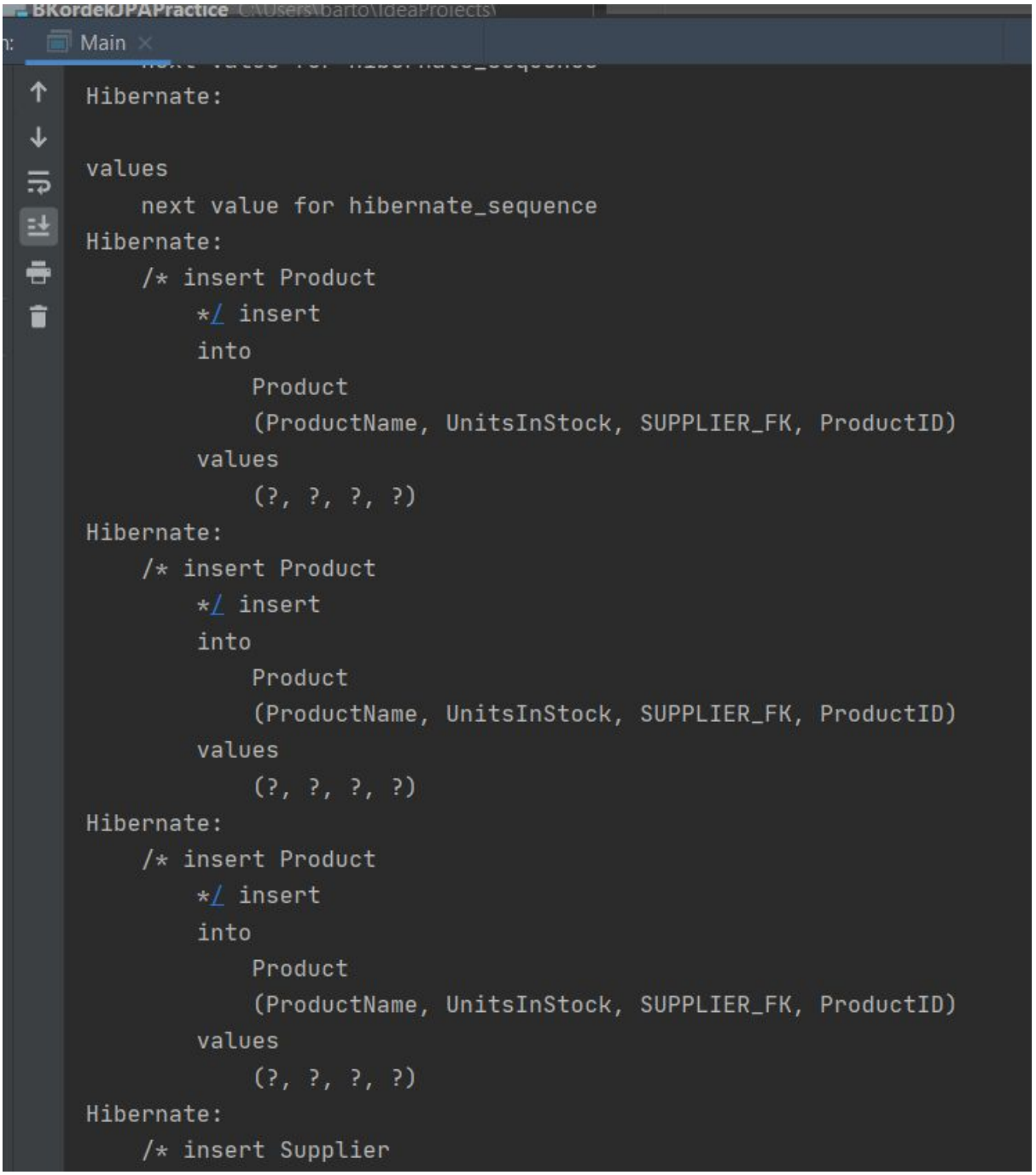


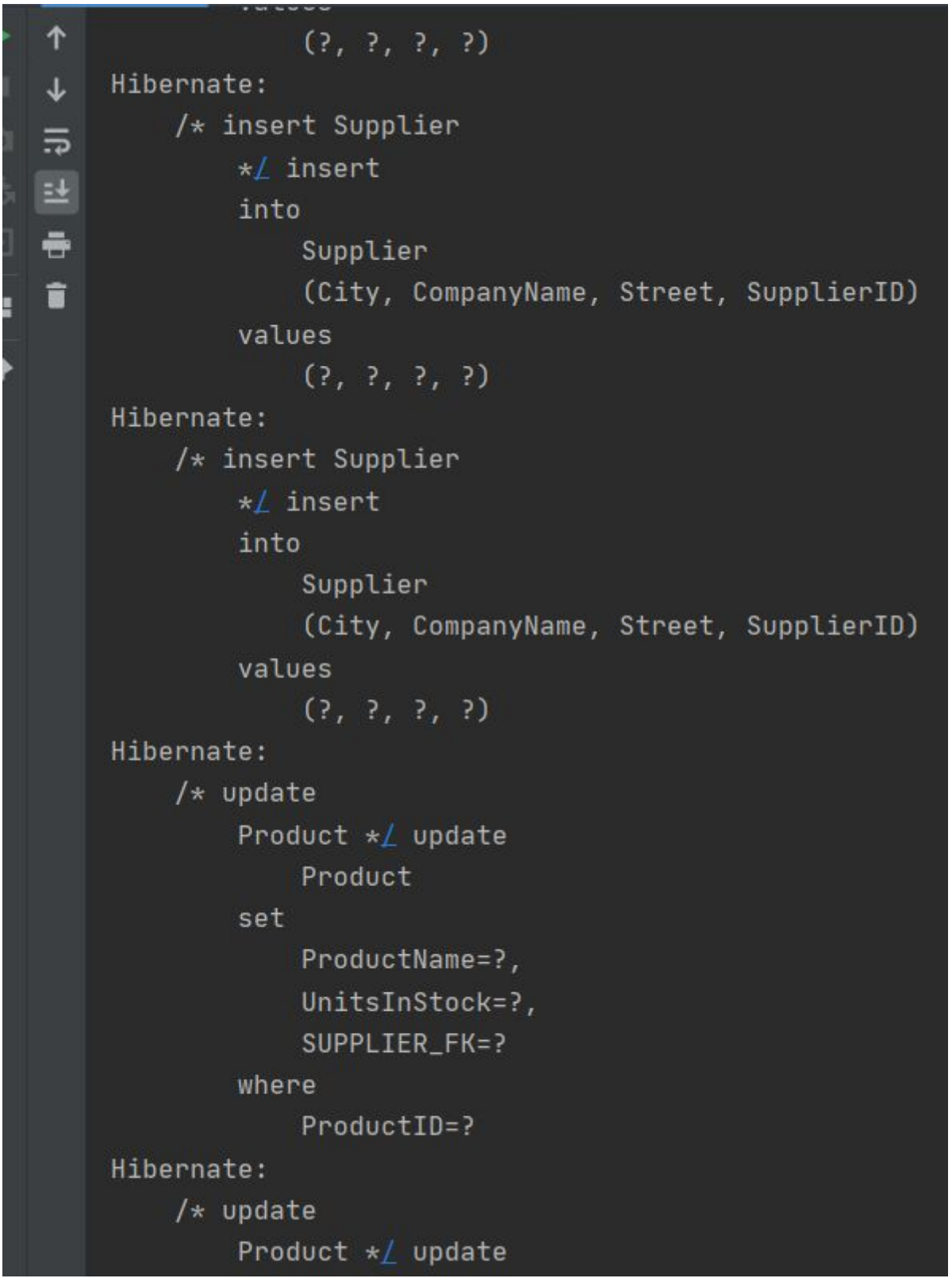


```
uct.java | Main.java | APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] | APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] | APP | Supplier.java

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", productSetForSupplier
35      Supplier supplier2 = new Supplier( CompanyName: "Żabka",  Street: "Sołtysowka",  City: "Kraków", productSetForSupplie
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 {
```

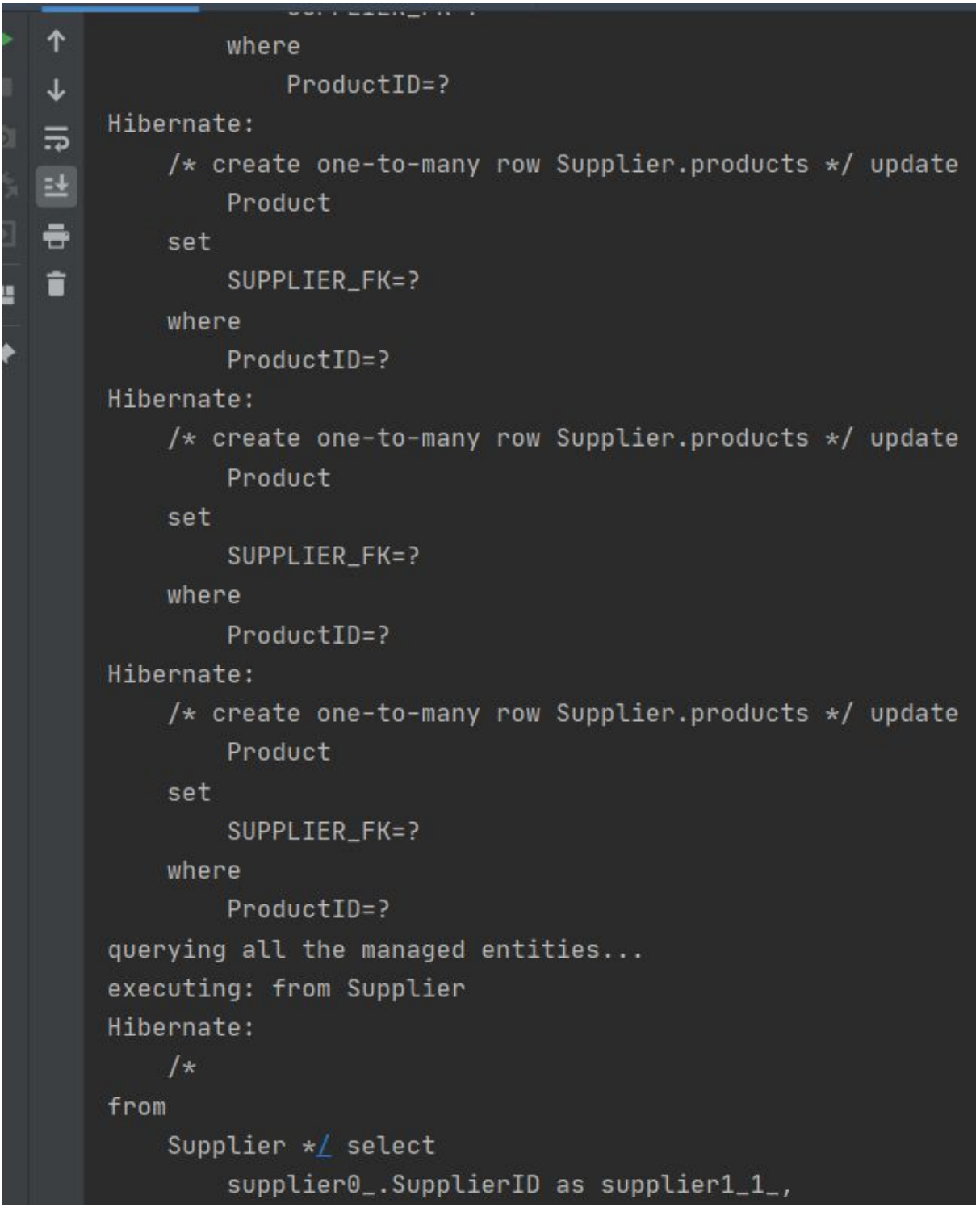
```
56
57          try {
58              System.out.println("querying all the managed entities...");
59              final Metamodel metamodel = session.getSessionFactory().getMetamodel();
60              for (EntityType<?> entityType : metamodel.getEntities()) {
61                  final String entityName = 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

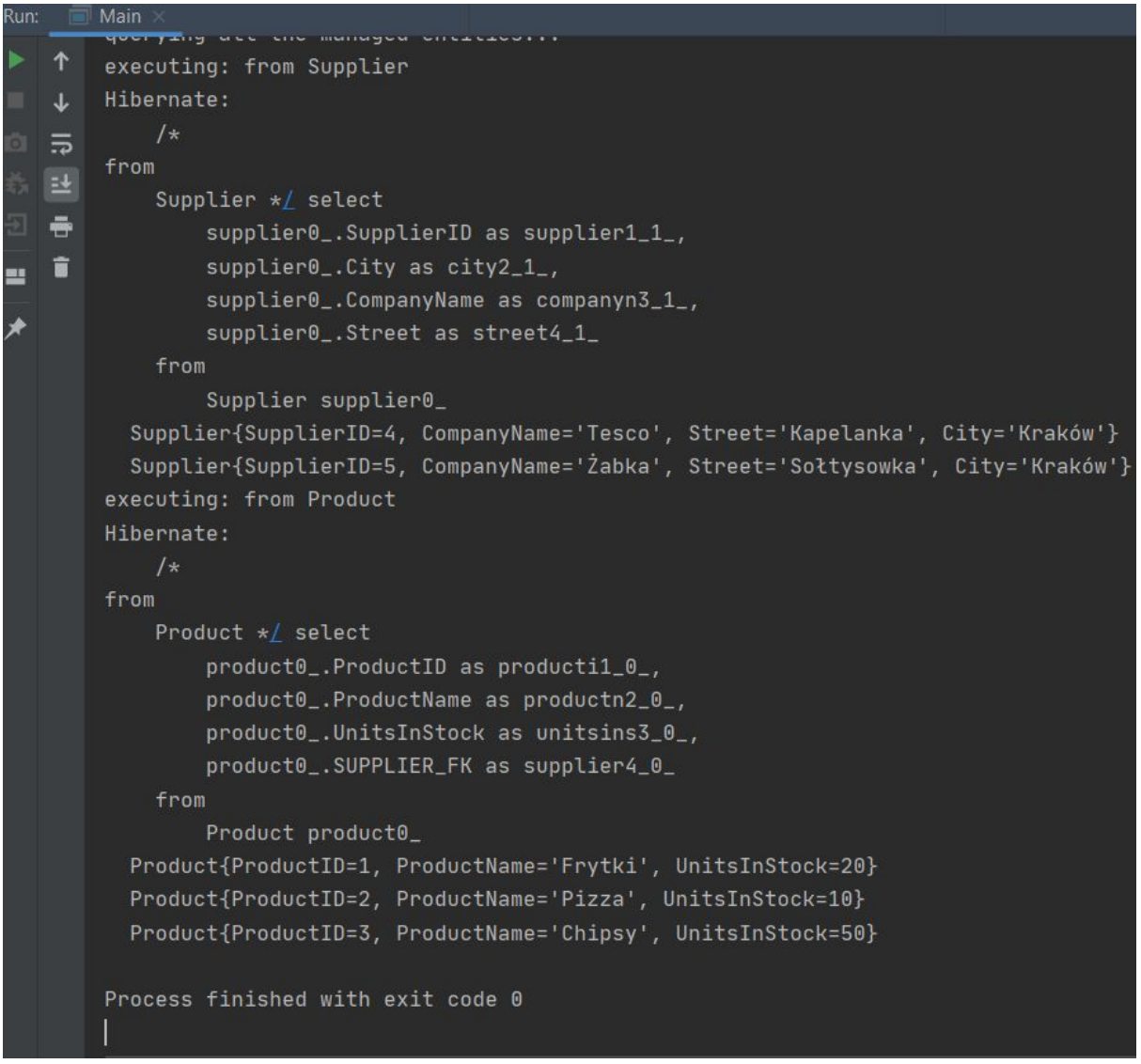




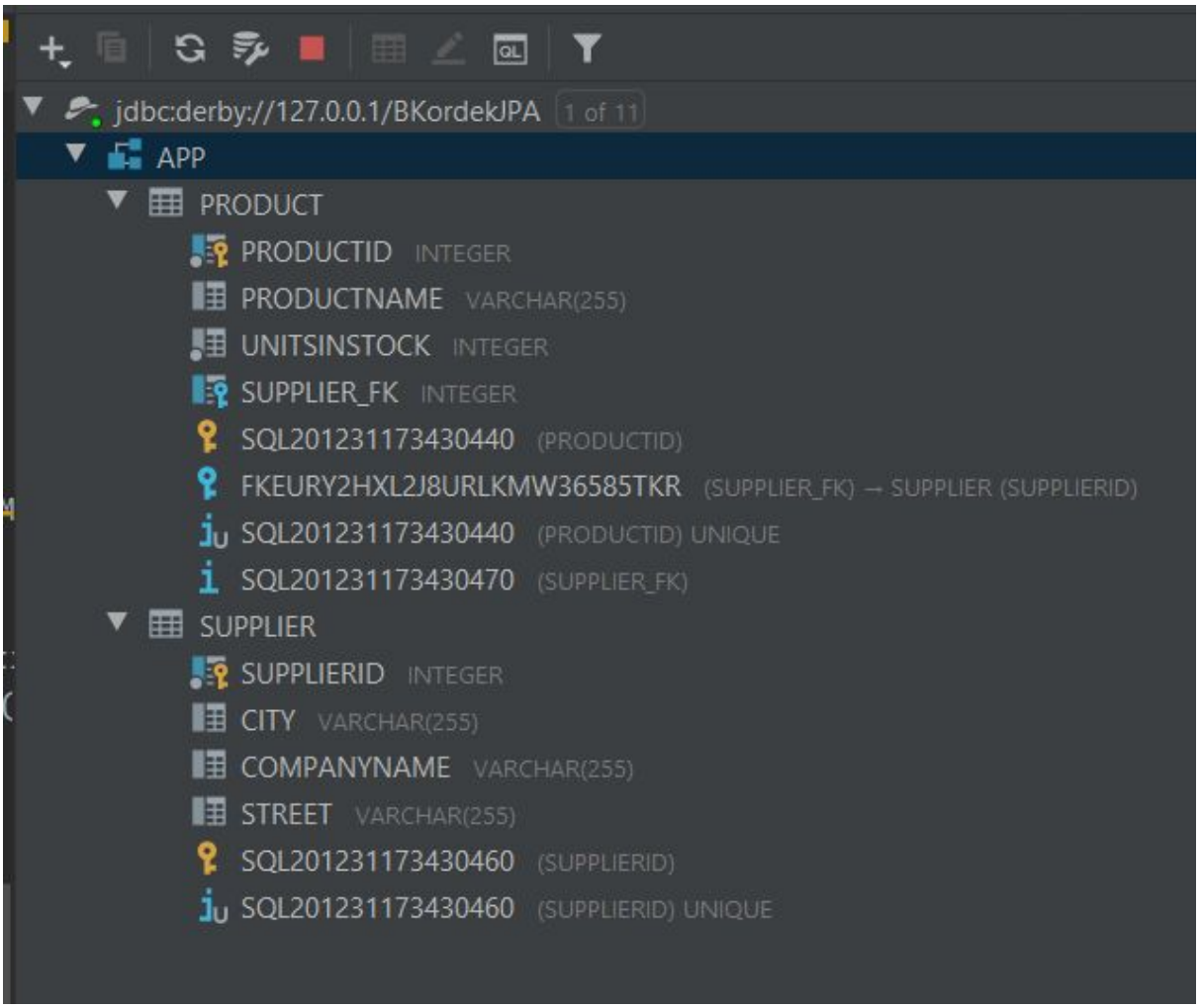
```
            (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
        */ insert
        into
            Supplier
            (City, CompanyName, Street, SupplierID)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert Supplier
        */ insert
        into
            Supplier
       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

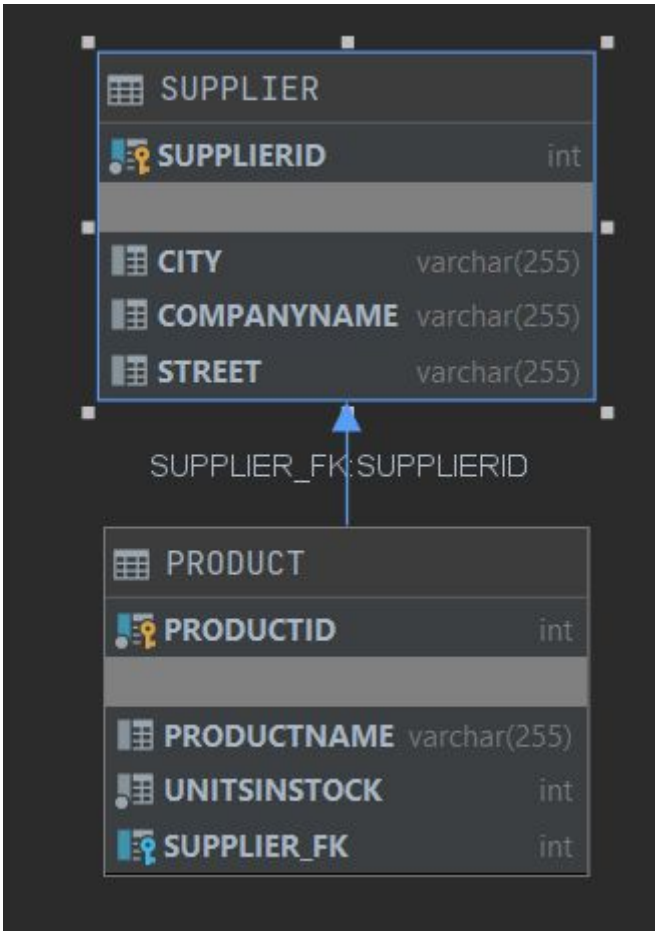


```
        where
            ProductID=?
Hibernate:
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?
    where
        ProductID=?
Hibernate:
    /* create one-to-many row Supplier.products */ update
        Product
    set
        SUPPLIER_FK=?
    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 */ select
        supplier0_.SupplierID as supplier1_1_,
```

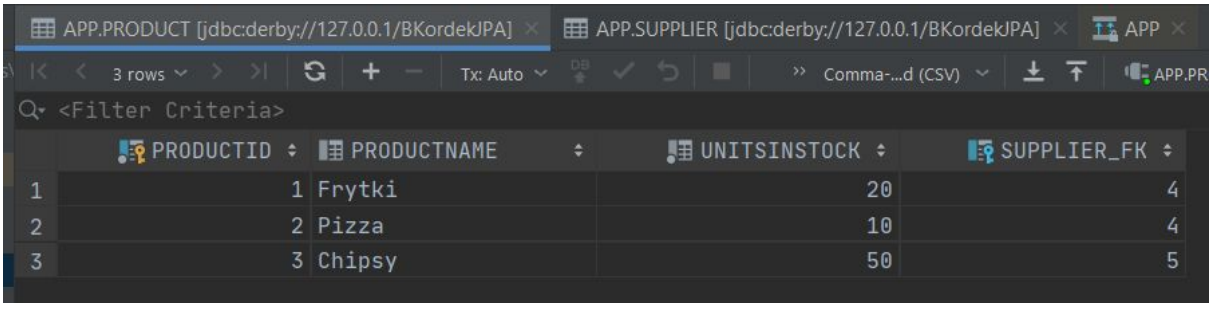

Run:
Main
executing: from Supplier
Hibernate:
/*
from
Supplier */ select
supplier0_.SupplierID as supplier1_1_,
supplier0_.City as city2_1_,
supplier0_.CompanyName as companyn3_1_,
supplier0_.Street as street4_1_
from
Supplier supplier0_
Supplier{SupplierID=4, CompanyName='Tesco', Street='Kapelanka', City='Kraków'}
Supplier{SupplierID=5, CompanyName='Żabka', Street='Sołtysowka', City='Kraków'}
executing: from Product
Hibernate:
/*
from
Product */ select
product0_.ProductID as producti1_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

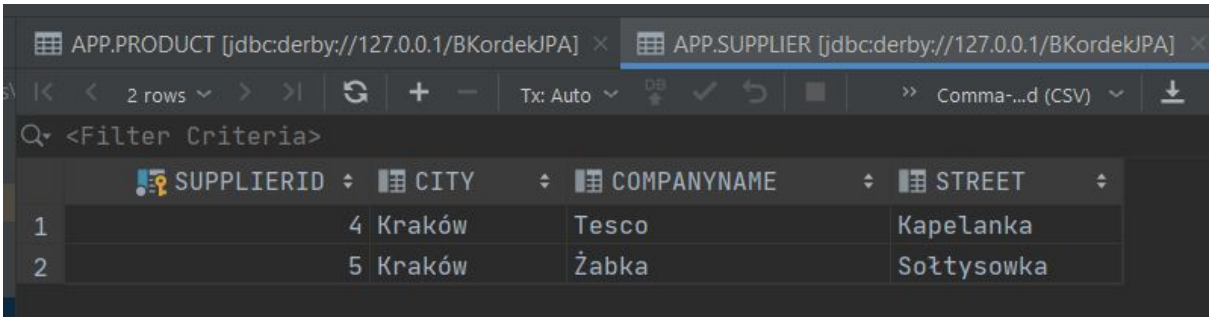
Schemat bazy danych





Zawartość tabel

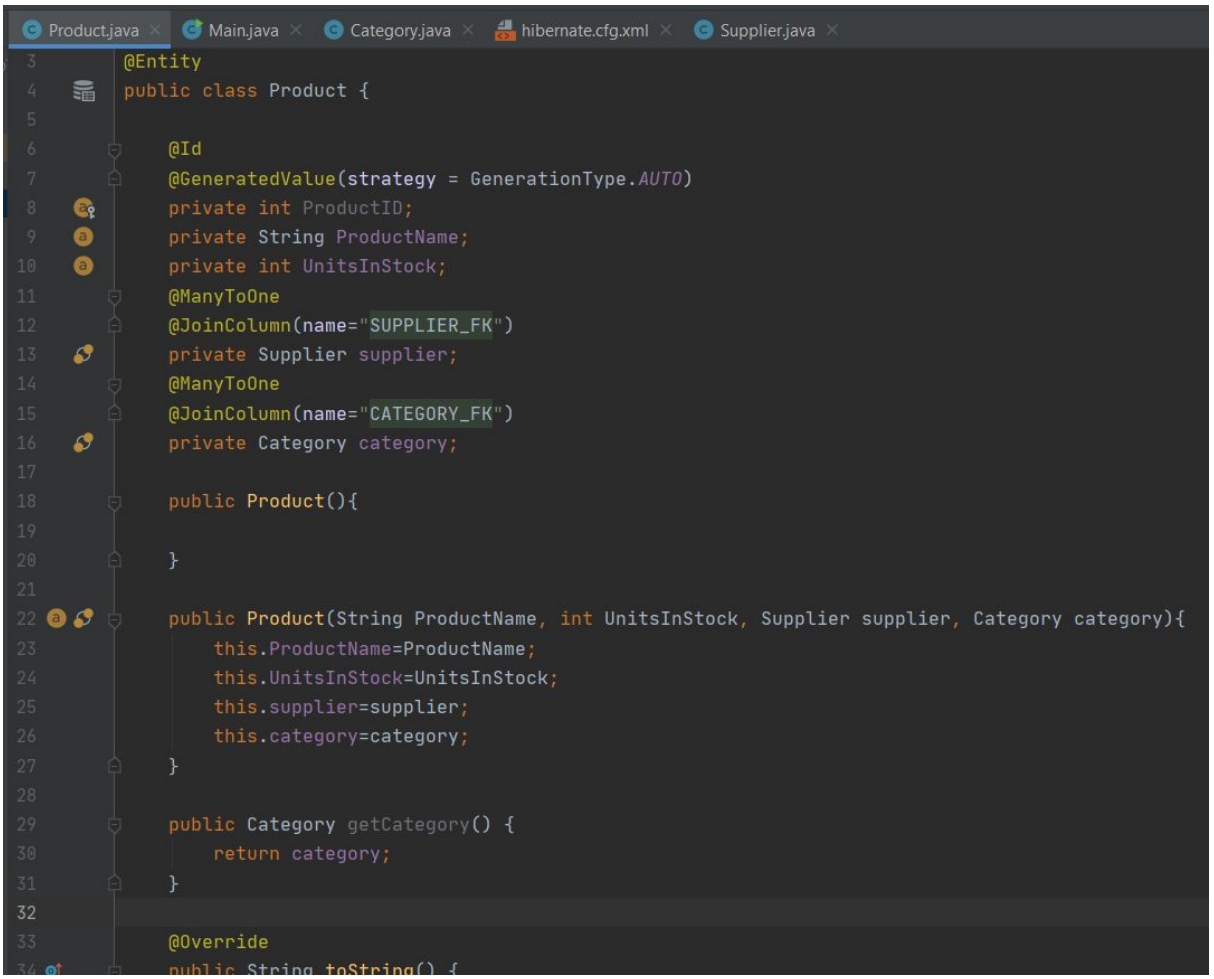




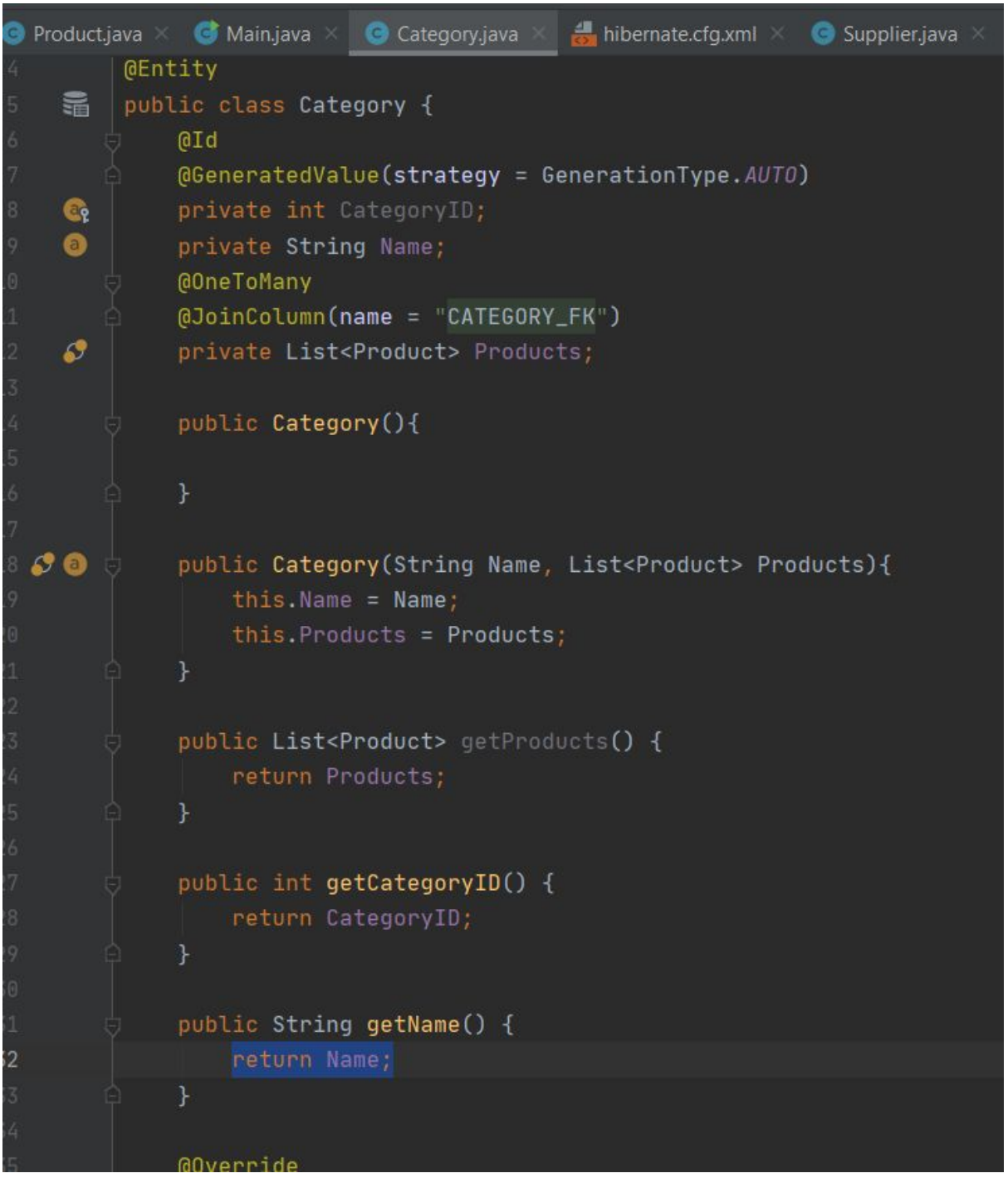
Z powyższego widać, że schemat jest taki sam jak w przypadku V. bez tabeli łącznikowej. Można natomiast dodać produkt przypisując do niego dostawcę i dostawcę przypisując do niego produkty.

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**



Klasa *Category*



Klasa *Supplier*

Product.java | Main.java | Category.java | hibernate.cfg.xml | Supplier.java

```java
@Entity
public class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    @JoinColumn(name="SUPPLIER_FK")
    private Set<Product> products;

    public Supplier(){

    }

    public Supplier(String CompanyName, String Street, String City, Set<Product> products){
        this.CompanyName = CompanyName;
        this.Street = Street;
        this.City = City;
        this.products = products;
    }

    public int getSupplierID() {
        return SupplierID;
    }

    public String getCompanyName() {
        return CompanyName;
    }
```

Modyfikacja Pliku konfiguracyjnego - dodanie klasy *Category* do mappingu

Product.java | Main.java | Category.java | hibernate.cfg.xml | Supplier.java

```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>
        <mapping class="Category"></mapping>
    </session-factory>
</hibernate-configuration>
```

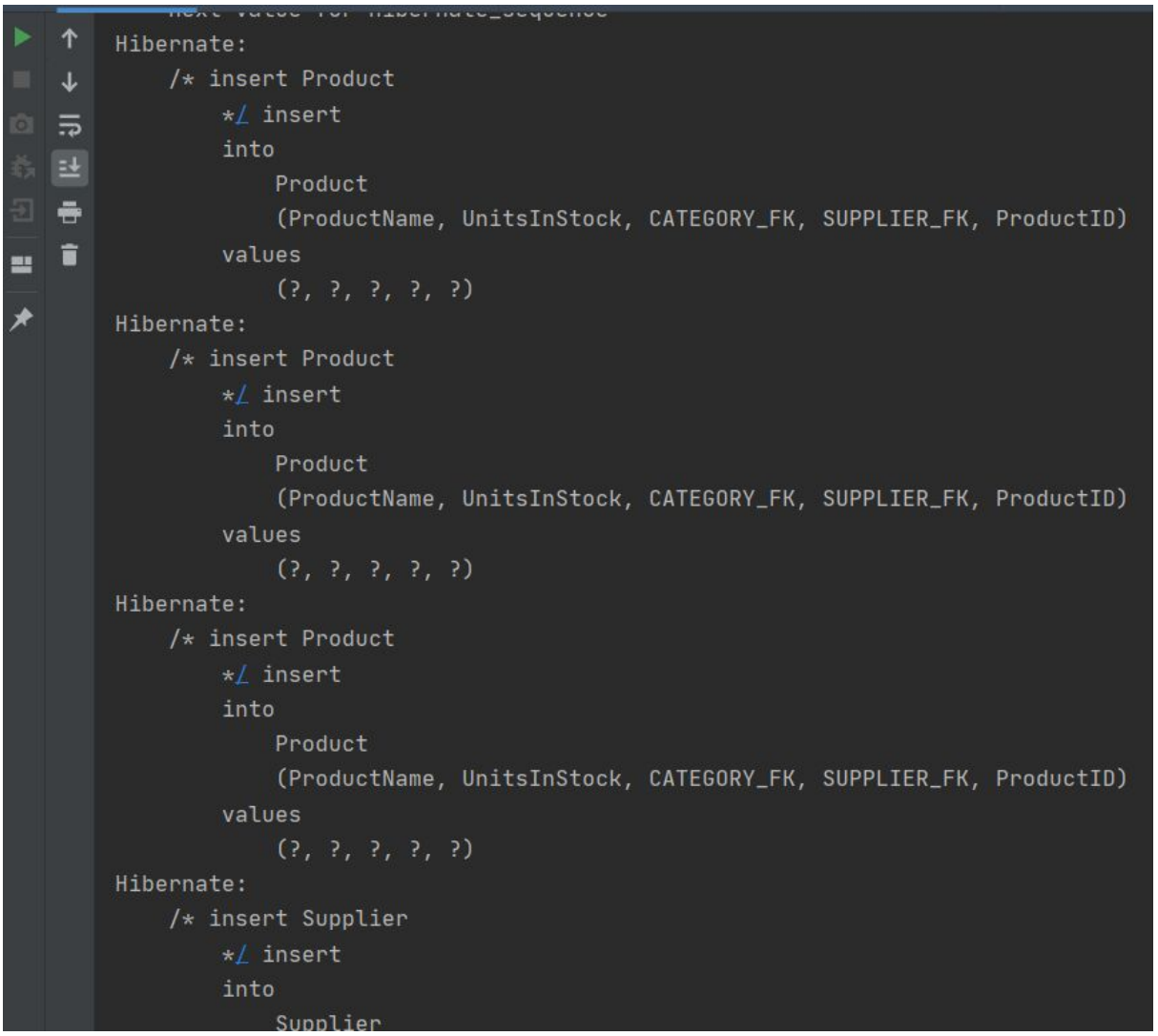
Klasa sterująca *Main* - **stworzenie kilku produktów i kilku kategorii**

Product.java | Main.java | Category.java | hibernate.cfg.xml | Supplier.java

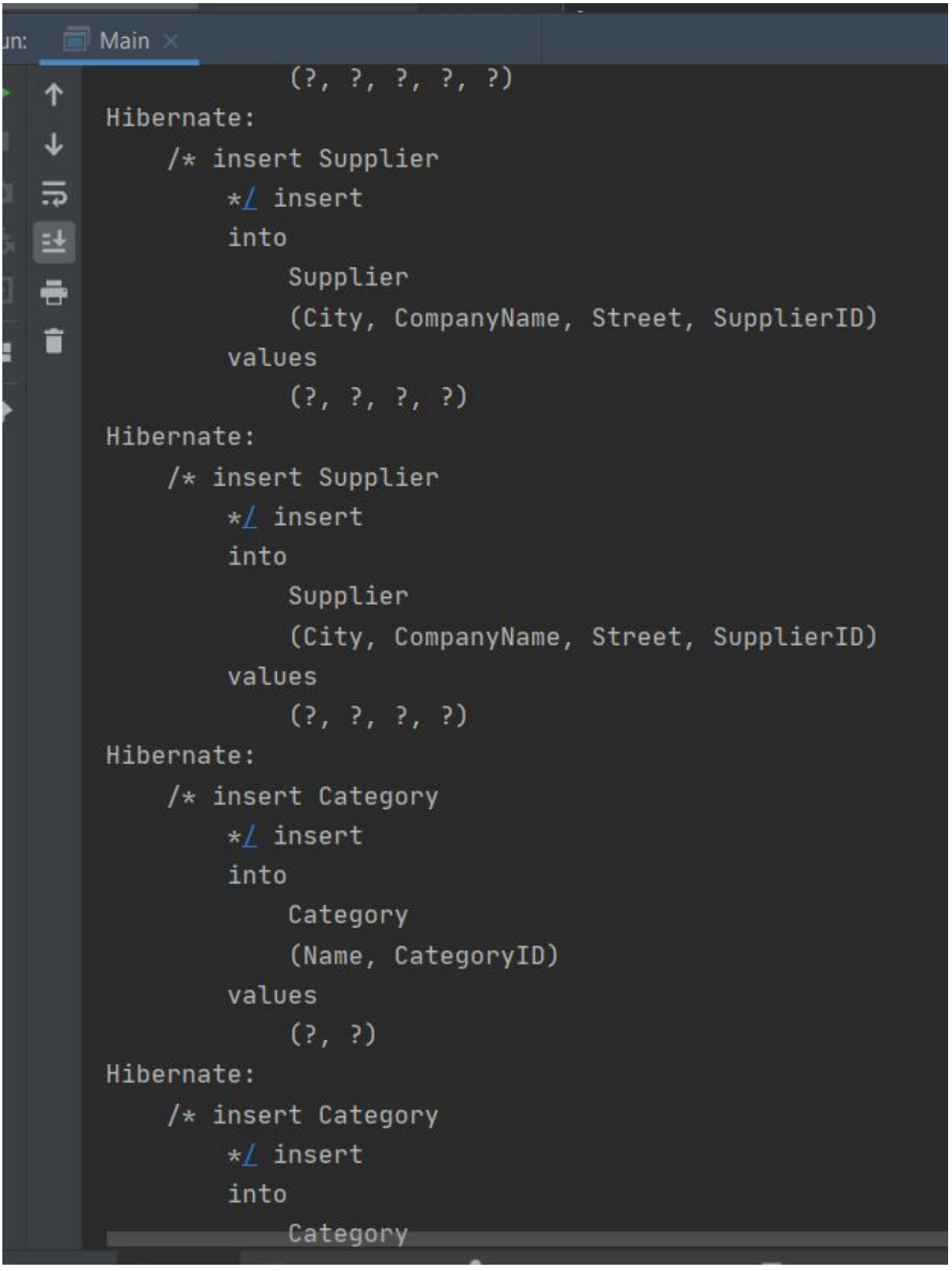
```
}

public static void main(final String[] args) throws Exception {
    final Session session = getSession();

    Set<Product> productSetForSupplier1 = new HashSet<>();
    Set<Product> productSetForSupplier2 = new HashSet<>();

    List<Product> productListForCategory1 = new ArrayList<>();
    List<Product> productListForCategory2 = new ArrayList<>();

    Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
    Category category2 = new Category( Name: "Przekąski", productListForCategory2);

    Supplier supplier1 = new Supplier( CompanyName: "Tesco", Street: "Kapelanka", City: "Kraków", productSetForSupplier1);
    Supplier supplier2 = new Supplier( CompanyName: "Żabka", Street: "Sołtysowka", City: "Kraków", productSetForSupplier2);

    Product product1 = new Product( ProductName: "Frytki", UnitsInStock: 20, supplier1, category1);
    Product product2 = new Product( ProductName: "Pizza", UnitsInStock: 10, supplier1, category1);
    Product product3 = new Product( ProductName: "Chipsy", UnitsInStock: 50, supplier2, category2);

    productSetForSupplier1.add(product1);
    productSetForSupplier1.add(product2);
    productSetForSupplier2.add(product3);

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

    Transaction tx = session.beginTransaction();
```

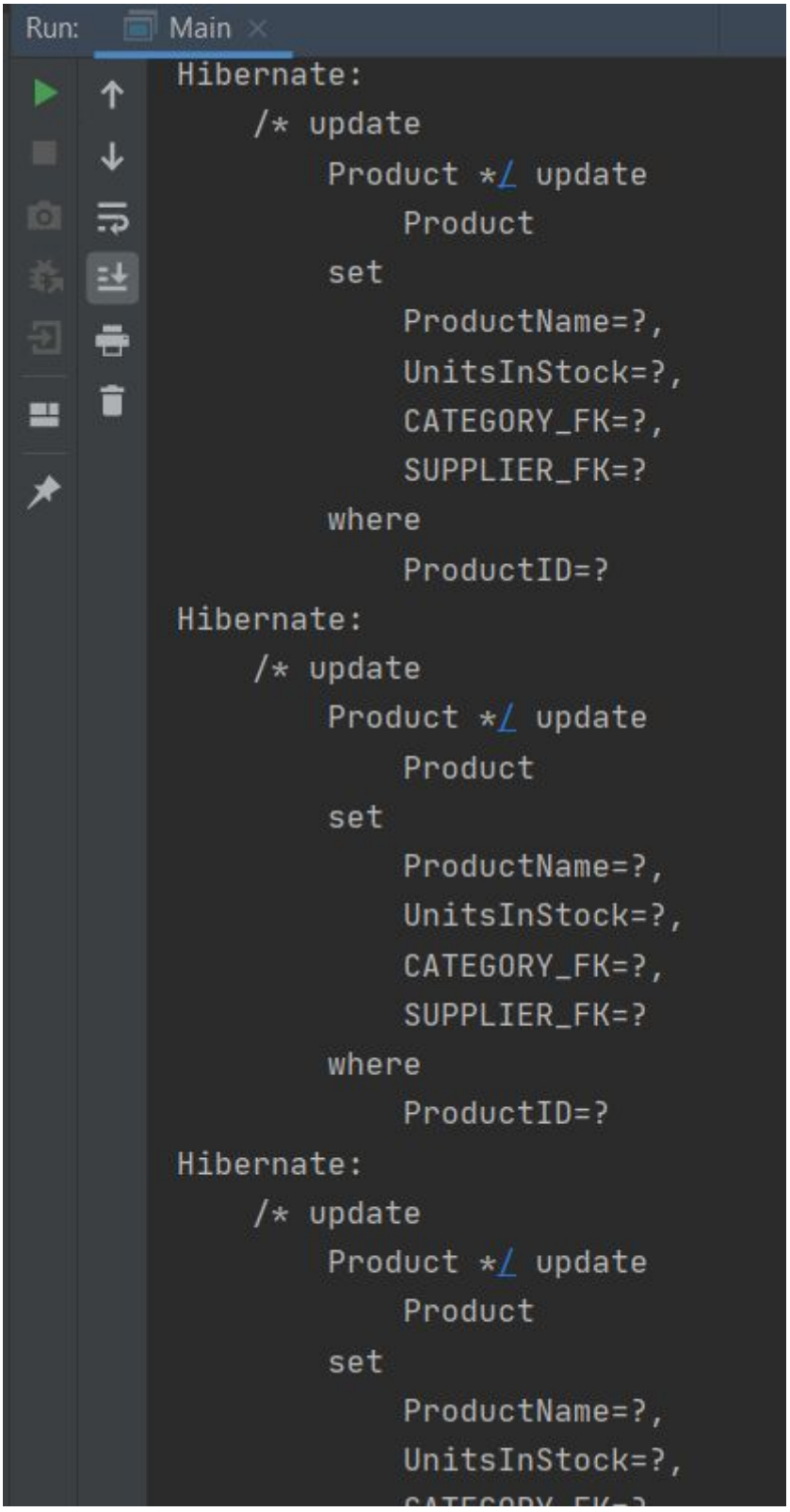
```
    Transaction tx = session.beginTransaction();

    session.save(product1);
    session.save(product2);
    session.save(product3);

    session.save(supplier1);
    session.save(supplier2);

    session.save(category1);
    session.save(category2);

    tx.commit();
    //session.close();

    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
```

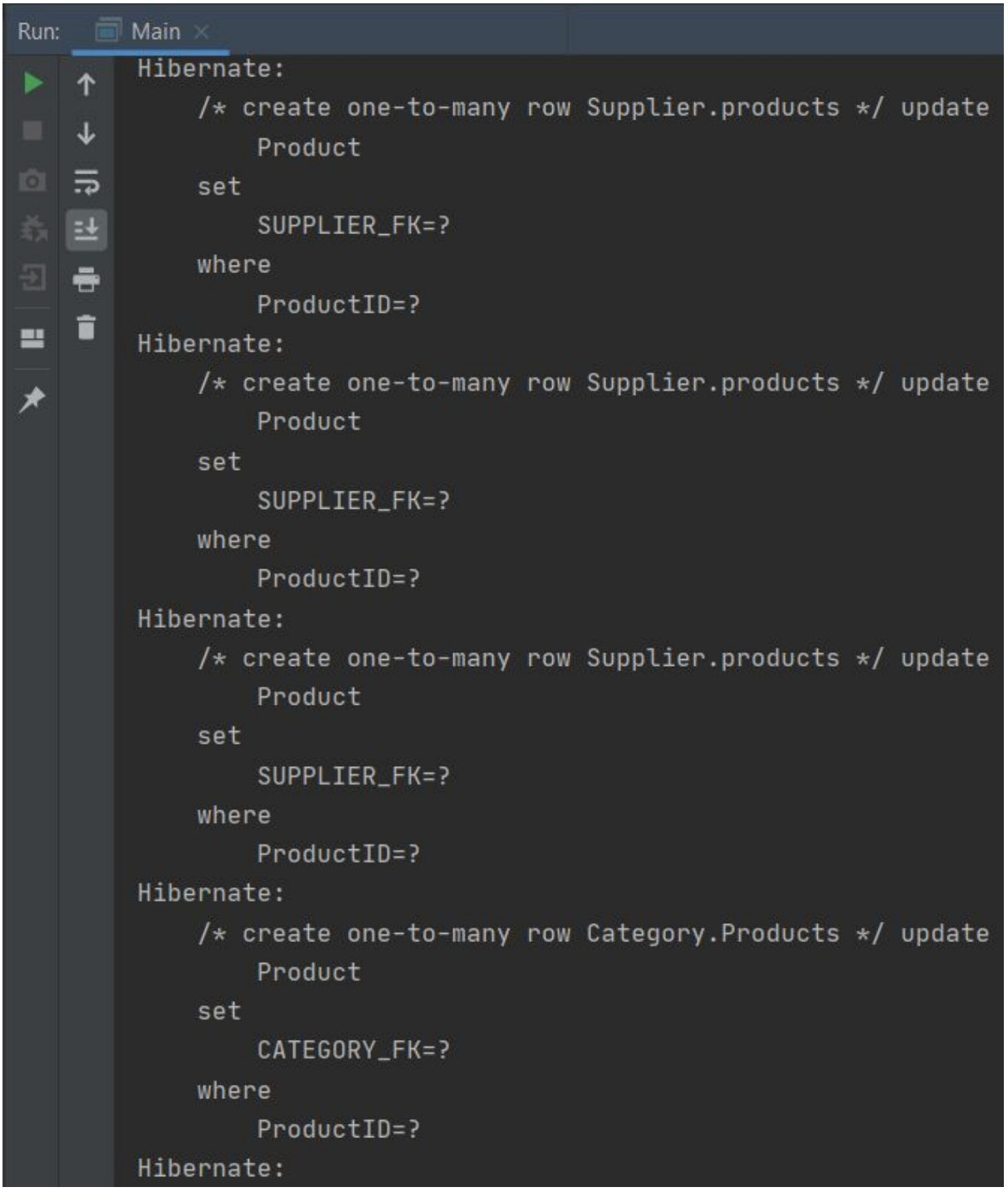
Uruchomienie programu

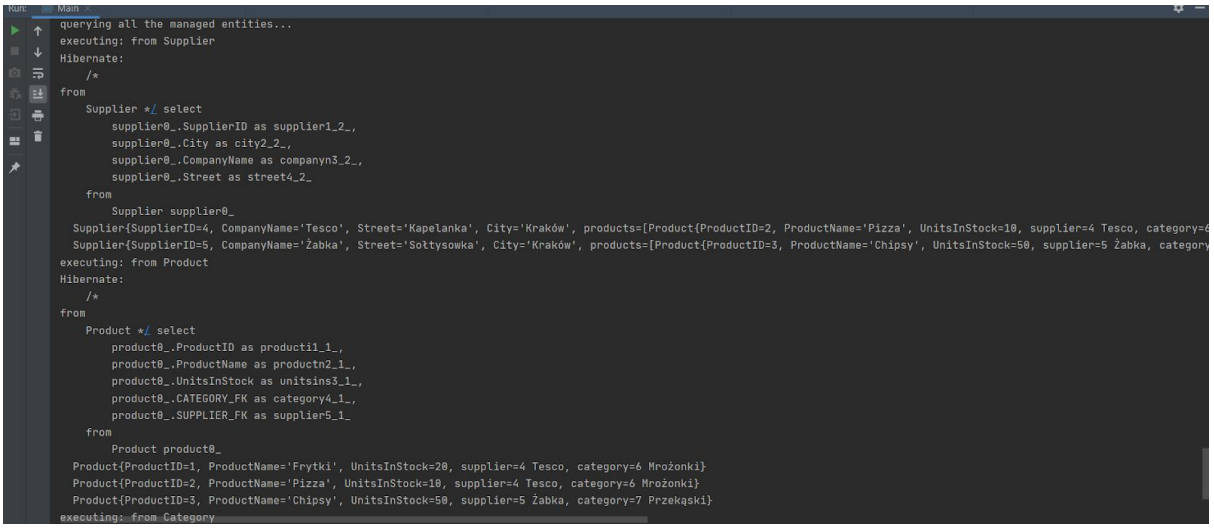


```
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, UnitsInStock, CATEGORY_FK, SUPPLIER_FK, ProductID)
        values
            (?, ?, ?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (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
```

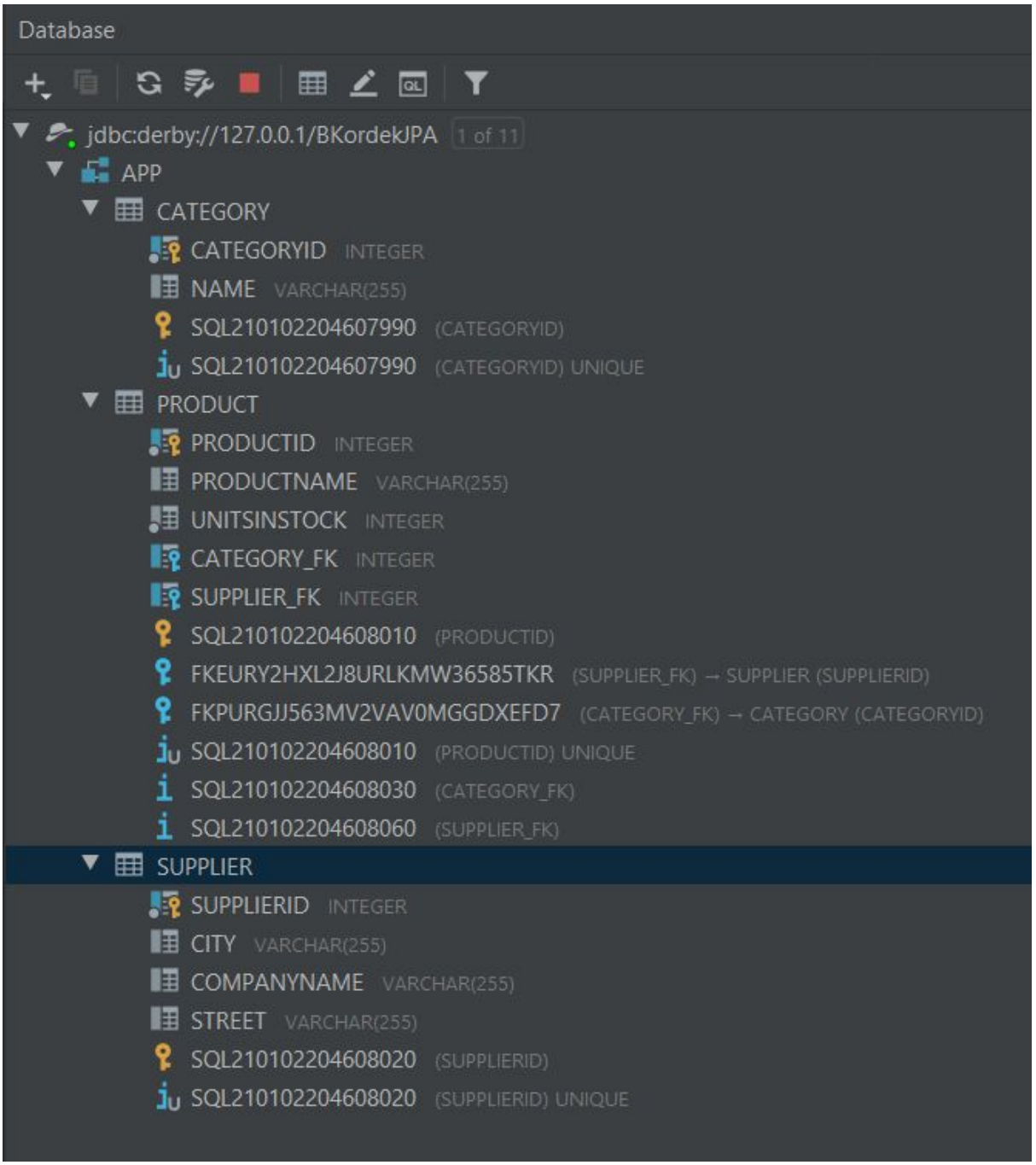

un:
Main
(?, ?, ?, ?, ?)
Hibernate:
/* insert Supplier
*/ insert
into
Supplier
(City, CompanyName, Street, SupplierID)
values
(?, ?, ?, ?)
Hibernate:
/* insert Supplier
*/ insert
into
Supplier
(City, CompanyName, Street, SupplierID)
values
(?, ?, ?, ?)
Hibernate:
/* insert Category
*/ insert
into
Category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

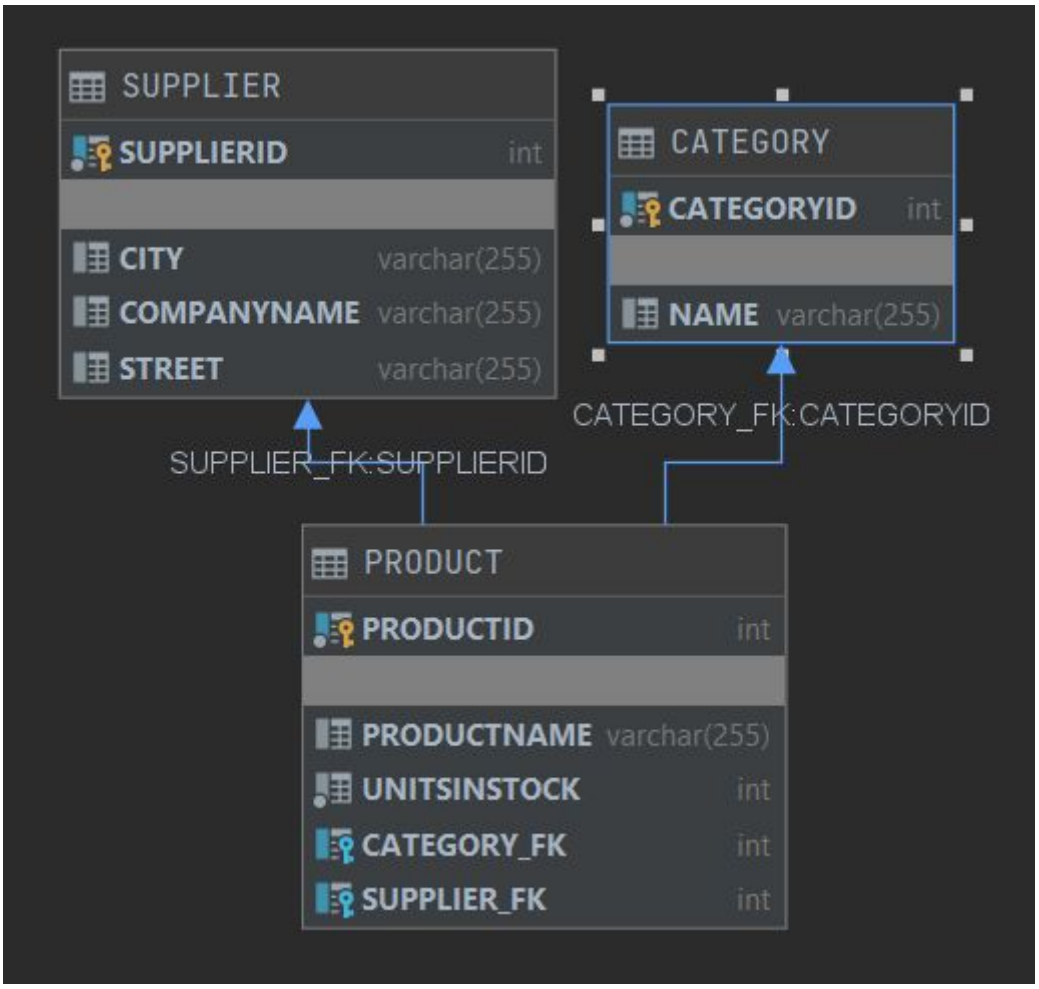

Run:
Main
Hibernate:
/* update
Product */ update
Product
set
ProductName=?,
UnitsInStock=?,
CATEGORY_FK=?,
SUPPLIER_FK=?
where
ProductID=?
Hibernate:
/* update
Product */ update
Product
set
ProductName=?,
UnitsInStock=?,
CATEGORY_FK=?,
SUPPLIER_FK=?
where
ProductID=?
Hibernate:
/* update
Product */ update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


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:

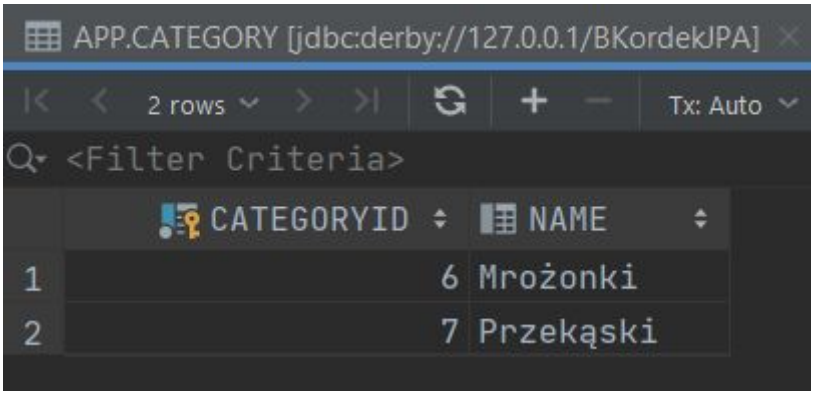

Run:
Main
querying all the managed entities...
executing: from Supplier
Hibernate:
/*
from
Supplier */ select
supplier0_.SupplierID as supplier1_2_,
supplier0_.City as city2_2_,
supplier0_.CompanyName as companyn3_2_,
supplier0_.Street as street4_2_
from
Supplier supplier0_
Supplier{SupplierID=4, CompanyName='Tesco', Street='Kapelanka', City='Kraków', products=[Product{ProductID=2, ProductName='Pizza', UnitsInStock=10, supplier=4 Tesco, category=6
Supplier{SupplierID=5, CompanyName='Żabka', Street='Sołtysowka', City='Kraków', products=[Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, supplier=5 Żabka, category
executing: from Product
Hibernate:
/*
from
Product */ select
product0_.ProductID as producti1_1_,
product0_.ProductName as productn2_1_,
product0_.UnitsInStock as unitsins3_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

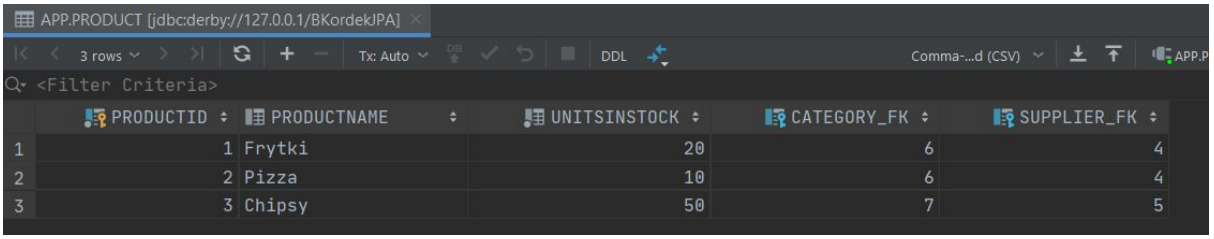
Schemat bazy danych





Zawartość tabel





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

2 rows | Tx: Auto | DDL

<Filter Criteria>

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

Z powyższego widać, że do produktów zostały przypisane kategorie.

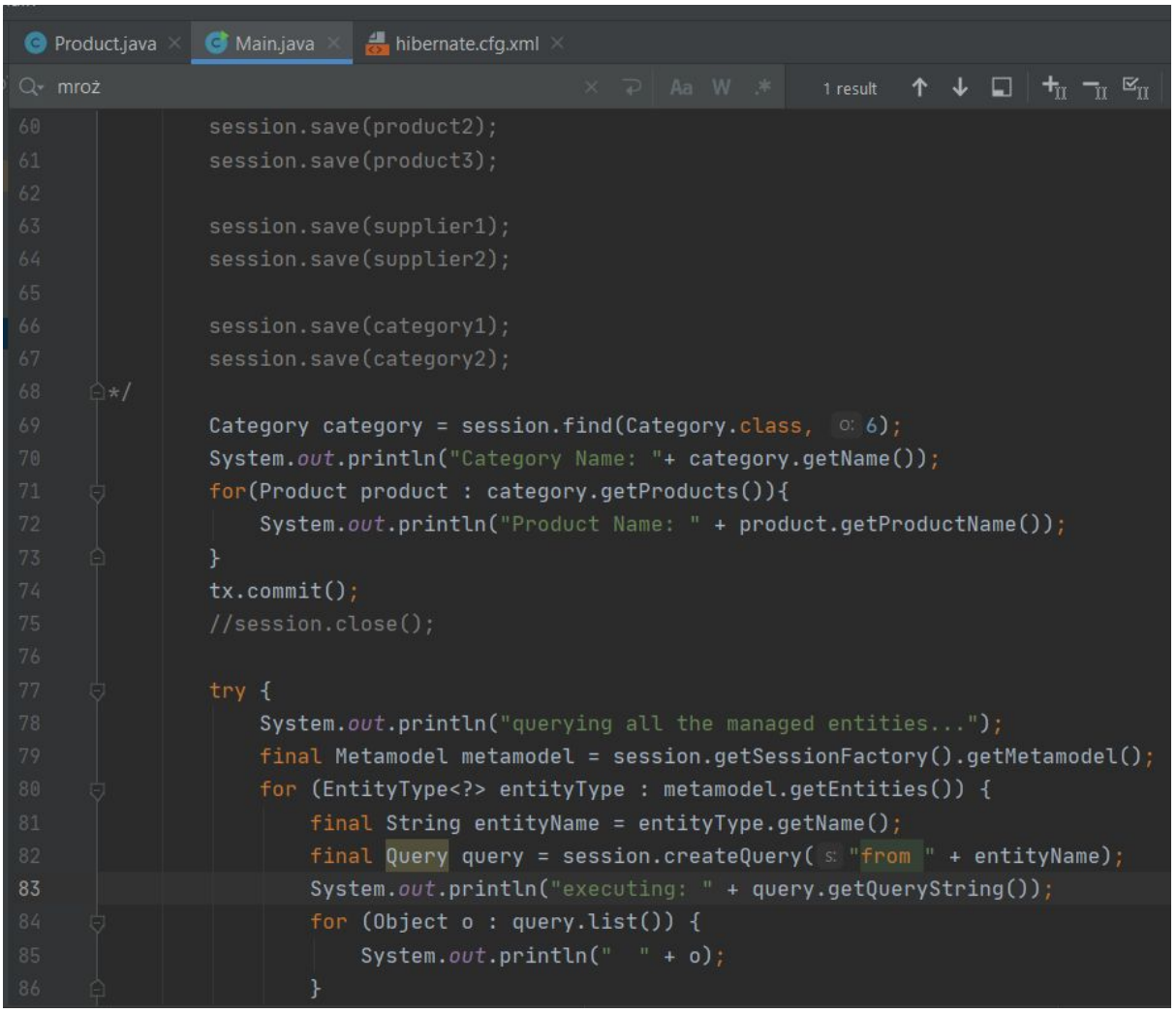
Wydobycie produktów z wybranej kategorii oraz kategorię do której należy wybrany produkt

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

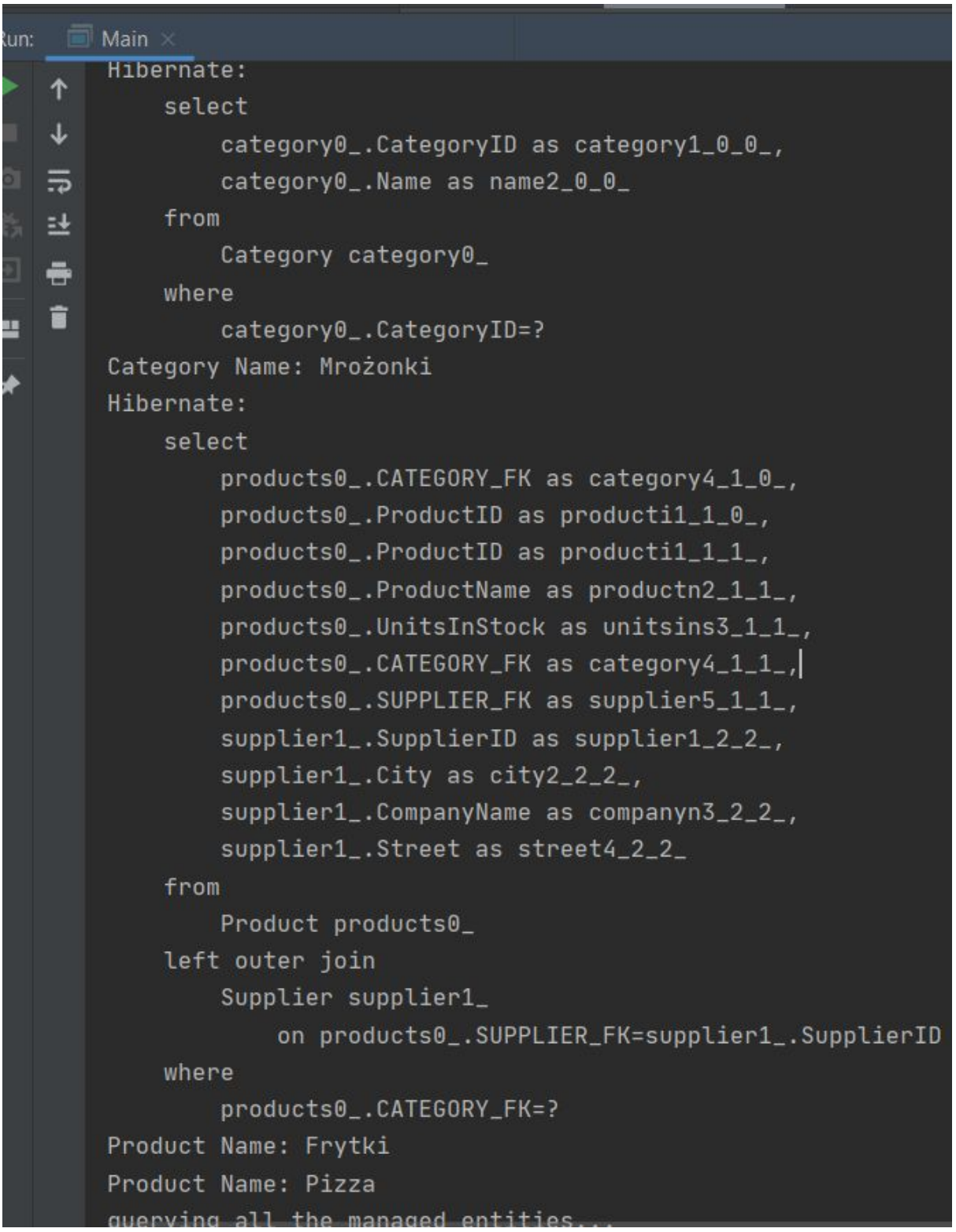
APP.SUPPLIER [jdbc:derby://127.0.0.1/BKordekJPA] | Main.java | hibernate.cfg.xml

```
<?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">update</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
        <mapping class="Category"></mapping>
    </session-factory>
</hibernate-configuration>
```

Modyfikacja klasy *Main* - wyszukanie produktów danej kategorii



Uruchomienie programu



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

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

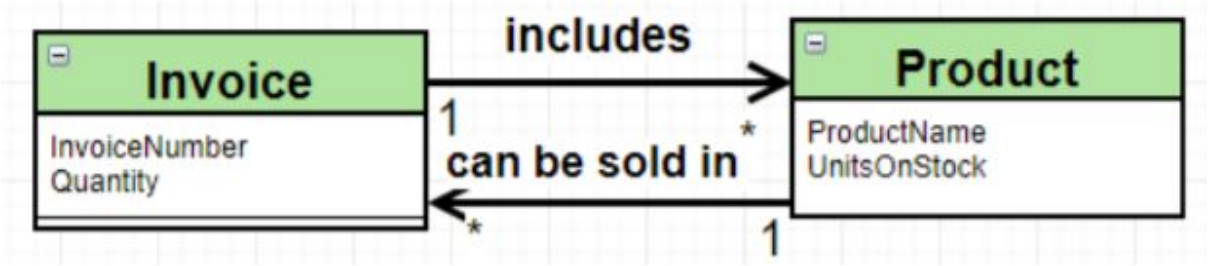
```
74          }
75
76          */
77          Product product = session.find(Product.class, o: 1);
78          System.out.println("Product Category ID: "+ product.getCategory().getCategoryID()+
79                  ", Product Category Name: " + product.getCategory().getName());
80
81          tx.commit();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

Uruchomienie programu

```
Hibernate:
    select
        product0_.ProductID as producti1_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:

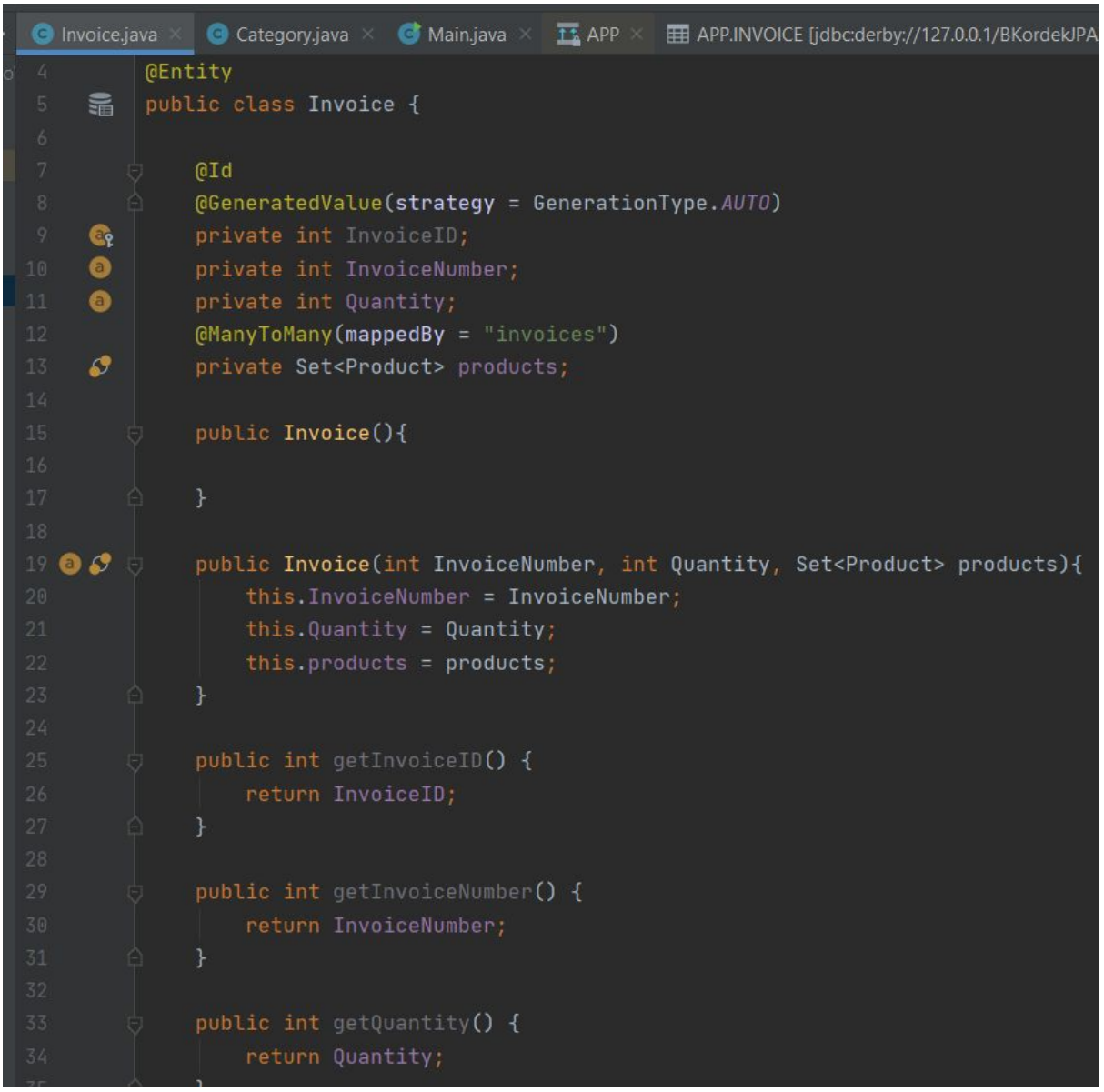


Klasa *Category*

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

```java
@Entity
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CategoryID;
    private String Name;
    @OneToMany
    @JoinColumn(name = "CATEGORY_FK")
    private List<Product> Products;

    public Category(){

    }

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

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

    public int getCategoryID() {
        return CategoryID;
    }

    public String getName() {
        return Name;
    }
```

Klasa *Invoice*



```
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]

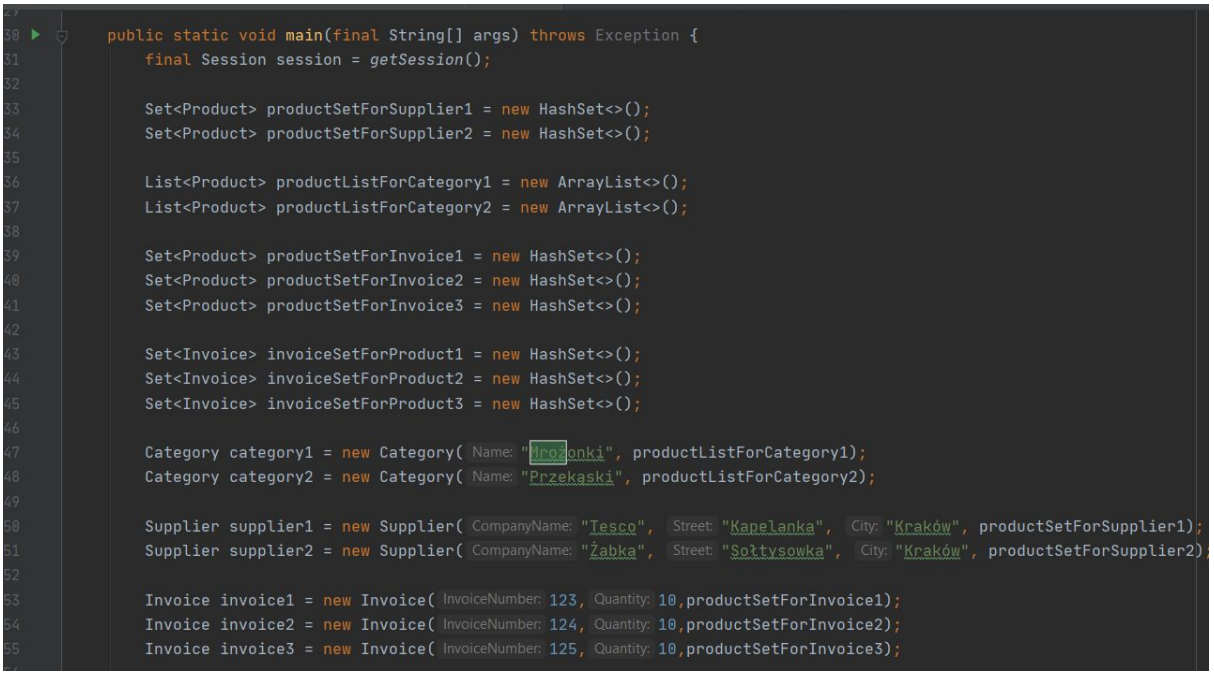
```
4   @Entity
5   public class Product {
6
7       @Id
8       @GeneratedValue(strategy = GenerationType.AUTO)
9       private int ProductID;
10      private String ProductName;
11      private int UnitsInStock;
12      @ManyToOne
13      @JoinColumn(name="SUPPLIER_FK")
14      private Supplier supplier;
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;
```

```java

    public String getProductName() {
        return ProductName;
    }

    public Category getCategory() { return category; }

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

    @Override
    public String toString() {
        return "Product{" +
                "ProductID=" + ProductID +
                ", ProductName='" + ProductName + '\'' +
                ", UnitsInStock=" + UnitsInStock +
                ", supplier=" + supplier.getSupplierID() + ' ' + supplier.getCompanyName() +
                ", category=" + category.getCategoryID() + ' ' + category.getName() +
                '}';
    }
}
```

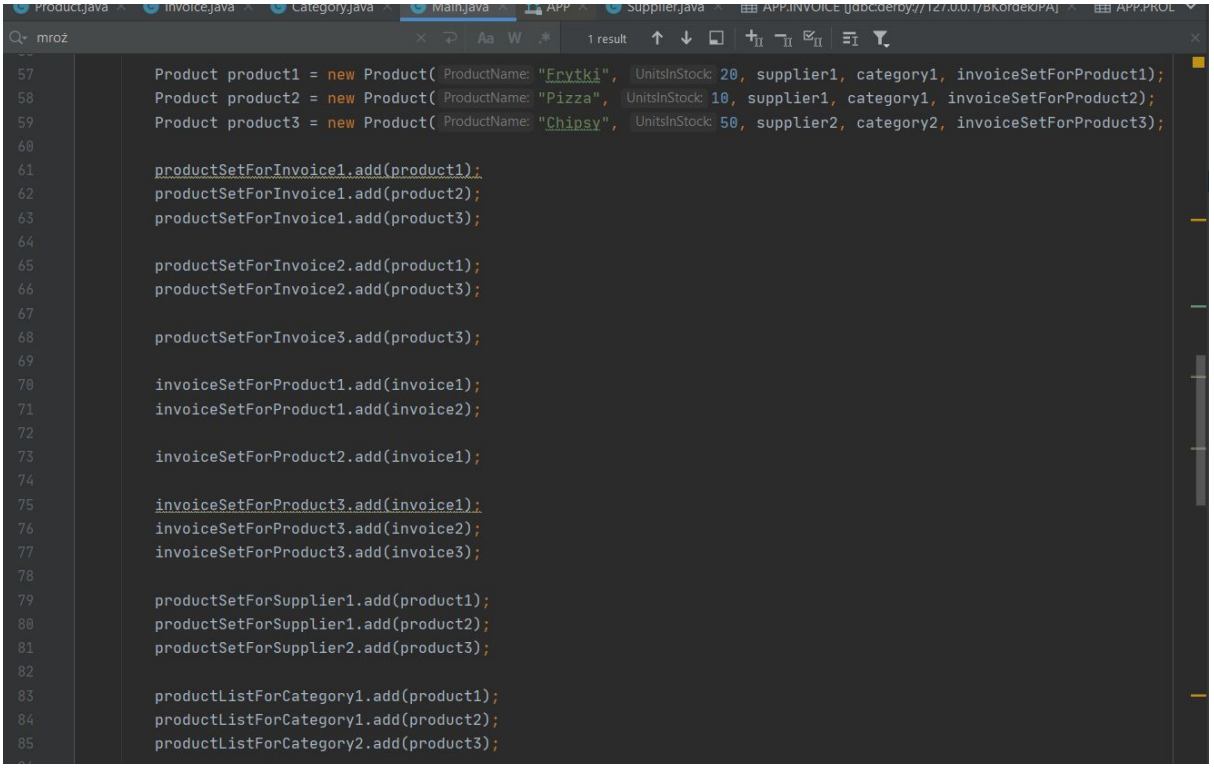
Klasa *Supplier*

```java
@Entity
public class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    @JoinColumn(name="SUPPLIER_FK")
    private Set<Product> products;

    public Supplier(){

    }

    public Supplier(String CompanyName, String Street, String City, Set<Product> products){
        this.CompanyName = CompanyName;
        this.Street = Street;
        this.City = City;
        this.products = products;
    }

    public int getSupplierID() {
        return SupplierID;
    }

    public String getCompanyName() {
        return CompanyName;
    }
}
```

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



```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();

    Set<Product> productSetForSupplier1 = new HashSet<>();
    Set<Product> productSetForSupplier2 = new HashSet<>();

    List<Product> productListForCategory1 = new ArrayList<>();
    List<Product> productListForCategory2 = new ArrayList<>();

    Set<Product> productSetForInvoice1 = new HashSet<>();
    Set<Product> productSetForInvoice2 = new HashSet<>();
    Set<Product> productSetForInvoice3 = new HashSet<>();

    Set<Invoice> invoiceSetForProduct1 = new HashSet<>();
    Set<Invoice> invoiceSetForProduct2 = new HashSet<>();
    Set<Invoice> invoiceSetForProduct3 = new HashSet<>();

    Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
    Category category2 = new Category( Name: "Przekąski", productListForCategory2);

    Supplier supplier1 = new Supplier( CompanyName: "Tesco",  Street: "Kapelanka",  City: "Kraków", productSetForSupplier1);
    Supplier supplier2 = new Supplier( CompanyName: "Żabka",  Street: "Sołtysowka",  City: "Kraków", productSetForSupplier2);

    Invoice invoice1 = new Invoice( InvoiceNumber: 123, Quantity: 10,productSetForInvoice1);
    Invoice invoice2 = new Invoice( InvoiceNumber: 124, Quantity: 10,productSetForInvoice2);
    Invoice invoice3 = new Invoice( InvoiceNumber: 125, Quantity: 10,productSetForInvoice3);
```



```
mroż                                                    1 result

    Product product1 = new Product( ProductName: "Frytki",  UnitsInStock: 20, supplier1, category1, invoiceSetForProduct1);
    Product product2 = new Product( ProductName: "Pizza",  UnitsInStock: 10, supplier1, category1, invoiceSetForProduct2);
    Product product3 = new Product( ProductName: "Chipsy",  UnitsInStock: 50, supplier2, category2, invoiceSetForProduct3);

    productSetForInvoice1.add(product1);
    productSetForInvoice1.add(product2);
    productSetForInvoice1.add(product3);

    productSetForInvoice2.add(product1);
    productSetForInvoice2.add(product3);

    productSetForInvoice3.add(product3);

    invoiceSetForProduct1.add(invoice1);
    invoiceSetForProduct1.add(invoice2);

    invoiceSetForProduct2.add(invoice1);

    invoiceSetForProduct3.add(invoice1);
    invoiceSetForProduct3.add(invoice2);
    invoiceSetForProduct3.add(invoice3);

    productSetForSupplier1.add(product1);
    productSetForSupplier1.add(product2);
    productSetForSupplier2.add(product3);

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

```
        Transaction tx = session.beginTransaction();

        session.save(product1);
        session.save(product2);
        session.save(product3);

        session.save(supplier1);
        session.save(supplier2);

        session.save(category1);
        session.save(category2);

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

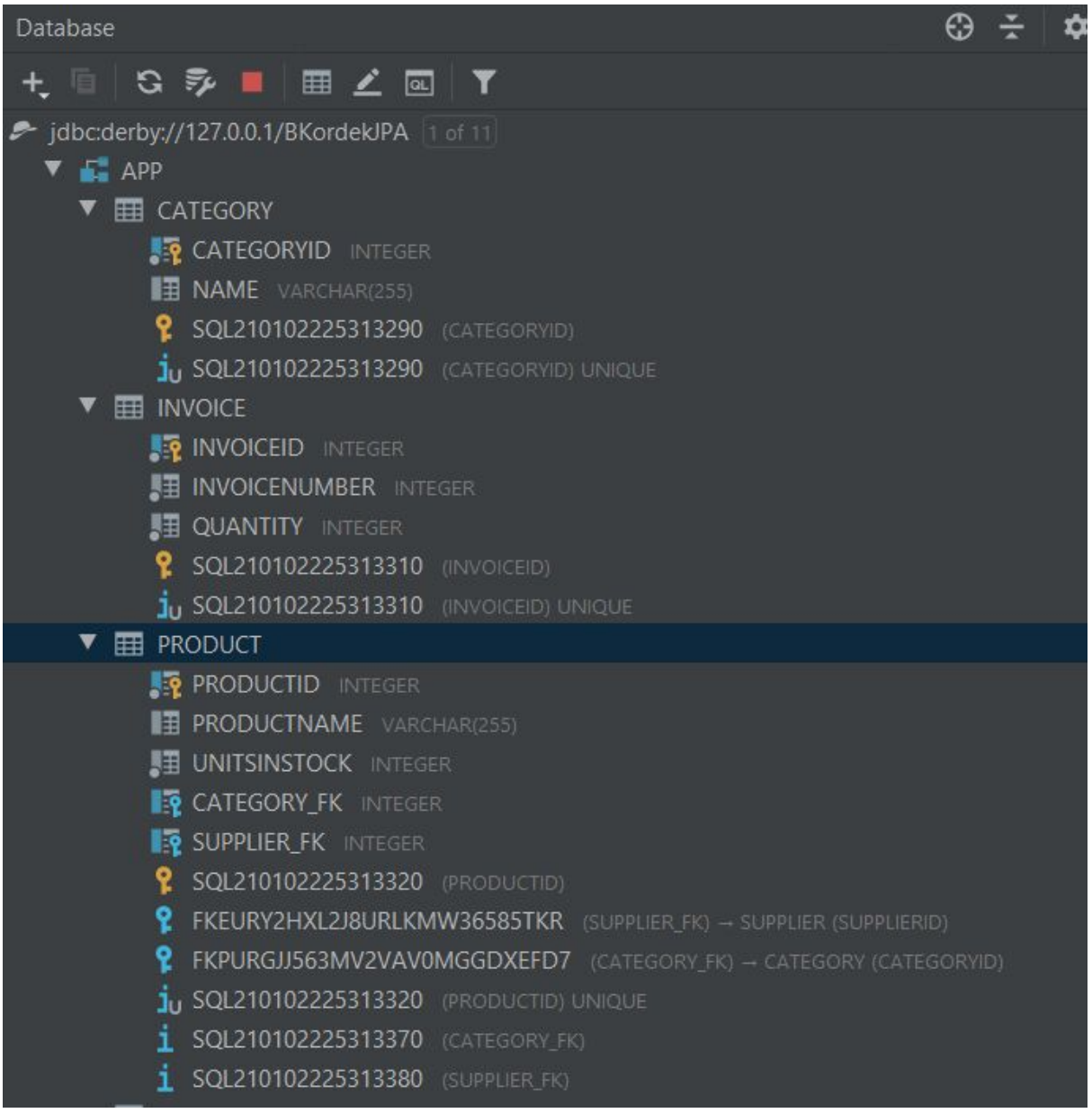
Modyfikacja pliku konfiguracyjnego - dodanie mapowania klasy *Invoice*

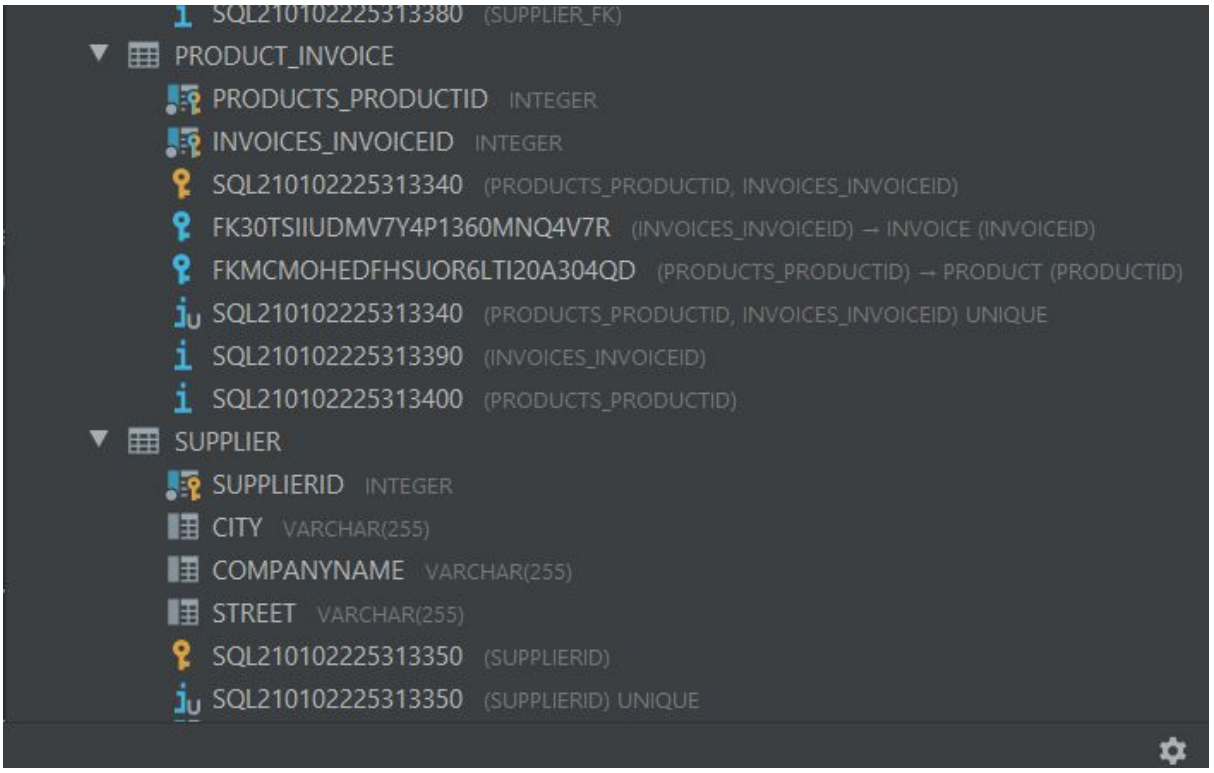
```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</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>
        <mapping class="Category"></mapping>
        <mapping class="Invoice"></mapping>
    </session-factory>
</hibernate-configuration>
```

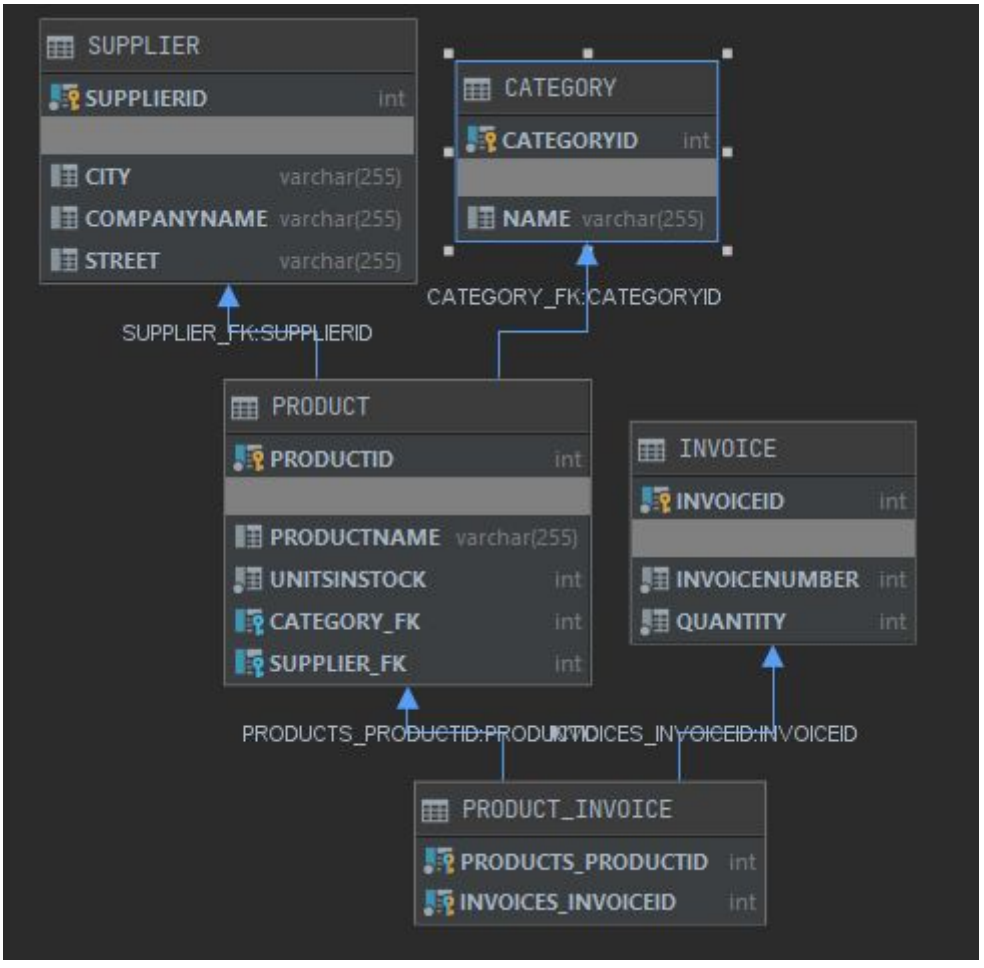
Uruchomienie programu

```
Run:  Main
Hibernate:
    /*
from
    Invoice */ select
        invoice0_.InvoiceID as invoicei1_1_,
        invoice0_.InvoiceNumber as invoicen2_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{Prod
  Invoice{InvoiceID=9, InvoiceNumber=124, Quantity=10, products=[Product{ProductID=1, ProductName='Frytki', UnitsInStock=20, supplier=4 Tesco, category=6 Mrożonki}, Product{Prod
  Invoice{InvoiceID=10, InvoiceNumber=125, Quantity=10, products=[Product{ProductID=3, ProductName='Chipsy', UnitsInStock=50, supplier=5 Żabka, category=7 Przekąski}]}
executing: from Product
Hibernate:
    /*
from
    Product */ select
        product0_.ProductID as productid1_2_,
        product0_.ProductName as productn2_2_,
        product0_.UnitsInStock as unitsins3_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 Żabka, category=7 Przekąski}

Process finished with exit code 0
```

Schemat bazy danych




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)
SQL210102225313340 (PRODUCTS_PRODUCTID, INVOICES_INVOICEID) UNIQUE
SQL210102225313390 (INVOICES_INVOICEID)
SQL210102225313400 (PRODUCTS_PRODUCTID)
SUPPLIER
SUPPLIERID INTEGER
CITY VARCHAR(255)
COMPANYNAME VARCHAR(255)
STREET VARCHAR(255)
SQL210102225313350 (SUPPLIERID)
SQL210102225313350 (SUPPLIERID) UNIQUE


SUPPLIER
SUPPLIERID int
CITY varchar(255)
COMPANYNAME varchar(255)
STREET varchar(255)
CATEGORY
CATEGORYID int
NAME varchar(255)
CATEGORY_FK:CATEGORYID
SUPPLIER_FK:SUPPLIERID
PRODUCT
PRODUCTID int
PRODUCTNAME varchar(255)
UNITSINSTOCK int
CATEGORY_FK int
SUPPLIER_FK int
INVOICE
INVOICEID int
INVOICENUMBER int
QUANTITY int
PRODUCTS_PRODUCTID:PRODUCTID
INVOICES_INVOICEID:INVOICEID
PRODUCT_INVOICE
PRODUCTS_PRODUCTID int
INVOICES_INVOICEID int

Zawartość tabel

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

2 rows | Tx: Auto

<Filter Criteria>

	CATEGORYID	NAME
1	6	Mrożonki
2	7	Przekąski

APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA] | APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]

3 rows | Tx: Auto | DDL

<Filter Criteria>

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

APP.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA] | APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA] | APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA]

3 rows | Tx: Auto | DDL | Comma-...d (CSV)

<Filter Criteria>

	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]

6 rows | Tx: Auto | DDL

<Filter Criteria>

	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]

2 rows | Tx: Auto | DDL

<Filter Criteria>

	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, o: 8);
104         for(Product product : invoice.getProducts())
105             System.out.println("Product Name: " + product.getProductName());
106
107         /*
```

Uruchomienie programu

Run: Main

```
        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         /*
```

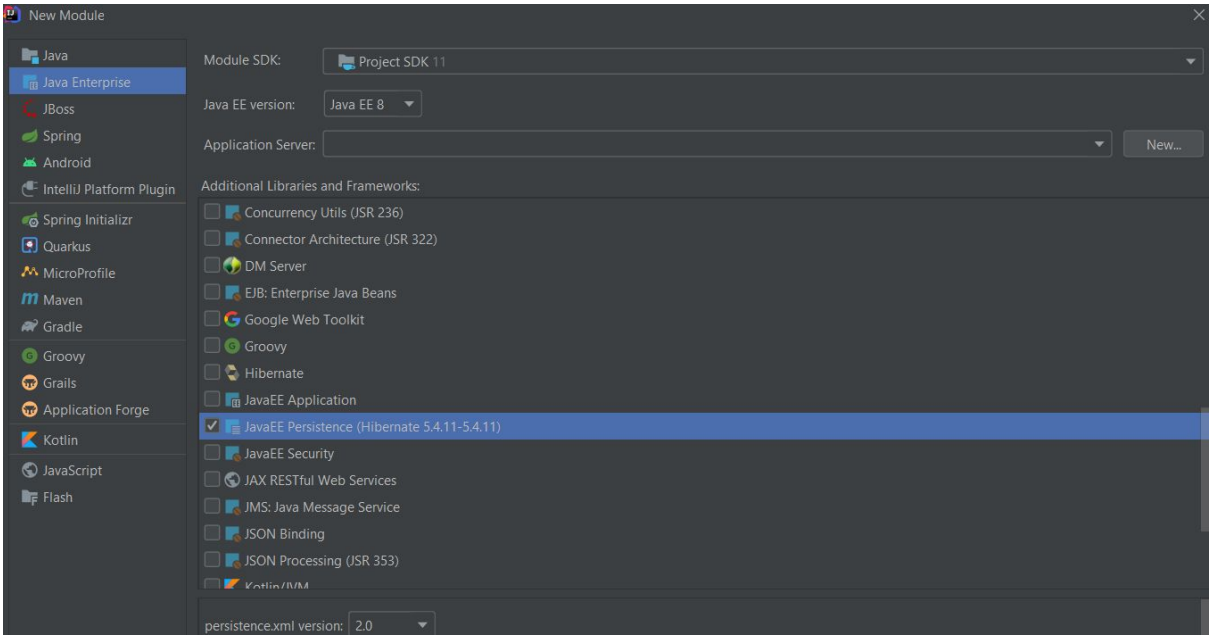
Uruchomienie programu

```
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'}
```

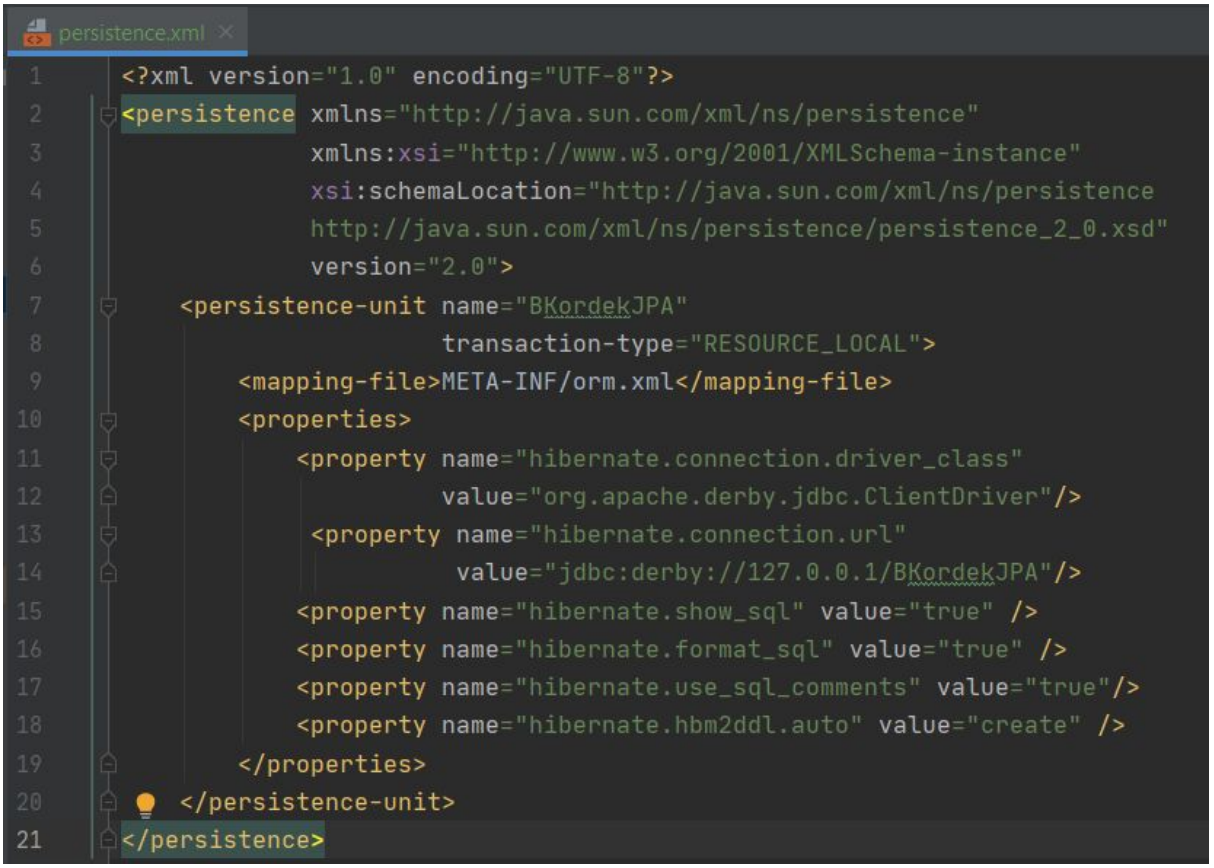
Z powyższego widać, że produkt o ID równym 1 jest przypisany do kategorii od ID równym 6 i nazwie Mrożonki.

X. JPA

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



Modyfikacja pliku konfiguracyjnego



Utworzenie pliku konfiguracyjnego z mappingiem klas.

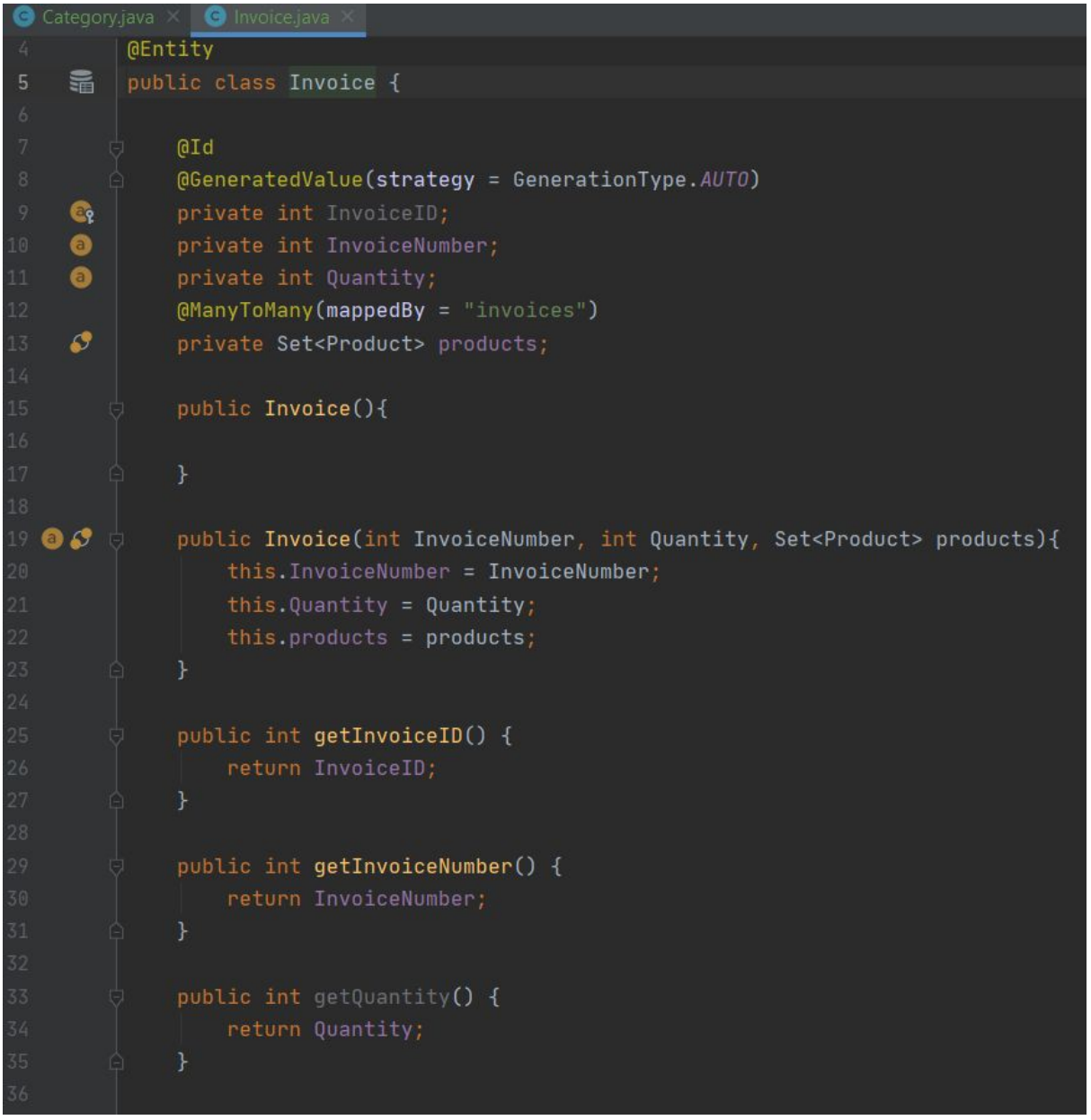
orm.xml

```xml
<?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>
```

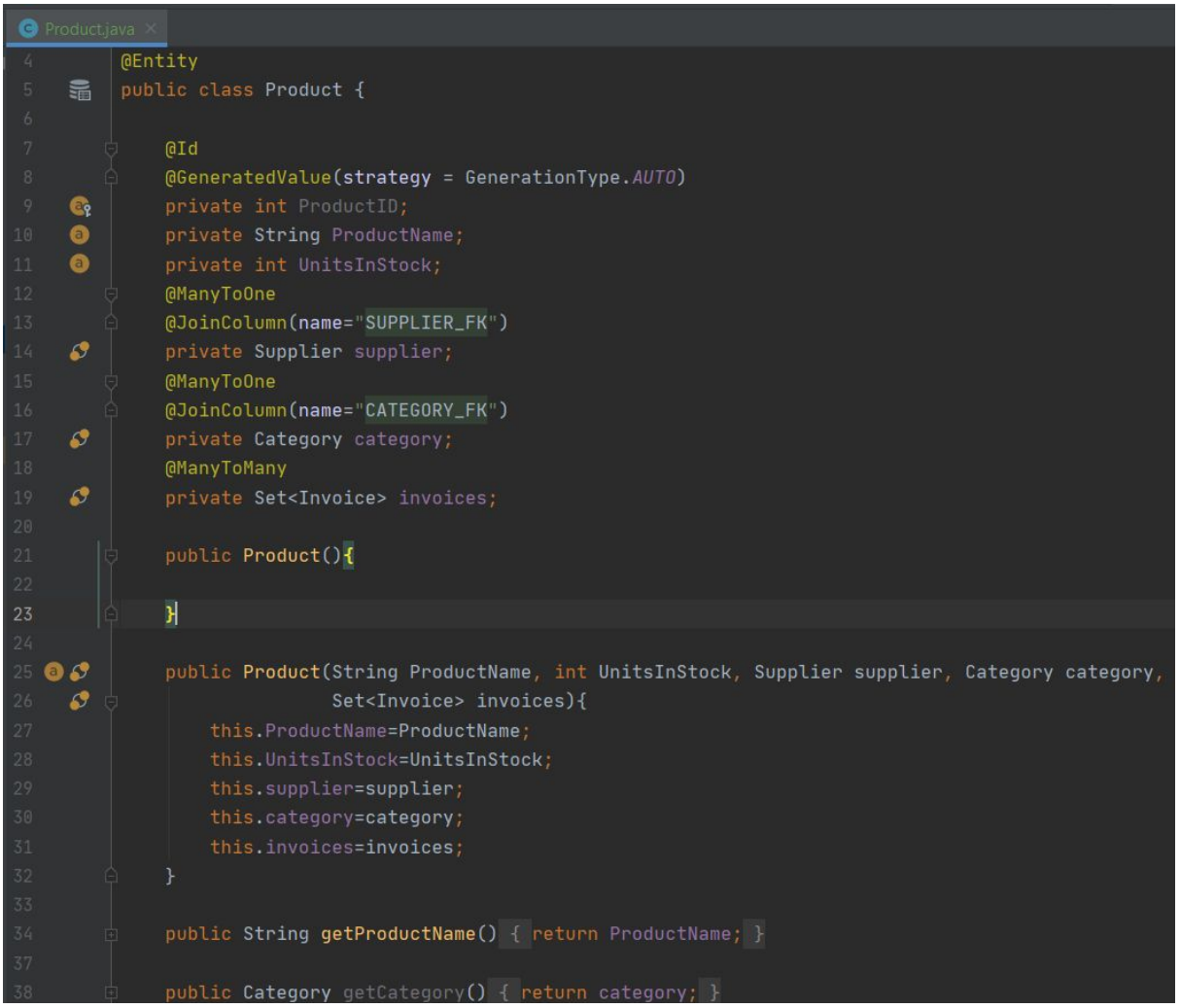
Klasy *Category*, *Invoice*, *Product*, *Supplier* są takie same jak poprzednio.

Klasa *Category*

Klasa *Invoice*



Klasa *Product*



Klasa *Supplier*

Product.java | Supplier.java

```java
@Entity
public class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    @JoinColumn(name="SUPPLIER_FK")
    private Set<Product> products;

    public Supplier(){

    }

    public Supplier(String CompanyName, String Street, String City, Set<Product> products){
        this.CompanyName = CompanyName;
        this.Street = Street;
        this.City = City;
        this.products = products;
    }

    public int getSupplierID() {
        return SupplierID;
    }

    public String getCompanyName() {
        return CompanyName;
    }

    @Override
```

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

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

Main.java

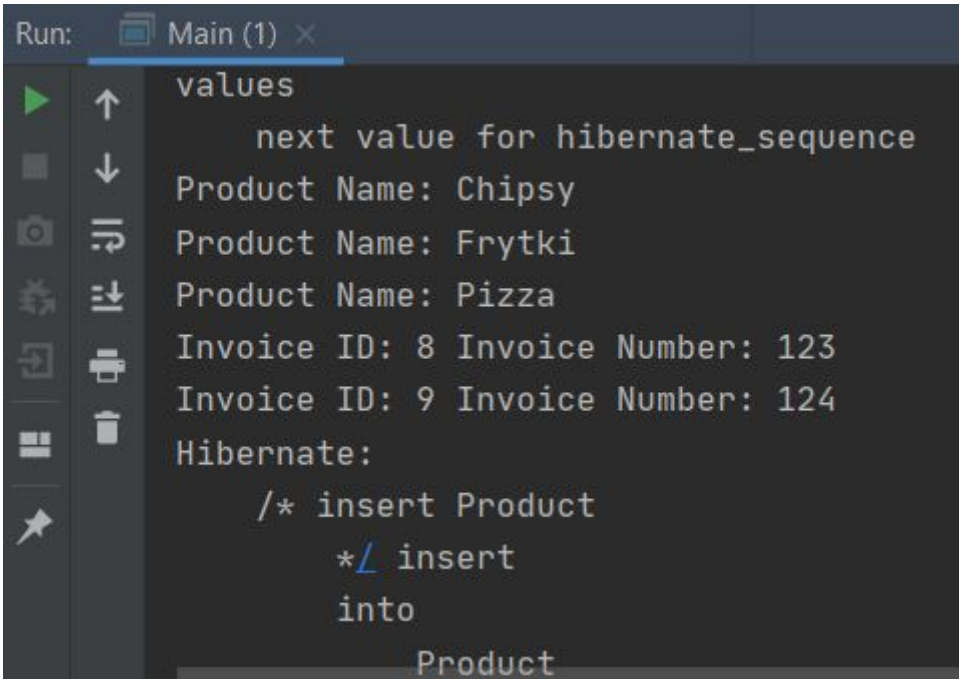
```java
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "BKordekJPA");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();

        Set<Product> productSetForSupplier1 = new HashSet<>();
        Set<Product> productSetForSupplier2 = new HashSet<>();

        List<Product> productListForCategory1 = new ArrayList<>();
        List<Product> productListForCategory2 = new ArrayList<>();

        Set<Product> productSetForInvoice1 = new HashSet<>();
        Set<Product> productSetForInvoice2 = new HashSet<>();
        Set<Product> productSetForInvoice3 = new HashSet<>();

        Set<Invoice> invoiceSetForProduct1 = new HashSet<>();
        Set<Invoice> invoiceSetForProduct2 = new HashSet<>();
        Set<Invoice> invoiceSetForProduct3 = new HashSet<>();

        Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
        Category category2 = new Category( Name: "Przekąski", productListForCategory2);
```

Main.java

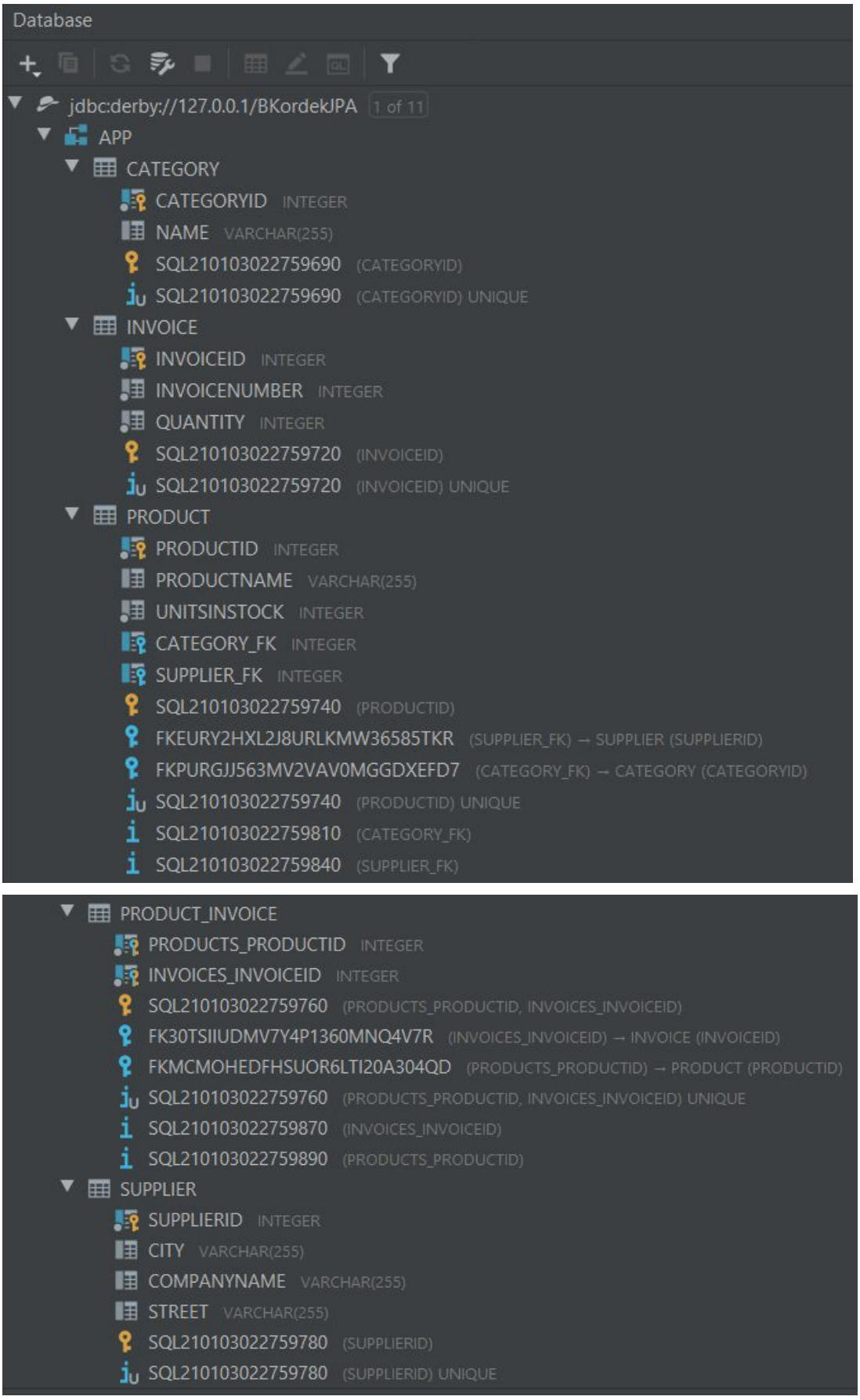
```java
    Supplier supplier1 = new Supplier( CompanyName: "Tesco",  Street: "Kapelanka",  City: "Kraków", productSetFor
    Supplier supplier2 = new Supplier( CompanyName: "Żabka",  Street: "Sołtysowka",  City: "Kraków", productSetFo

    Invoice invoice1 = new Invoice( InvoiceNumber: 123, Quantity: 10,productSetForInvoice1);
    Invoice invoice2 = new Invoice( InvoiceNumber: 124, Quantity: 10,productSetForInvoice2);
    Invoice invoice3 = new Invoice( InvoiceNumber: 125, Quantity: 10,productSetForInvoice3);

    Product product1 = new Product( ProductName: "Frytki",  UnitsInStock: 20, supplier1, category1, invoiceSetForP
    Product product2 = new Product( ProductName: "Pizza",  UnitsInStock: 10, supplier1, category1, invoiceSetForPr
    Product product3 = new Product( ProductName: "Chipsy",  UnitsInStock: 50, supplier2, category2, invoiceSetForP

    productSetForInvoice1.add(product1);
    productSetForInvoice1.add(product2);
    productSetForInvoice1.add(product3);

    productSetForInvoice2.add(product1);
    productSetForInvoice2.add(product3);

    productSetForInvoice3.add(product3);

    invoiceSetForProduct1.add(invoice1);
    invoiceSetForProduct1.add(invoice2);

    invoiceSetForProduct2.add(invoice1);

    invoiceSetForProduct3.add(invoice1);
    invoiceSetForProduct3.add(invoice2);
    invoiceSetForProduct3.add(invoice3);

    productSetForSupplier1.add(product1);
    productSetForSupplier1.add(product2);
    productSetForSupplier2.add(product3);
```

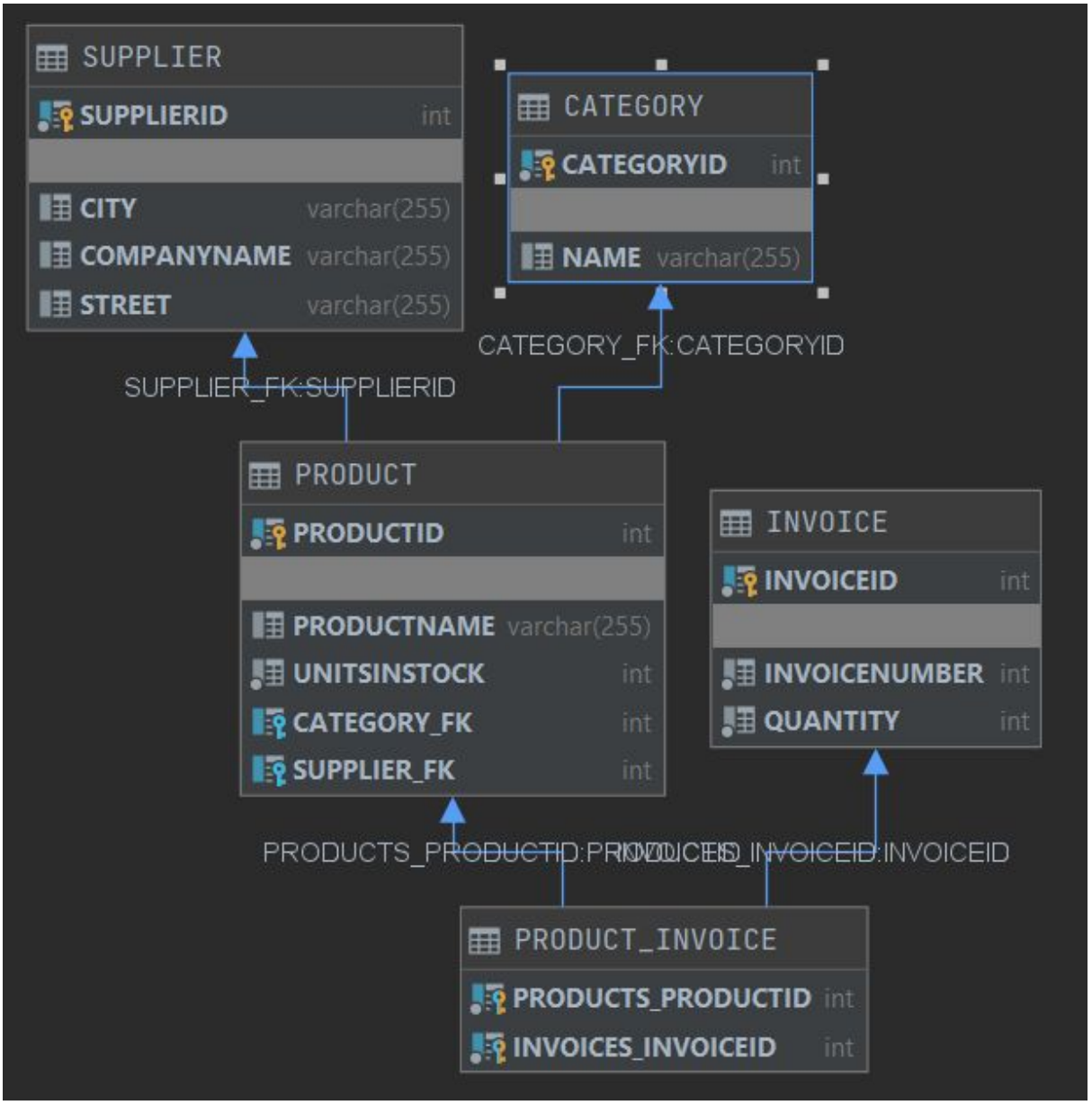
```java
productListForCategory1.add(product1);
productListForCategory1.add(product2);
productListForCategory2.add(product3);

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

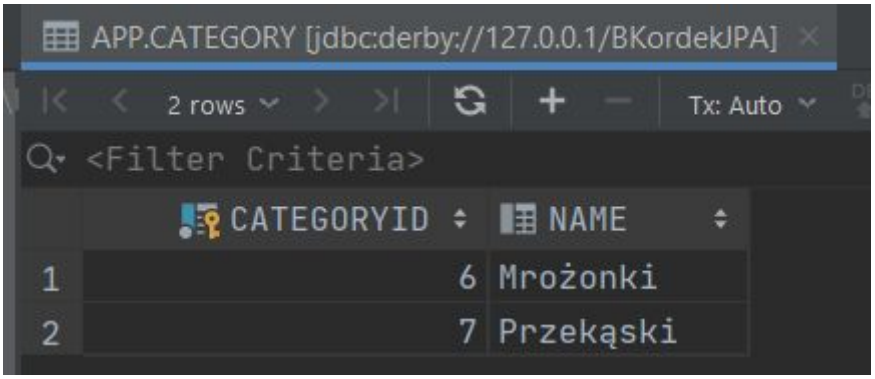


Schemat bazy danych





Zawartość tabel



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

3 rows | Tx: Auto | DDL

<Filter Criteria>

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

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

3 rows | Tx: Auto | DDL | Comma-...d (CSV) | APP.PRODUCT

<Filter Criteria>

	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]

6 rows | Tx: Auto | Comm

<Filter Criteria>

	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]

2 rows | Tx: Auto | DDL | Comma-...d

<Filter Criteria>

	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*

Invoice.java

```java
@Entity
public class Invoice {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int InvoiceID;
    private int InvoiceNumber;
    private int Quantity;
    @ManyToMany(mappedBy = "invoices", cascade = CascadeType.PERSIST)
    private Set<Product> products;

    public Invoice(){

    }

    public Invoice(int InvoiceNumber, int Quantity, Set<Product> products){
        this.InvoiceNumber = InvoiceNumber;
        this.Quantity = Quantity;
        this.products = products;
    }

    public int getInvoiceID() {
        return InvoiceID;
    }

    public int getInvoiceNumber() {
        return InvoiceNumber;
    }

    public int getQuantity() {
        return Quantity;
    }
```

Klasa *Product*

Product.java

```java
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsInStock;
    @ManyToOne
    @JoinColumn(name="SUPPLIER_FK")
    private Supplier supplier;
    @ManyToOne
    @JoinColumn(name="CATEGORY_FK")
    private Category category;
    @ManyToMany(cascade = CascadeType.PERSIST)
    private Set<Invoice> invoices;

    public Product(){

    }

    public Product(String ProductName, int UnitsInStock, Supplier supplier, Category category,
                   Set<Invoice> invoices){
        this.ProductName=ProductName;
        this.UnitsInStock=UnitsInStock;
        this.supplier=supplier;
        this.category=category;
        this.invoices=invoices;
    }

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

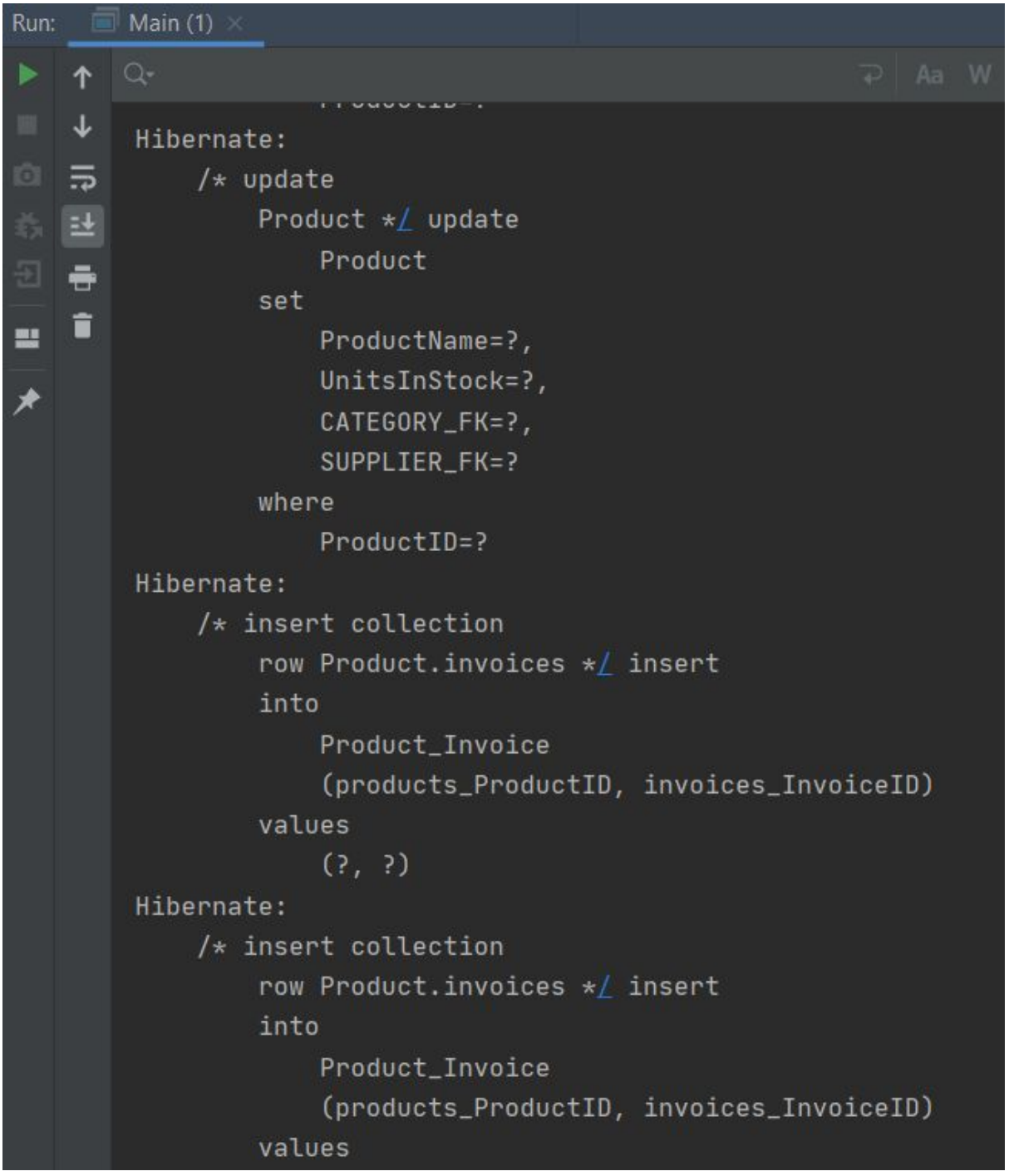
Klasa sterująca *Main*

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

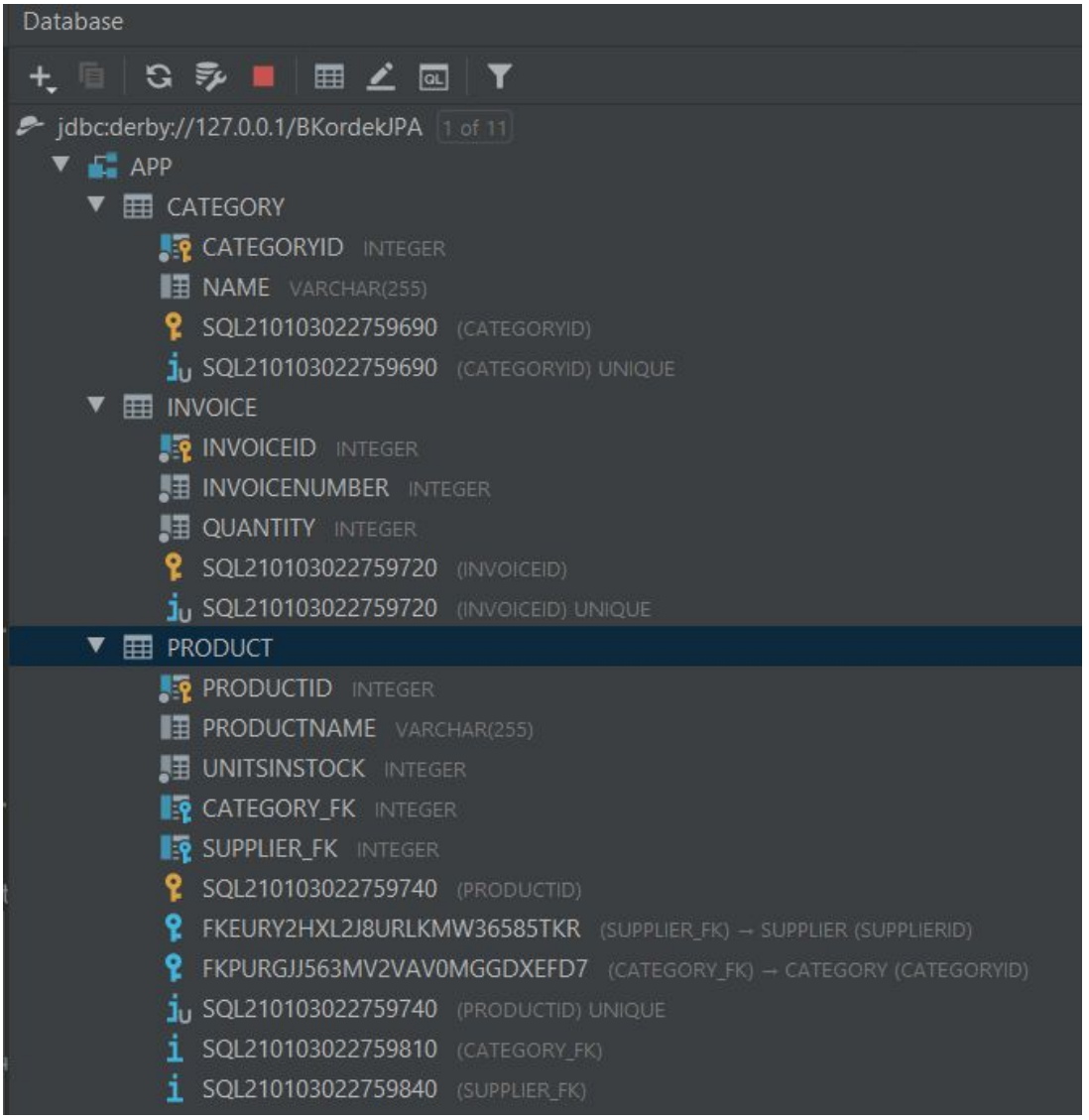
Main.java

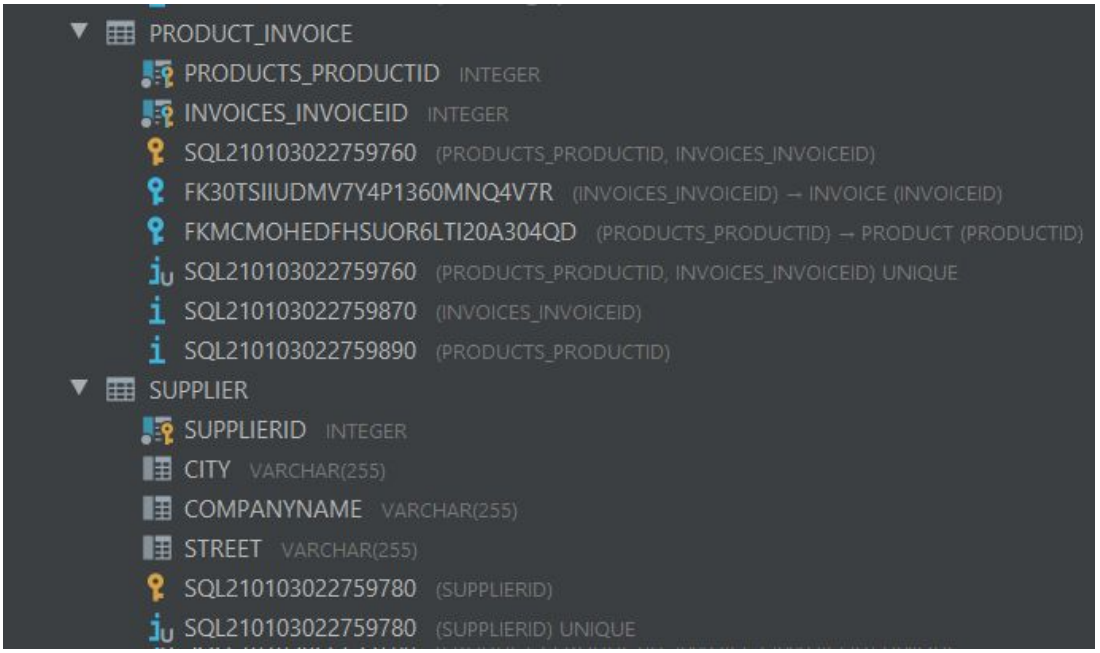
```
64          productSetForSupplier1.add(product2);
65          productSetForSupplier2.add(product3);
66
67          productListForCategory1.add(product1);
68          productListForCategory1.add(product2);
69          productListForCategory2.add(product3);
70
71          em.persist(product1);
72          //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();
```

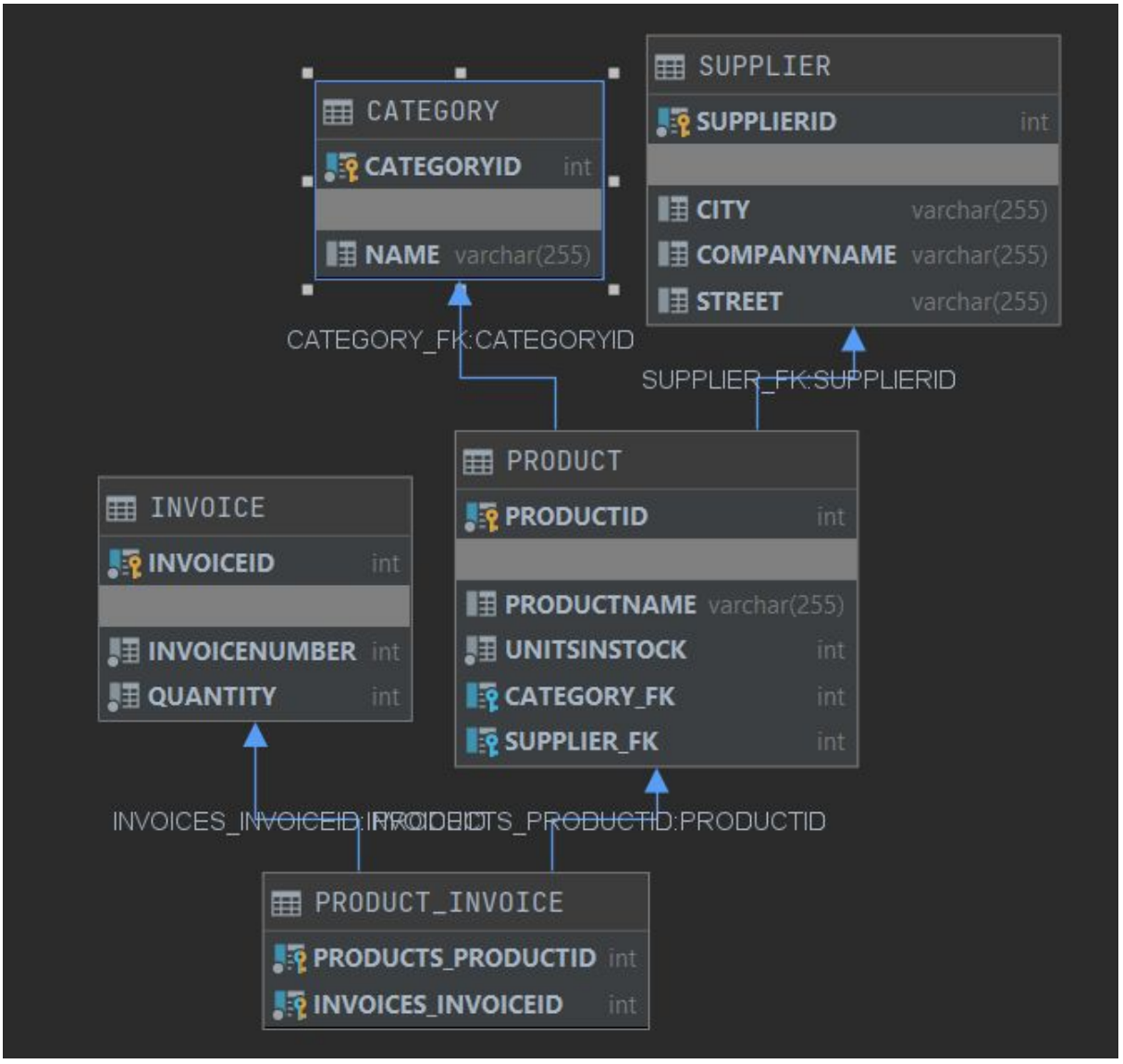
Uruchomienie programu



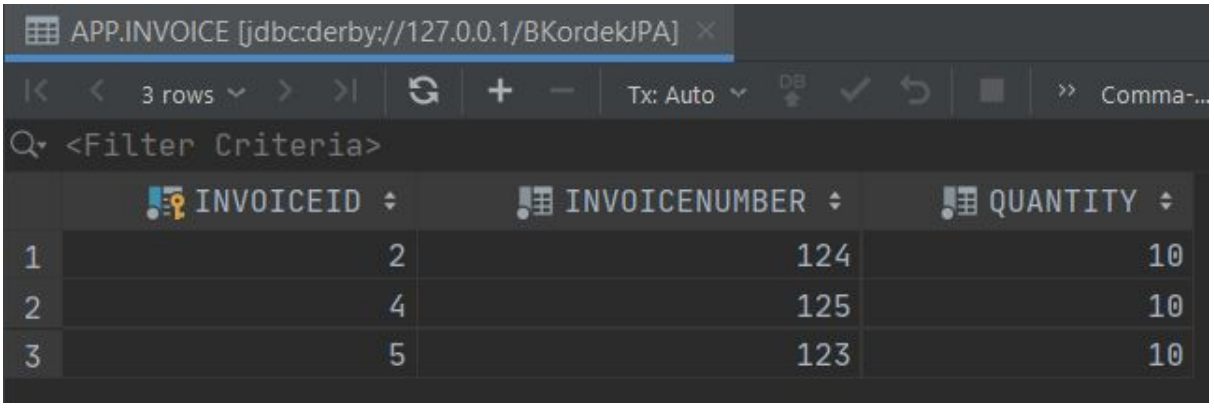
Schemat bazy danych







Zawartość bazy danych



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

APP.PRODUCT [jdbc:derby://127.0.0.1/BKordekJPA] | APP.INVOICE [jdbc:derby://127.0.0.1/BKordekJPA]

3 rows | Tx: Auto | DDL | Comma-...d (CSV) | APP.PRODUCT

<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.CATEGORY [jdbc:derby://127.0.0.1/BKordekJPA]

2 rows | Tx: Auto

<Filter Criteria>

	CATEGORYID	NAME
1	9	Mrożonki
2	10	Przekąski

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

6 rows | Tx: Auto

<Filter Criteria>

	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 | Tx: Auto | DDL | Comma-...

<Filter Criteria>

	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

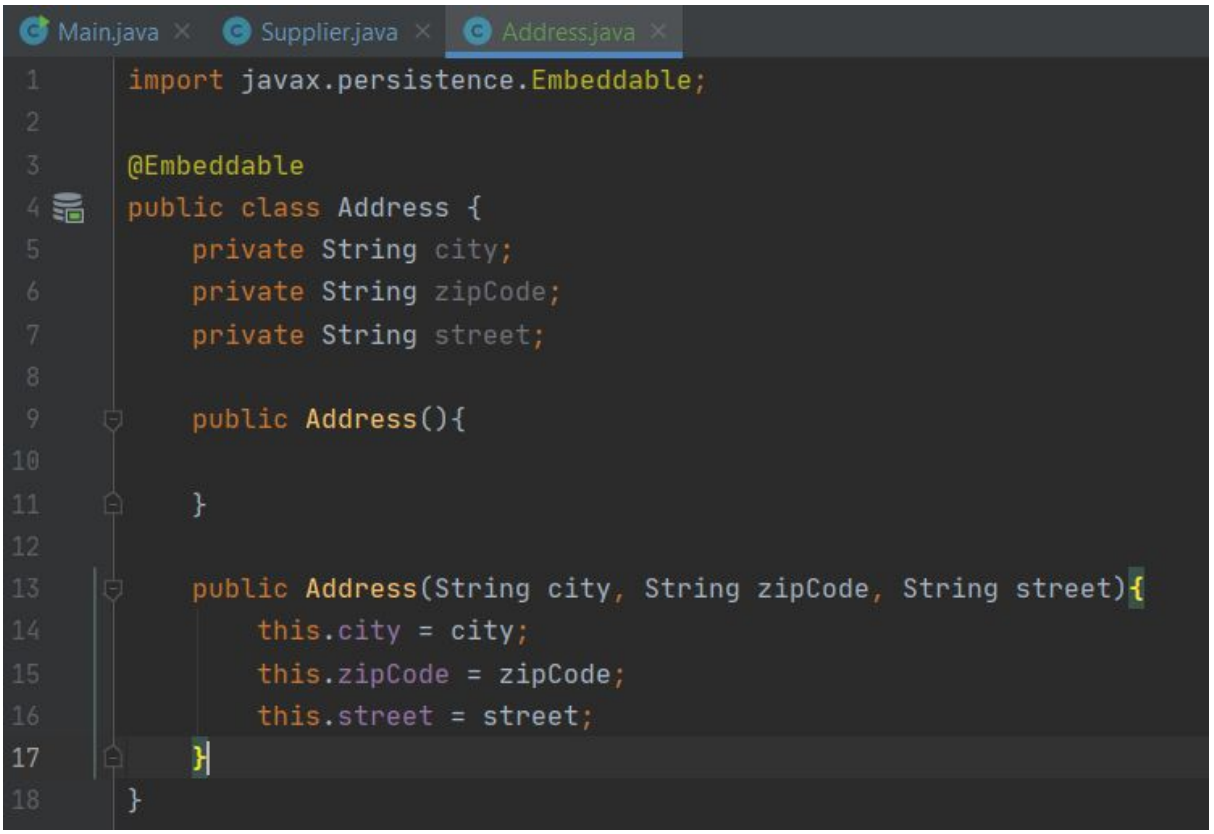
Dodaj do modelu klasę adres. „Wbuduj” ją do tabeli Dostawców

Klasa *Supplier* - modyfikacja

Main.java | Supplier.java | Address.java

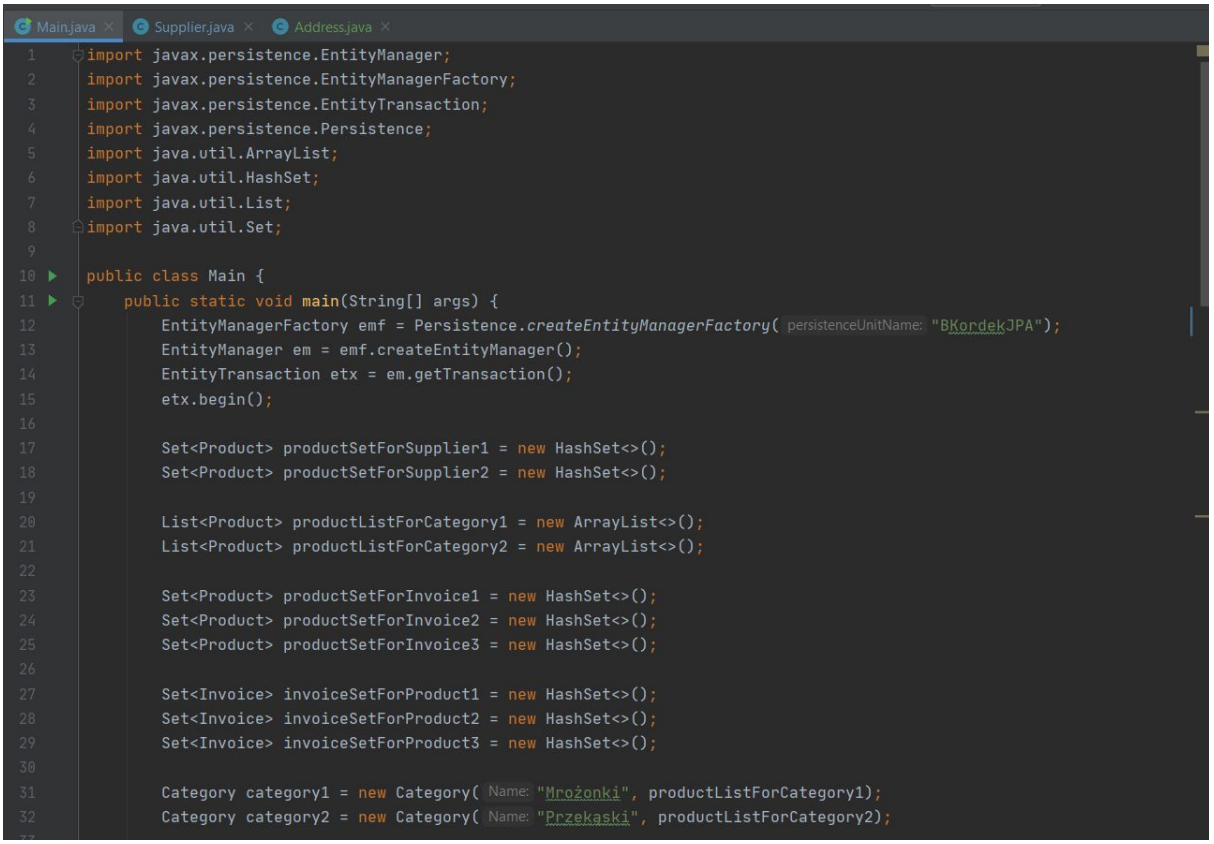
```
1
2   import javax.persistence.*;
3   import java.util.Set;
4
5   @Entity
6   public class Supplier {
7
8       @Id
9       @GeneratedValue(strategy = GenerationType.AUTO)
10      private int SupplierID;
11      private String CompanyName;
12      //private String Street;
13      //private String City;
14      @OneToMany
15      @JoinColumn(name="SUPPLIER_FK")
16      private Set<Product> products;
17      @Embedded
18      private Address address;
19
20      public Supplier(){
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



```java
import javax.persistence.Embeddable;

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

    public Address(){

    }

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

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



```java
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "BKordekJPA");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();

        Set<Product> productSetForSupplier1 = new HashSet<>();
        Set<Product> productSetForSupplier2 = new HashSet<>();

        List<Product> productListForCategory1 = new ArrayList<>();
        List<Product> productListForCategory2 = new ArrayList<>();

        Set<Product> productSetForInvoice1 = new HashSet<>();
        Set<Product> productSetForInvoice2 = new HashSet<>();
        Set<Product> productSetForInvoice3 = new HashSet<>();

        Set<Invoice> invoiceSetForProduct1 = new HashSet<>();
        Set<Invoice> invoiceSetForProduct2 = new HashSet<>();
        Set<Invoice> invoiceSetForProduct3 = new HashSet<>();

        Category category1 = new Category( Name: "Mrożonki", productListForCategory1);
        Category category2 = new Category( Name: "Przekąski", productListForCategory2);
```

Main.java | Supplier.java | Address.java

```
        Address supplier1Address = new Address( city: "Kraków",  zipCode: "30-226",  street: "Kapelanka");
        Address supplier2Address = new Address( city: "Kraków",  zipCode: "31-589",  street: "Sołtysowka");

        Supplier supplier1 = new Supplier( CompanyName: "Tesco", productSetForSupplier1, supplier1Address);
        Supplier supplier2 = new Supplier( CompanyName: "Żabka", productSetForSupplier2, supplier2Address);

        Invoice invoice1 = new Invoice( InvoiceNumber: 123, Quantity: 10,productSetForInvoice1);
        Invoice invoice2 = new Invoice( InvoiceNumber: 124, Quantity: 10,productSetForInvoice2);
        Invoice invoice3 = new Invoice( InvoiceNumber: 125, Quantity: 10,productSetForInvoice3);

        Product product1 = new Product( ProductName: "Frytki",  UnitsInStock: 20, supplier1, category1, invoiceSetForProduct1);
        Product product2 = new Product( ProductName: "Pizza",  UnitsInStock: 10, supplier1, category1, invoiceSetForProduct2);
        Product product3 = new Product( ProductName: "Chipsy",  UnitsInStock: 50, supplier2, category2, invoiceSetForProduct3);

        productSetForInvoice1.add(product1);
        productSetForInvoice1.add(product2);
        productSetForInvoice1.add(product3);

        productSetForInvoice2.add(product1);
        productSetForInvoice2.add(product3);

        productSetForInvoice3.add(product3);

        invoiceSetForProduct1.add(invoice1);
        invoiceSetForProduct1.add(invoice2);

        invoiceSetForProduct2.add(invoice1);

        invoiceSetForProduct3.add(invoice1);
        invoiceSetForProduct3.add(invoice2);
        invoiceSetForProduct3.add(invoice3);
```

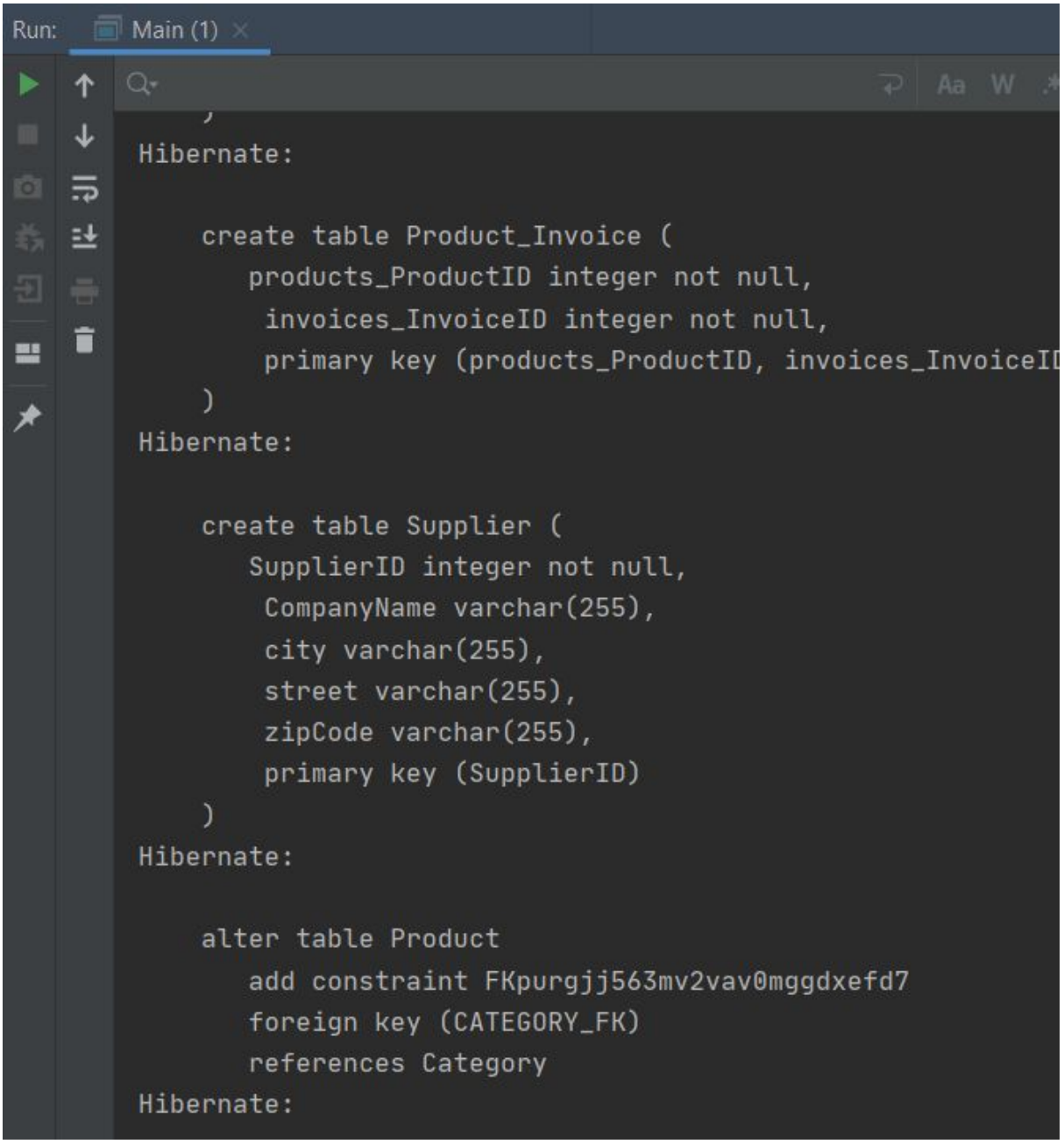
Main.java | Supplier.java | Address.java

```
        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();
    }
}
```

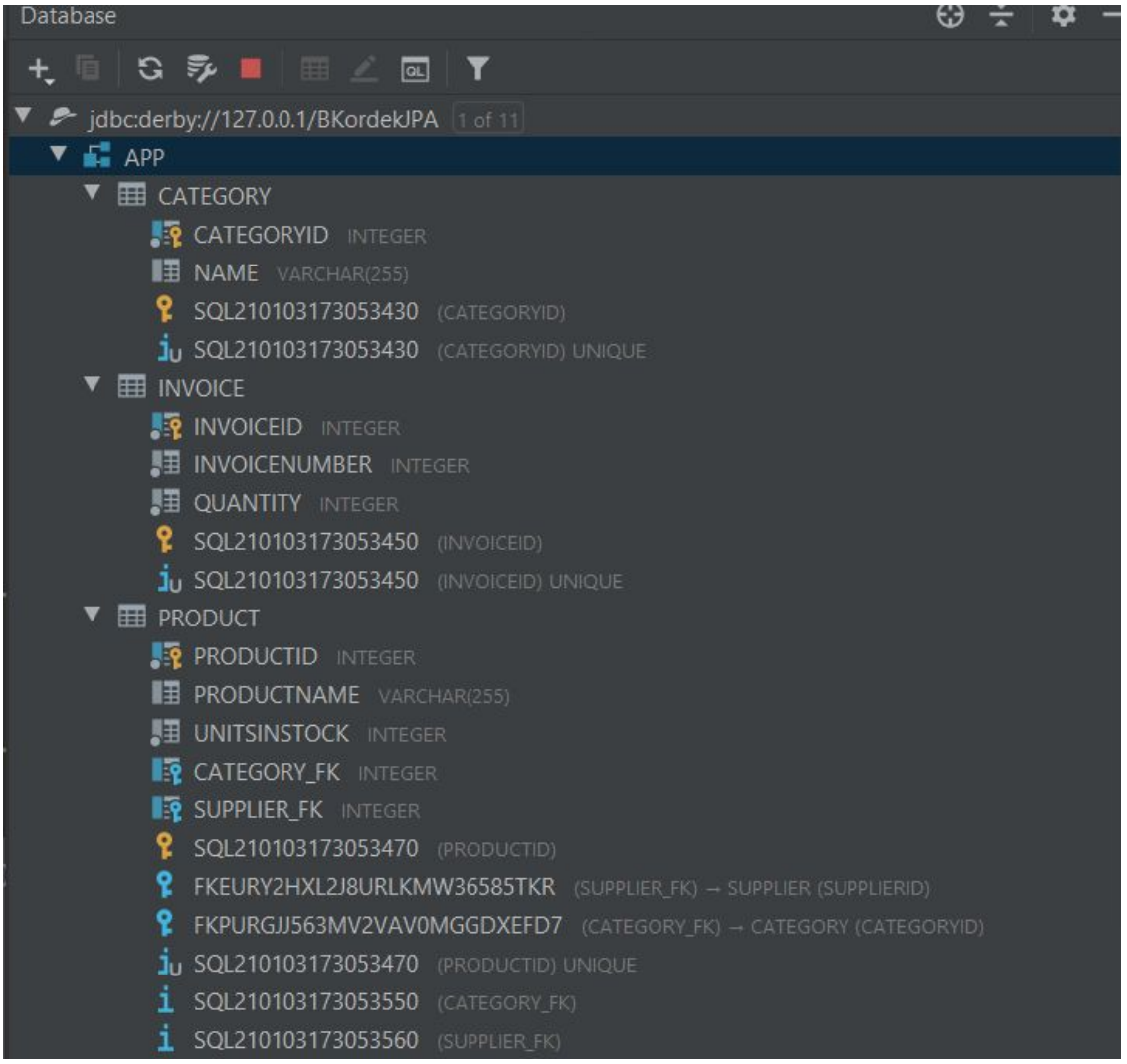
Zmiany w klasie Main

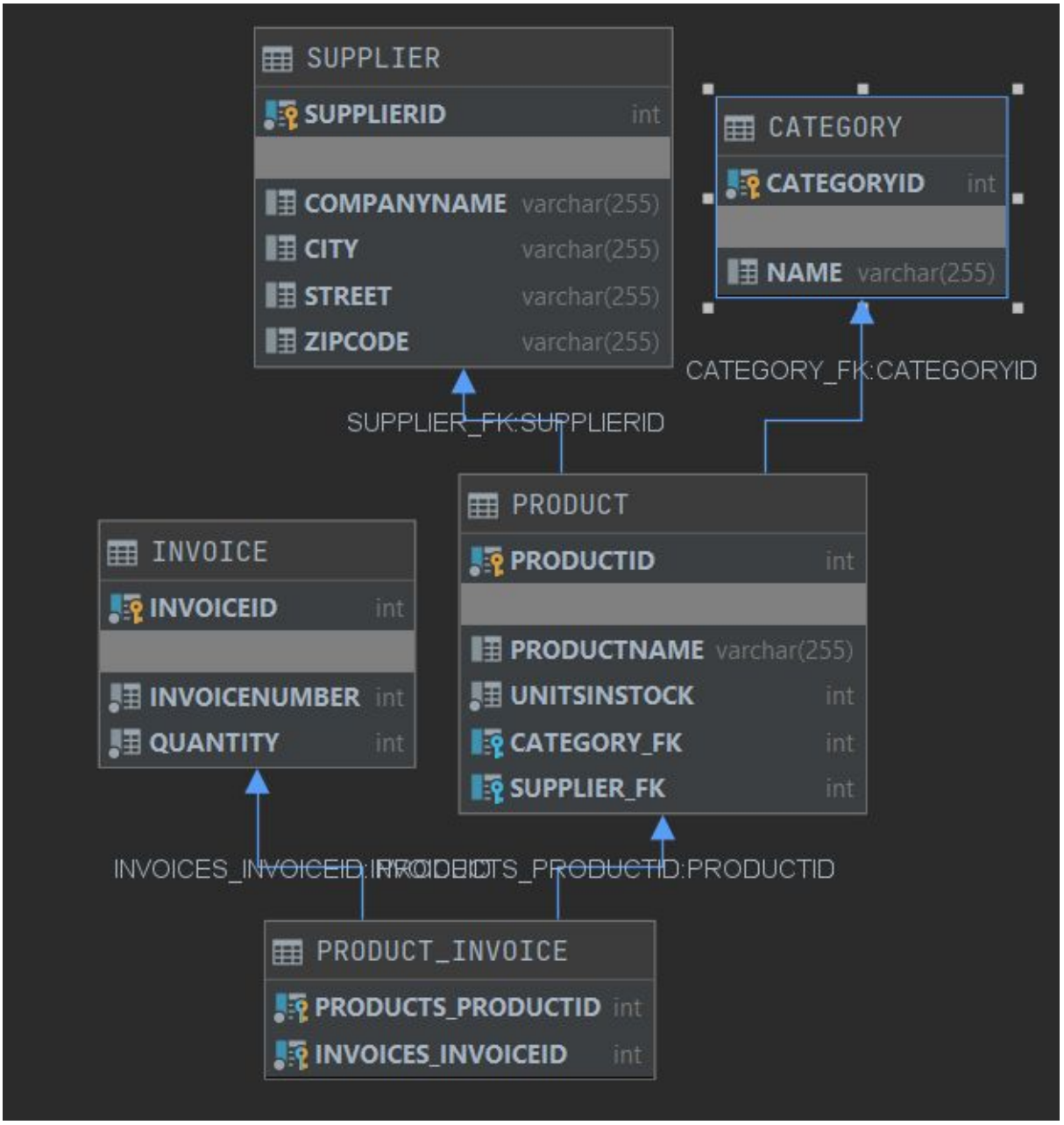
```
Address supplier1Address = new Address( city: "Kraków",  zipCode: "30-226",  street: "Kapelanka");
Address supplier2Address = new Address( city: "Kraków",  zipCode: "31-589",  street: "Sołtysowka");

Supplier supplier1 = new Supplier( CompanyName: "Tesco", productSetForSupplier1, supplier1Address);
Supplier supplier2 = new Supplier( CompanyName: "Żabka", productSetForSupplier2, supplier2Address);
```

Uruchomienie programu

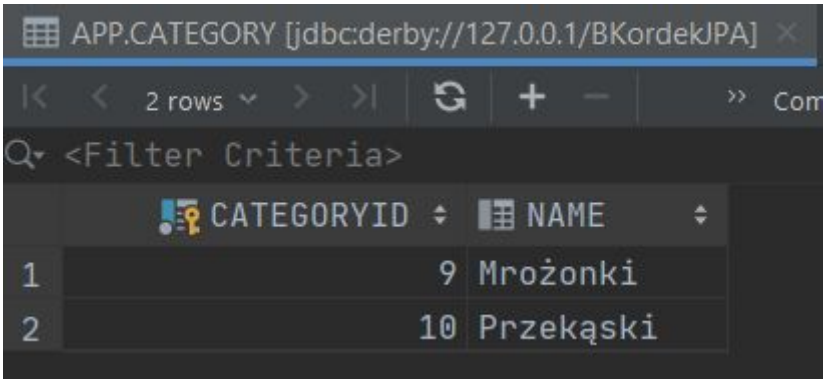


Schemat bazy danych





Zawartość tabel



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

3 rows | Comma-...d (CSV)

<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 | DDL | Comma-...d (CSV) | APP.PROD

<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 (

<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 | DDL | Comma-...d (CSV) | APP

<Filter Criteria>

	SUPPLIERID	COMPANYNAME	CITY	STREET	ZIPCODE
1	7	Tesco	Kraków	Kapelanka	30-226
2	8	Żabka	Kraków	Sołtysowka	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*

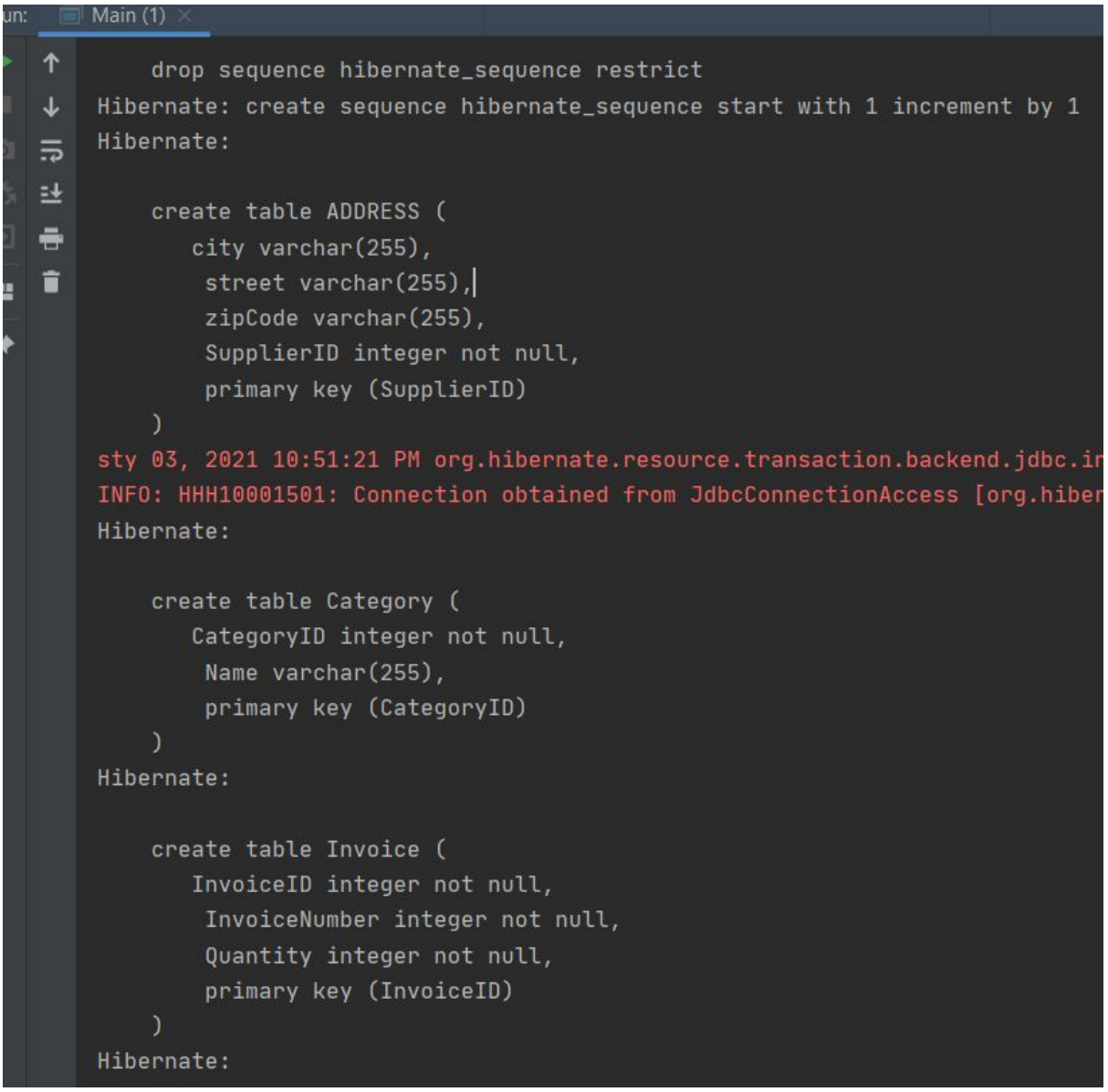
```
1
2   import javax.persistence.*;
3   import java.util.Set;
4
5   @Entity
6   @SecondaryTable(name = "ADDRESS")
7   public class Supplier {
8
9       @Id
10      @GeneratedValue(strategy = GenerationType.AUTO)
11      private int SupplierID;
12      private String CompanyName;
13      //private String 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      }
```

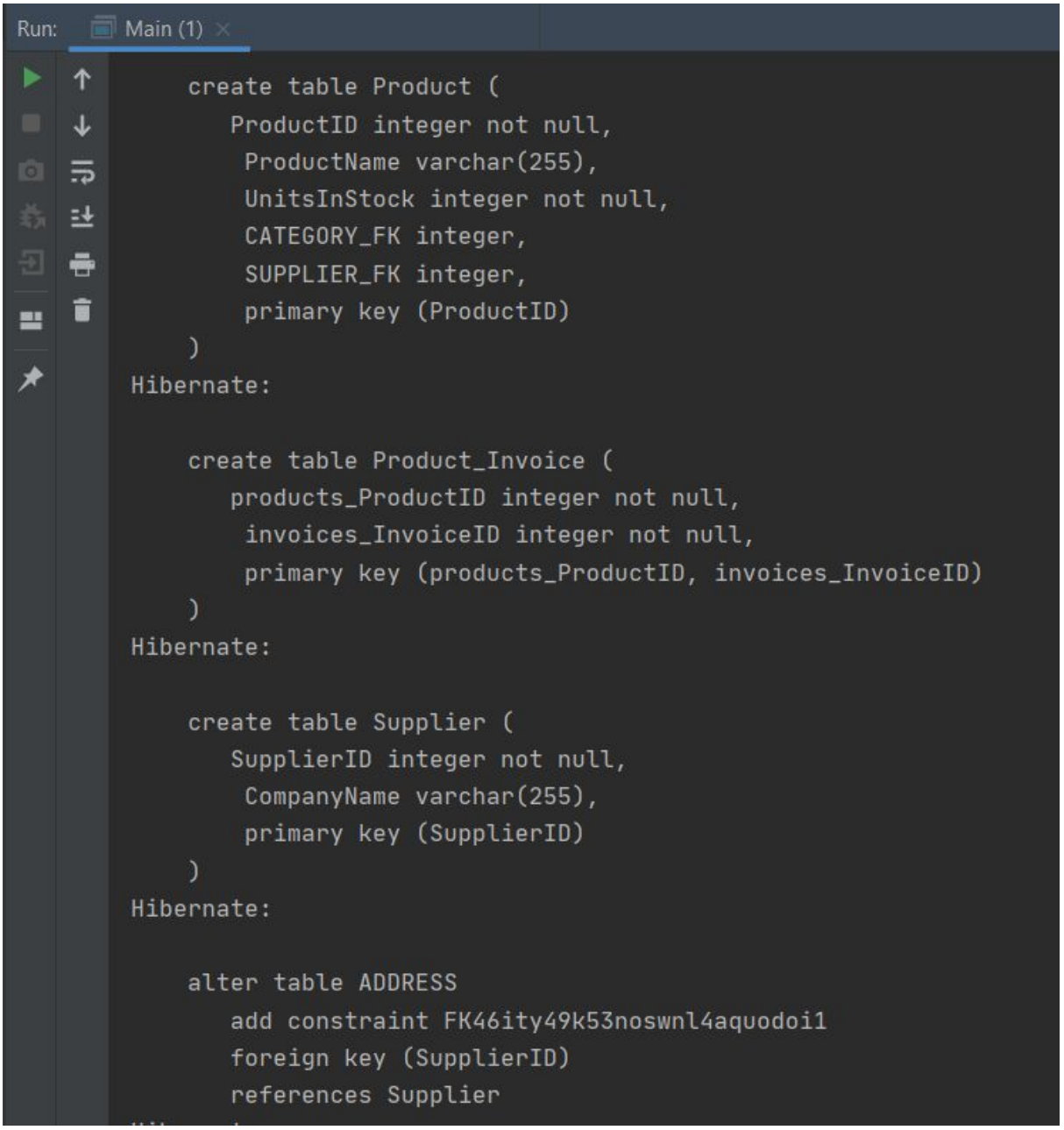
```java
    public Supplier(String CompanyName, Set<Product> products,
                    String city, String zipCode, String street){
        this.CompanyName = CompanyName;
        //this.Street = Street;
        //this.City = City;
        this.products = products;
        //this.address = address;
        this.city = city;
        this.zipCode = zipCode;
        this.street = street;
    }

    public int getSupplierID() { return SupplierID; }

    public String getCompanyName() { return CompanyName; }

    @Override
    public String toString() {
        return "Supplier{" +
                "SupplierID=" + SupplierID +
                ", CompanyName='" + CompanyName + '\'' +
                ", products=" + products +
                '}';
    }
}
```

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

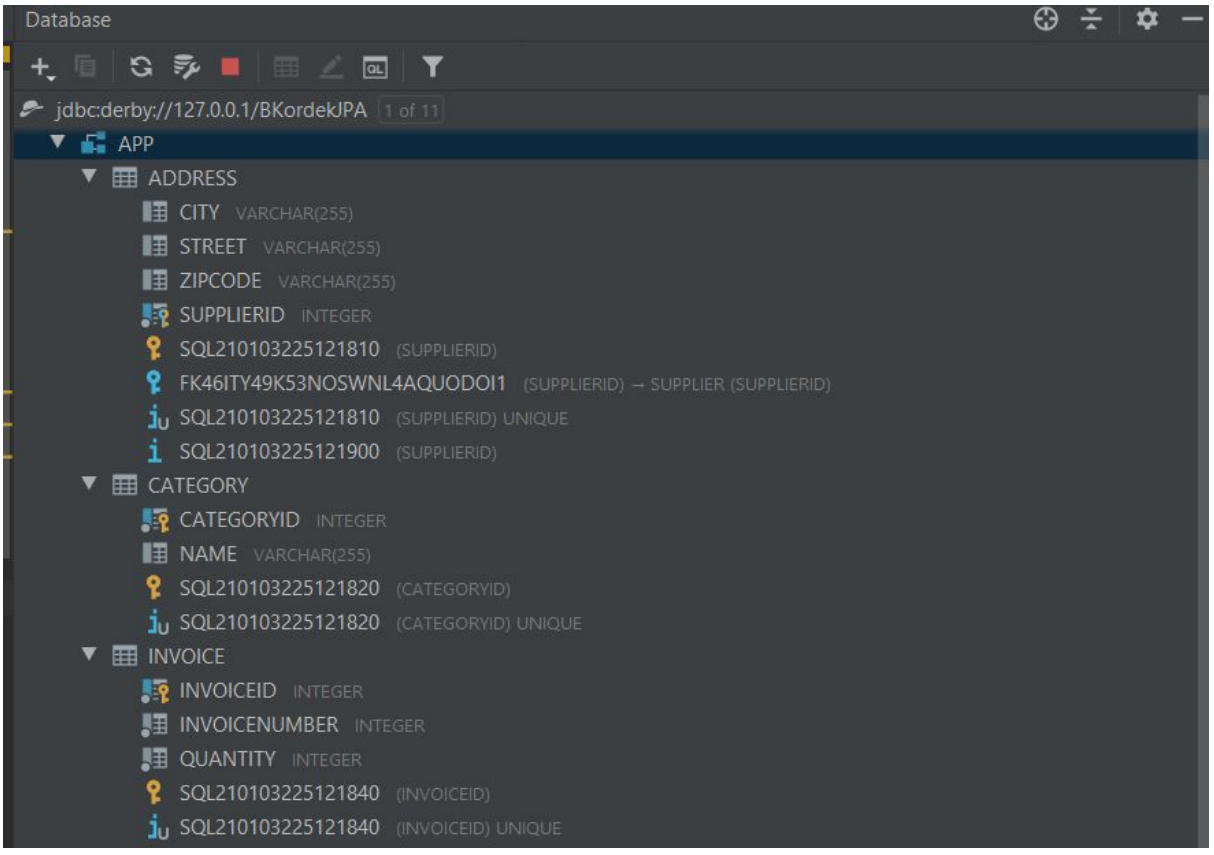
```java
//Address supplier1Address = new Address("Kraków", "30-226", "Kapelanka");
//Address supplier2Address = new Address("Kraków", "31-589", "Sołtysowka");

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: "Sołtysowka");
```

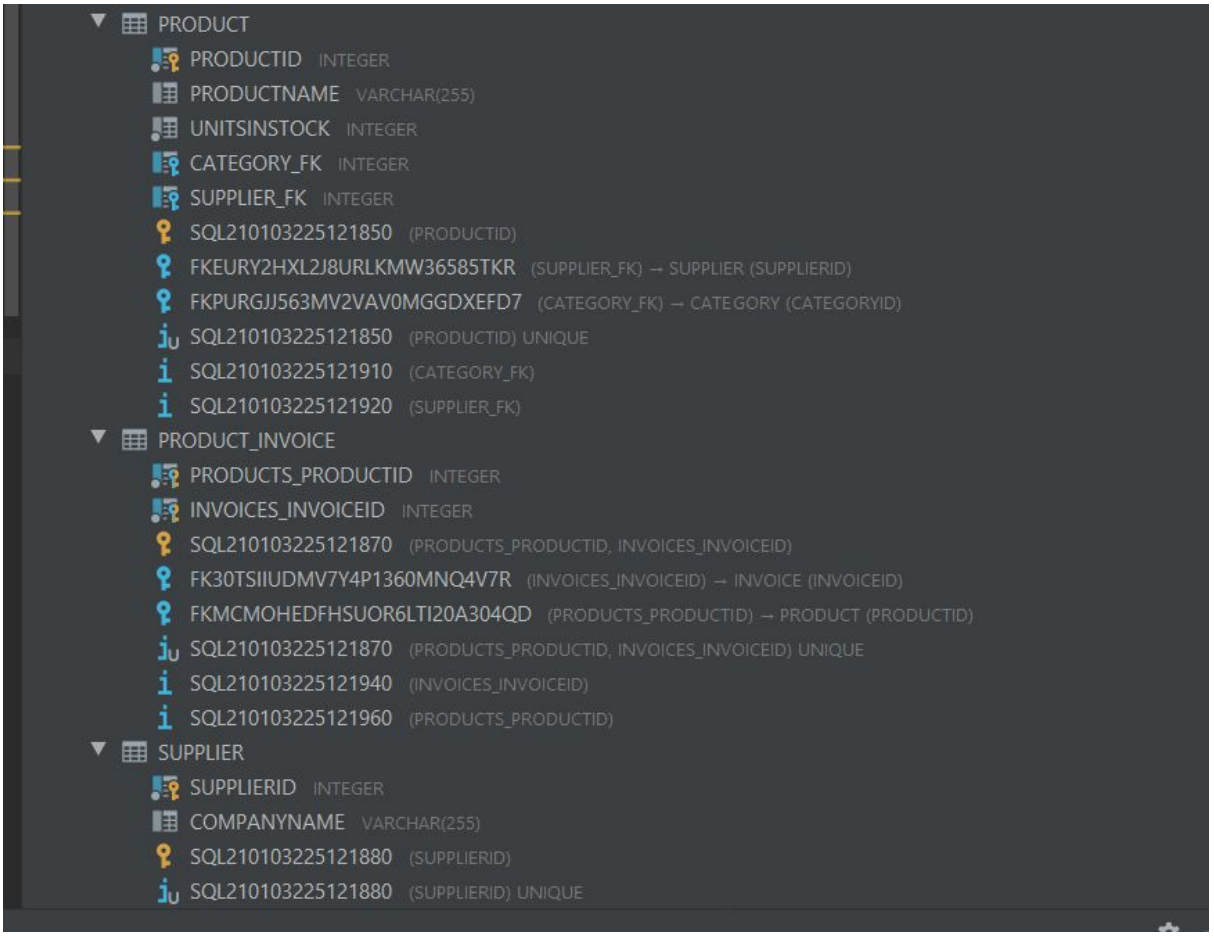
Uruchomienie programu

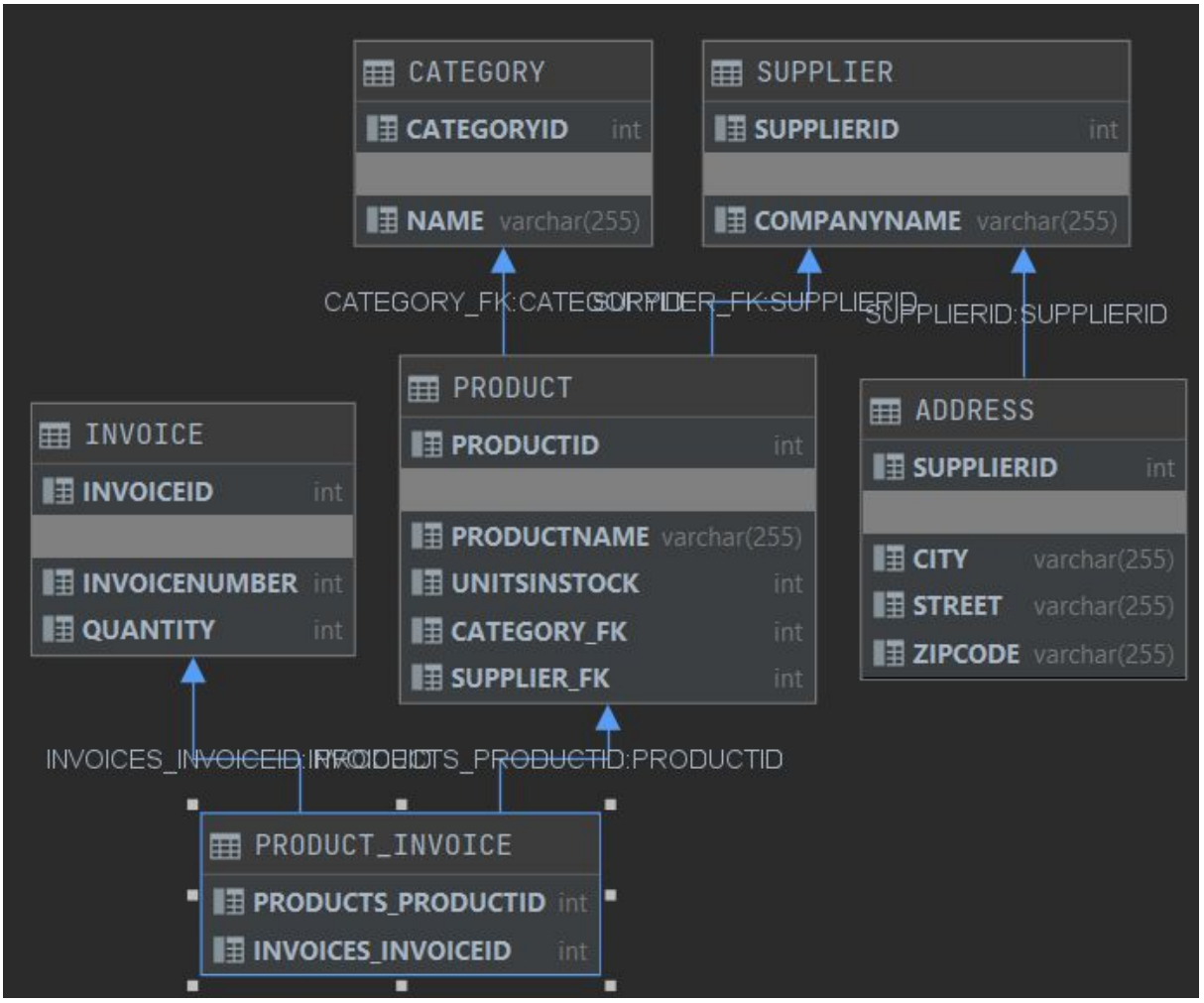



Run:
Main (1)
create table Product (
ProductID integer not null,
ProductName varchar(255),
UnitsInStock integer not null,
CATEGORY_FK integer,
SUPPLIER_FK integer,
primary key (ProductID)
)
Hibernate:
create table Product_Invoice (
products_ProductID integer not null,
invoices_InvoiceID integer not null,
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 FK46ity49k53noswnl4aquodoi1
foreign key (SupplierID)
references Supplier

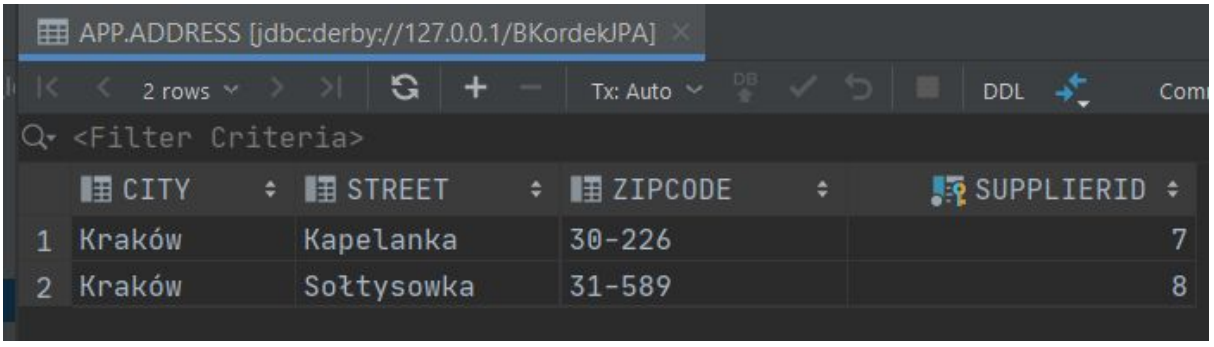
Schemat bazy danych

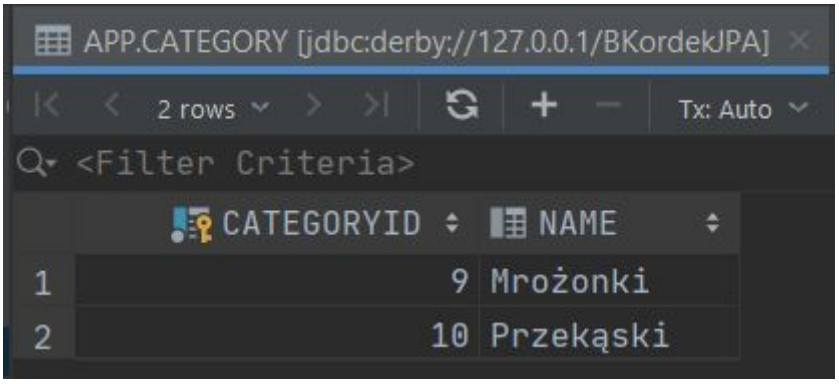






Zawartość tabel





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

3 rows | Tx: Auto | DDL

<Filter Criteria>

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

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

3 rows | Tx: Auto | DDL | Comma-...d (CSV) | APP.PRODUCT

<Filter Criteria>

	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]

6 rows | Tx: Auto | DDL | Co

<Filter Criteria>

	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]

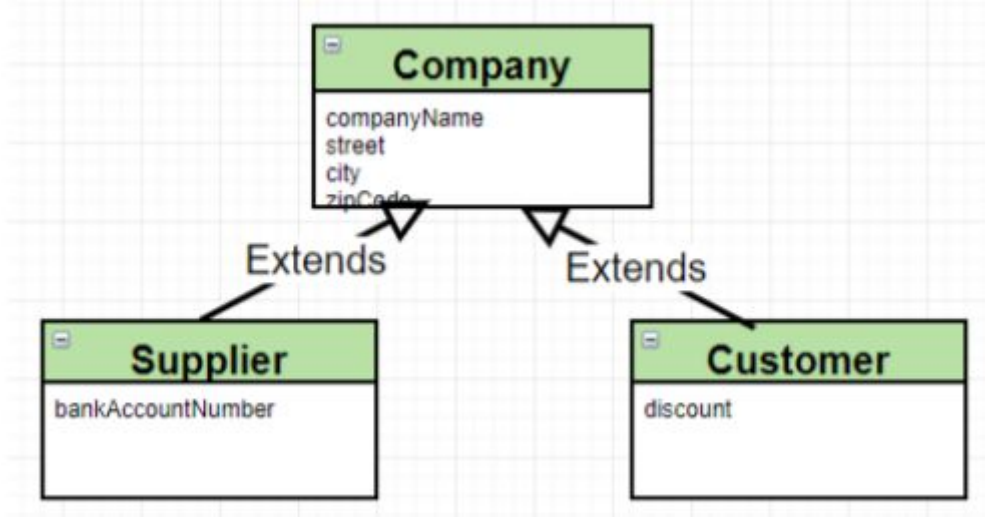
2 rows | Tx: Auto

<Filter Criteria>

	SUPPLIERID	COMPANYNAME
1	7	Tesco
2	8	Żabka

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

XIII. Dziedziczenie

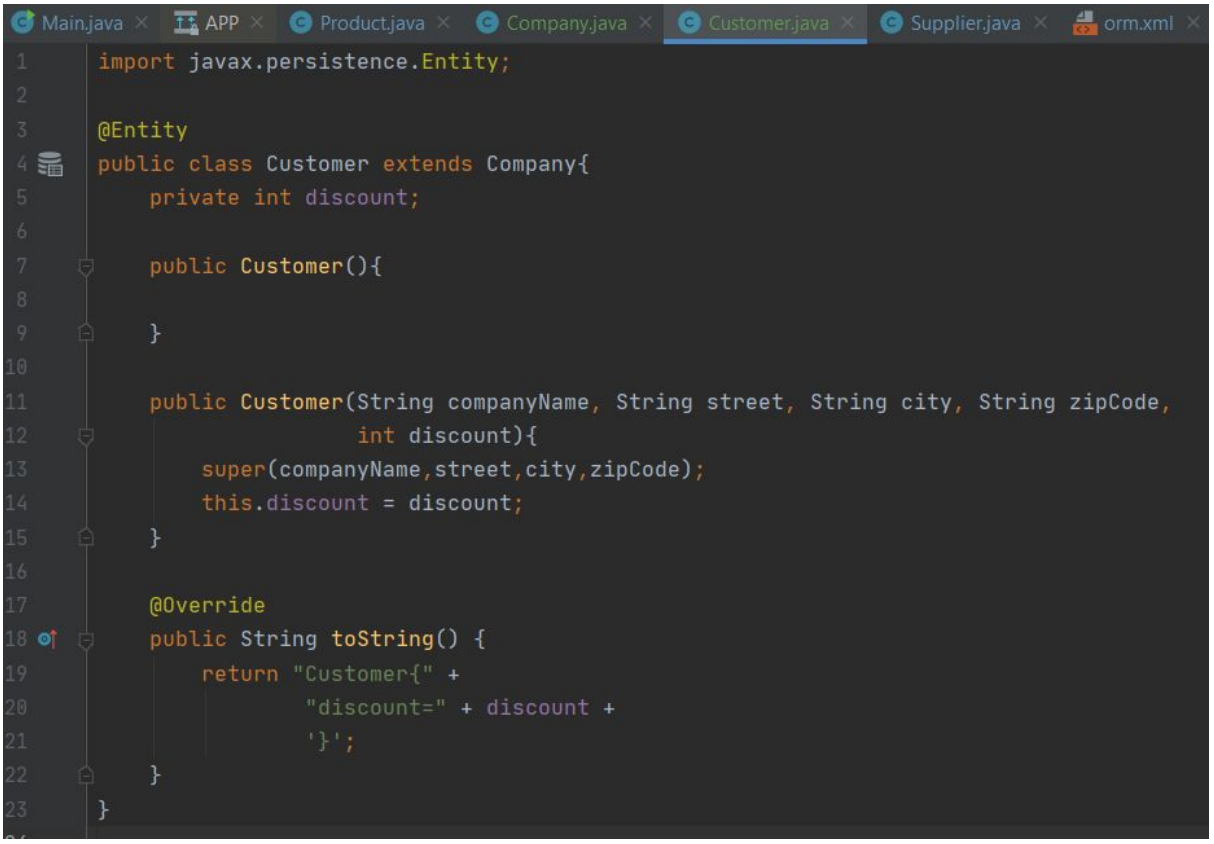


a) Single Table

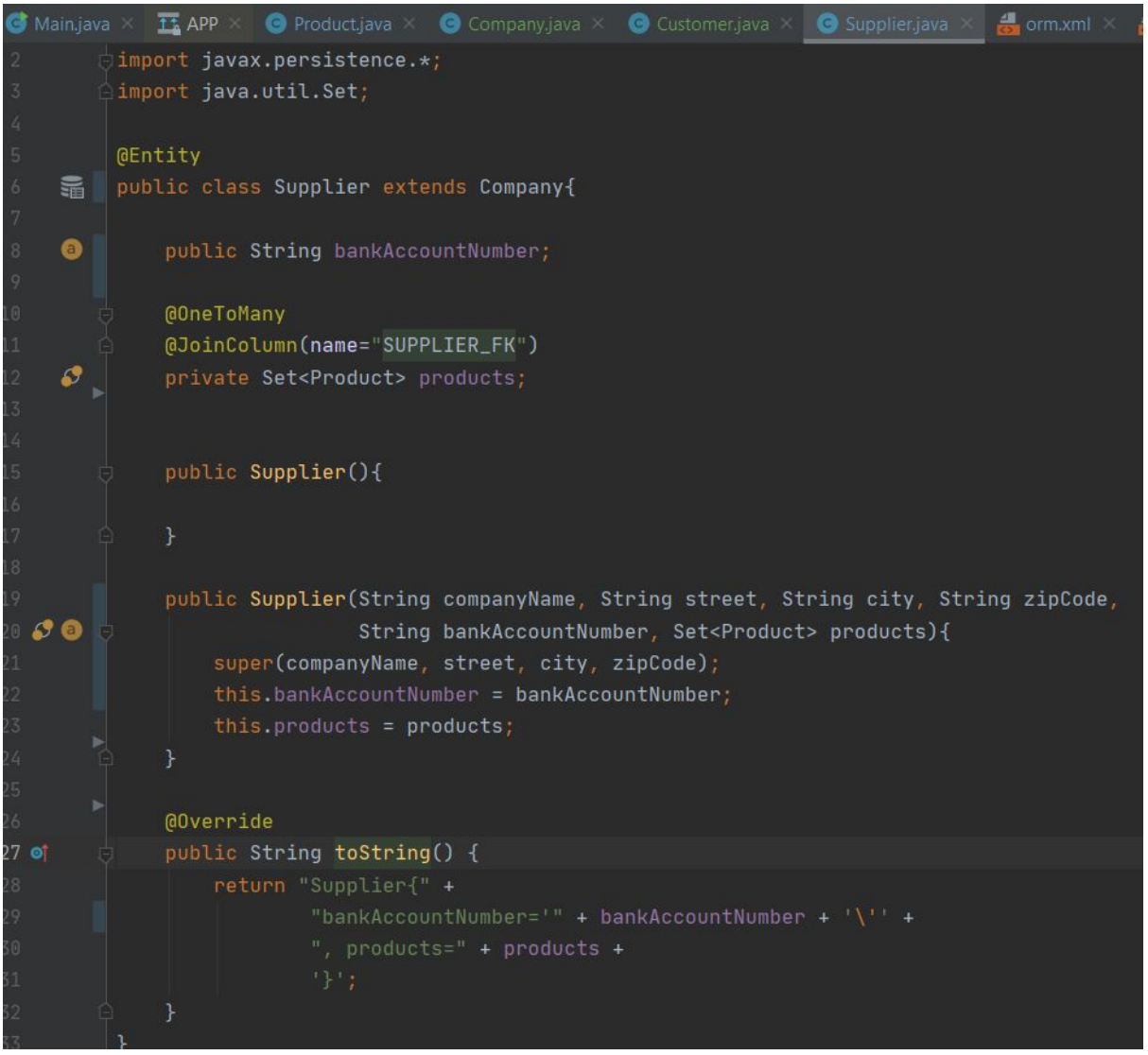
Klasa *Company*

```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int companyId;
    private String companyName;
    private String street;
    private String city;
    private String zipCode;

    public Company(){

    }

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

    @Override
    public String toString() {
        return "Company{" +
                "companyId=" + companyId +
                ", companyName='" + companyName + '\'' +
                ", street='" + street + '\'' +
                ", city='" + city + '\'' +
                ", zipCode='" + zipCode + '\'' +
                '}';
    }
```

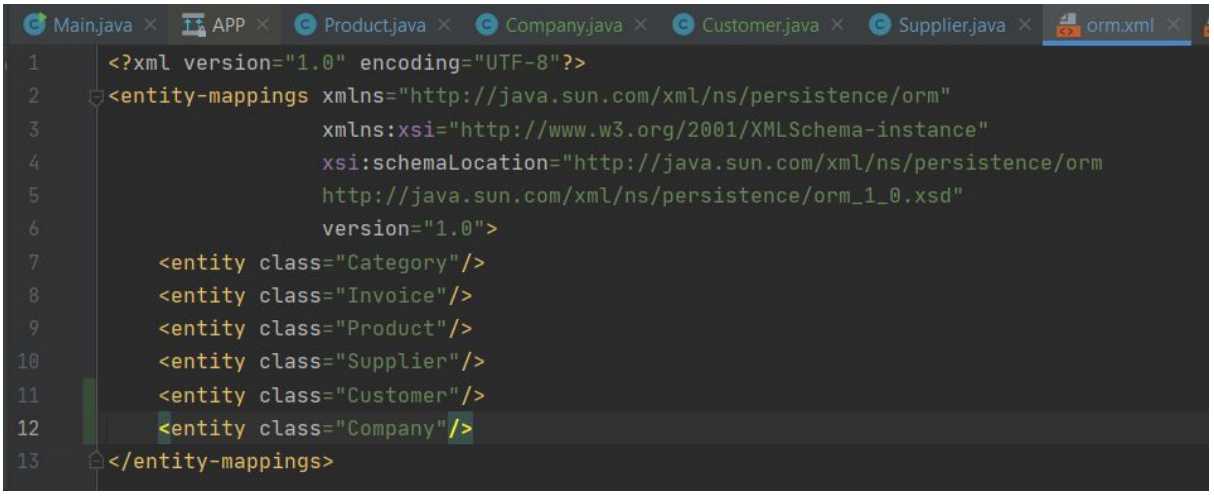
Klasa *Customer*



Klasa *Supplier*



Modyfikacja pliku konfiguracyjnego - dodanie mapowania klasy *Company*



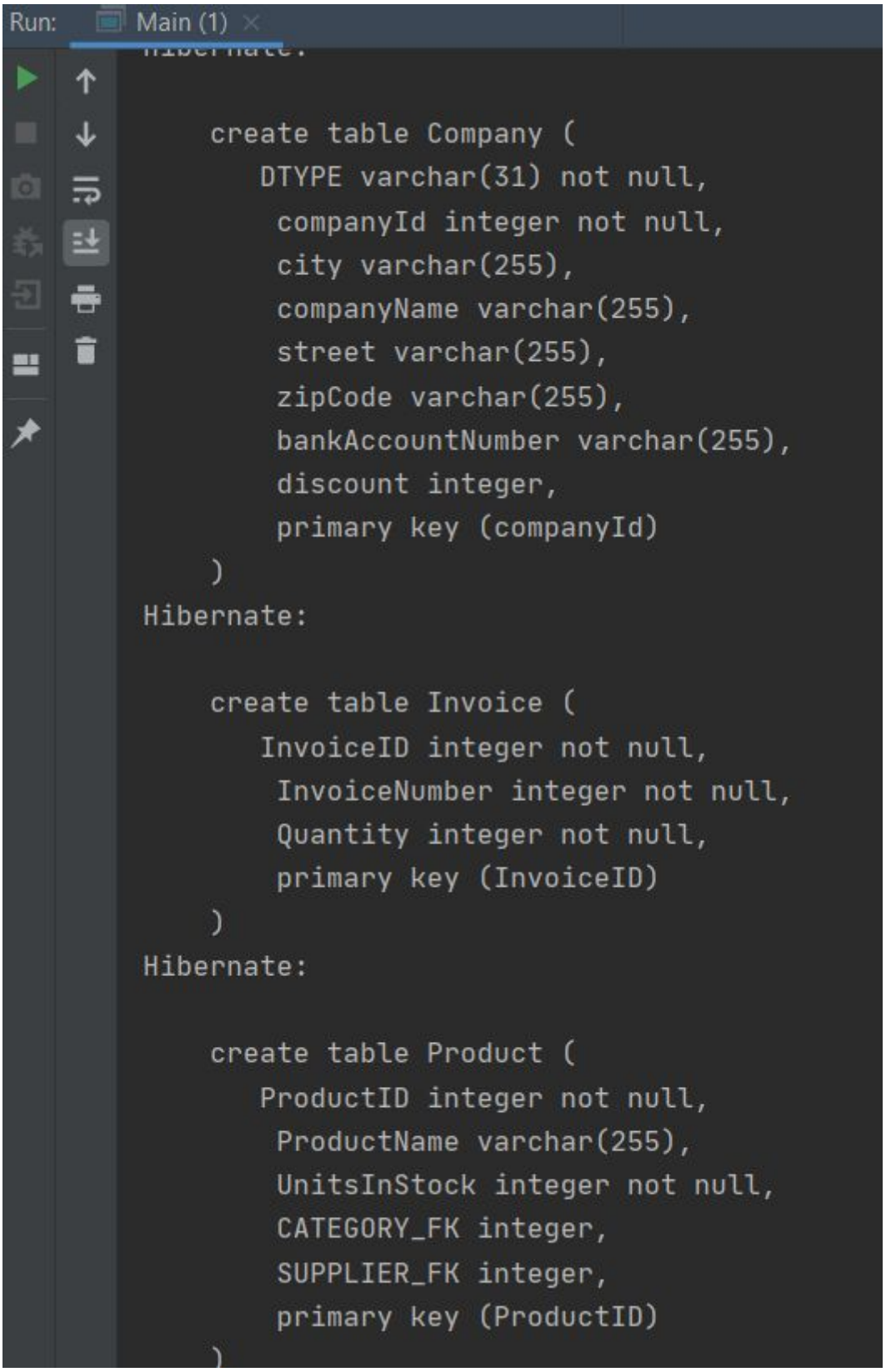
Modyfikacja klasy sterującej *Main* - modyfikacja obiektów dostawcy, stworzenie obiektów klienta

```
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: "Sołtysowska",  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);
```

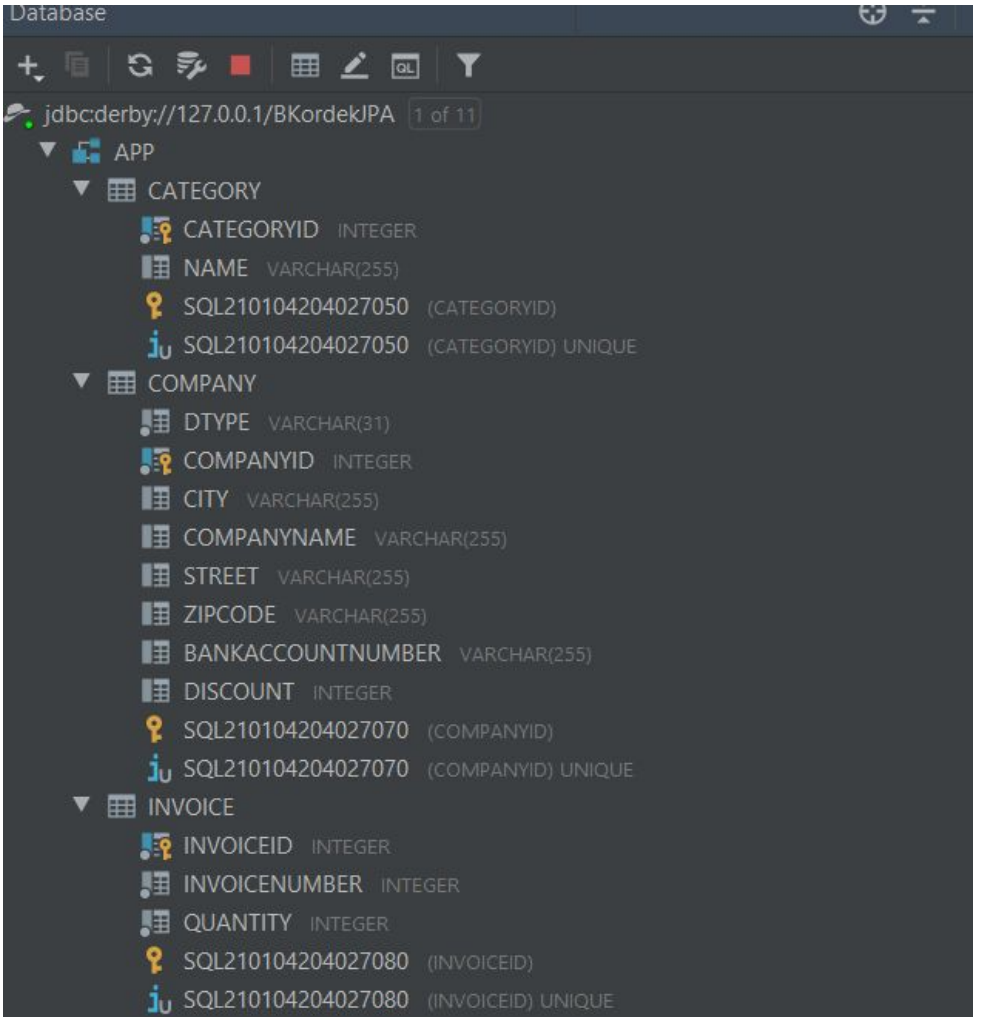
Dodanie do bazy danych obiektów klienta

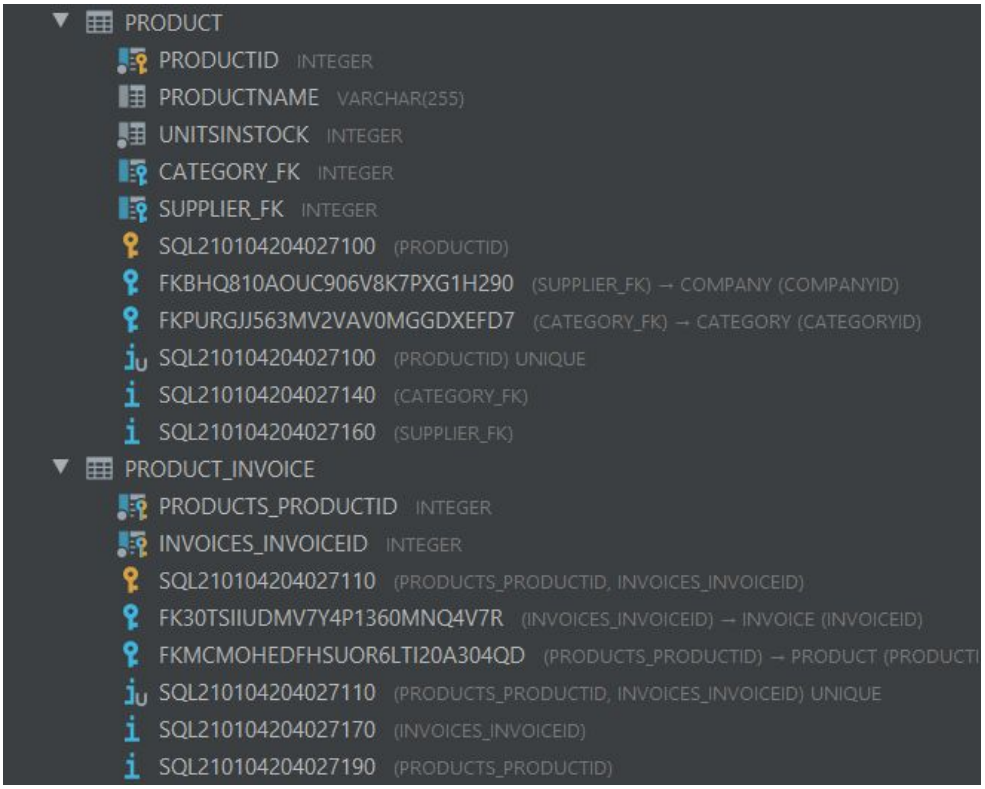
```
86
87          em.persist(supplier1);
88          em.persist(supplier2);
89
90          em.persist(customer1);
91          em.persist(customer2);
92
```

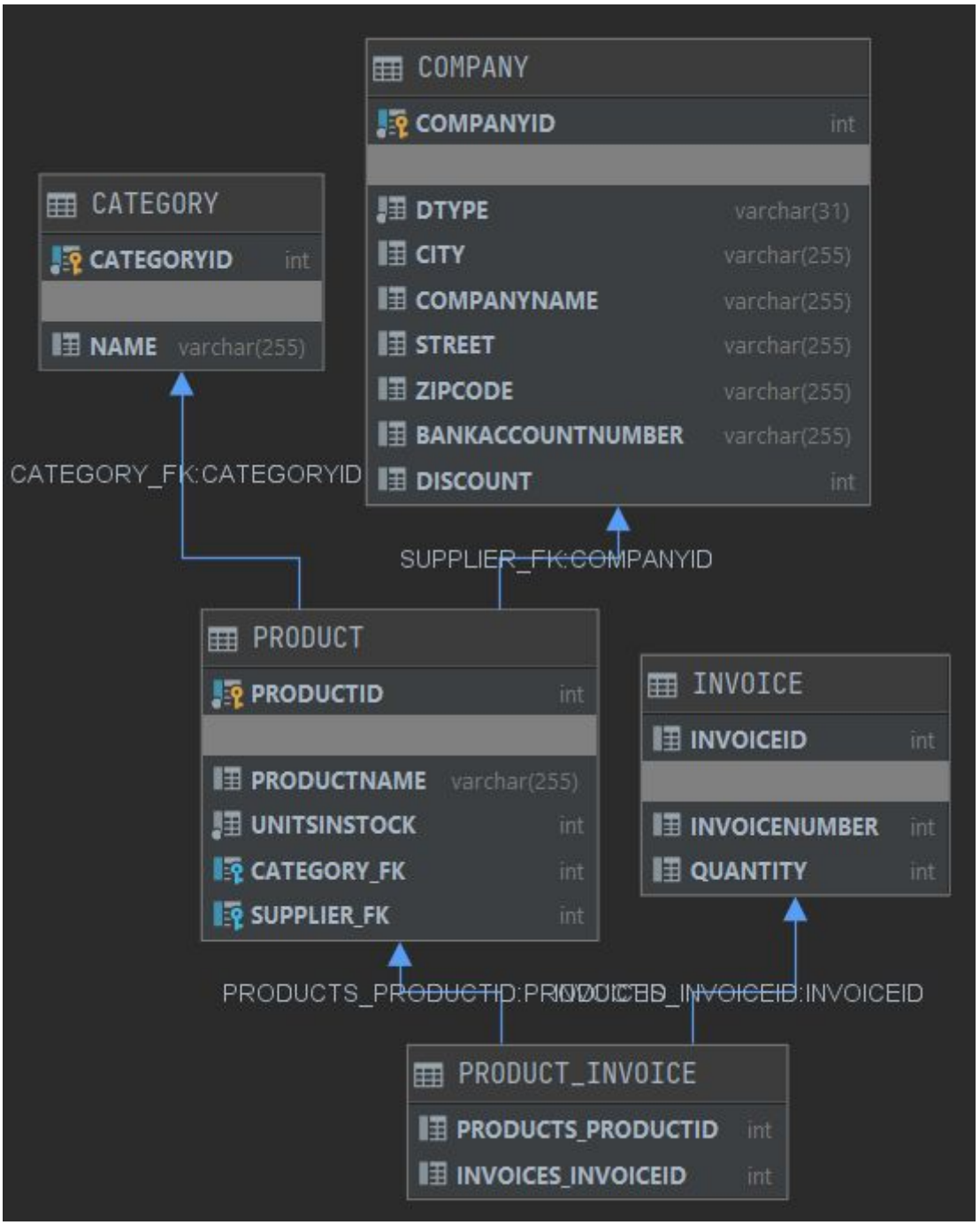
Uruchomienie programu



Schemat bazy danych







Zawartość tabeli *COMPANY*

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

4 rows | Tx: Auto | DDL | Comma-...d (CSV) | APP.COM

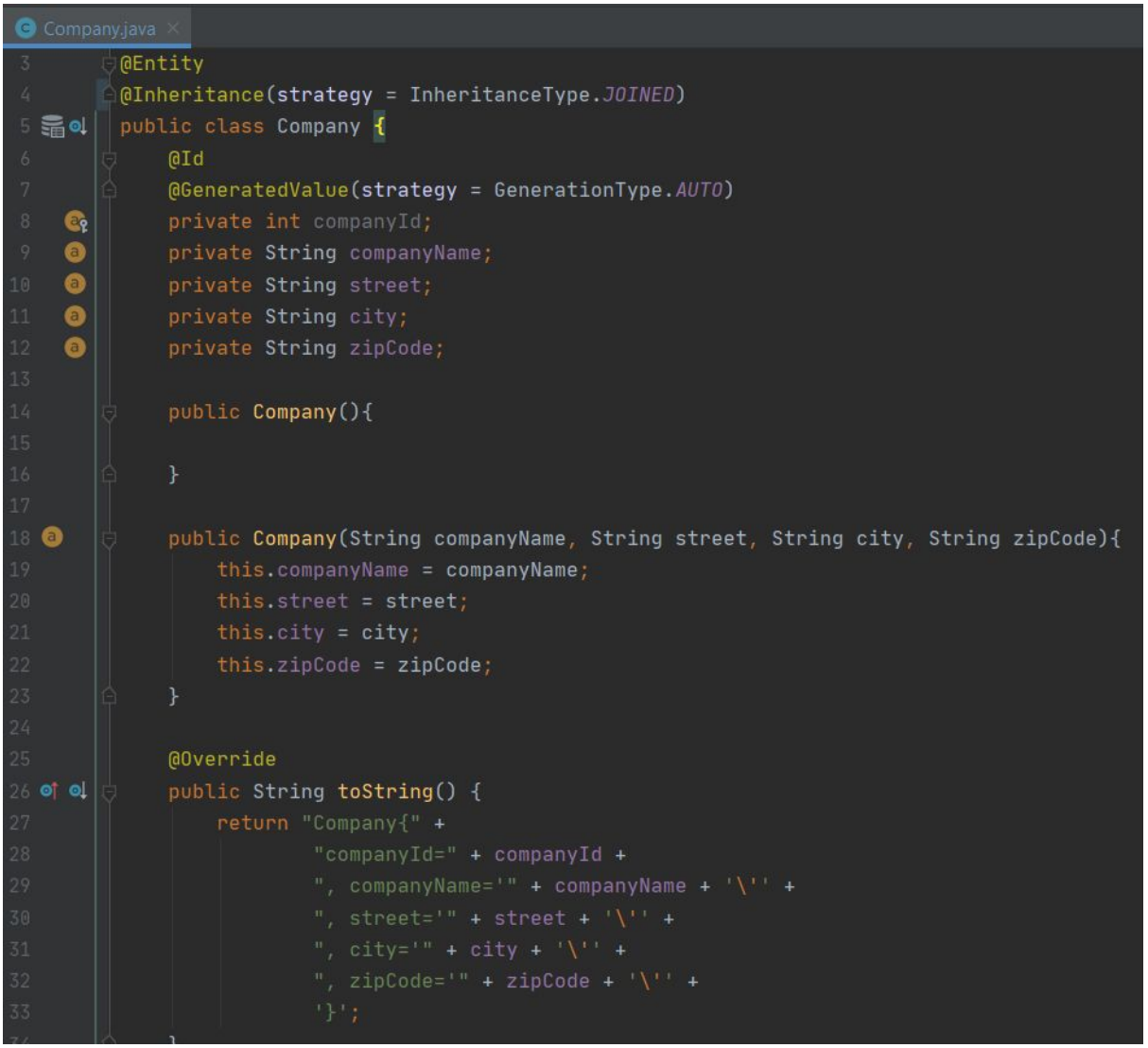
<Filter Criteria>

	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

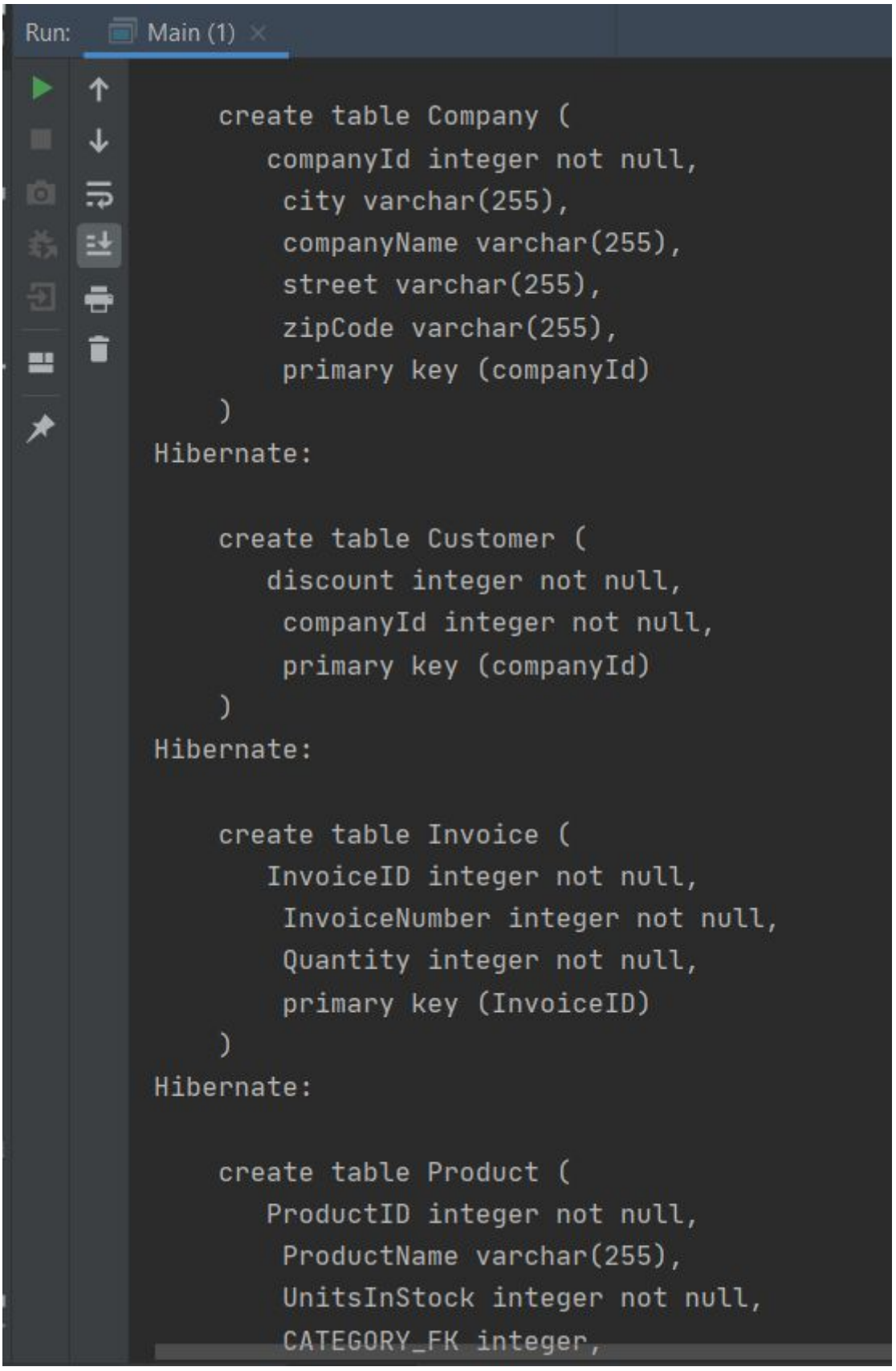
Została utworzona jedna wspólna tabela COMPANY dla klientów i dostawców.

b. Joined

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

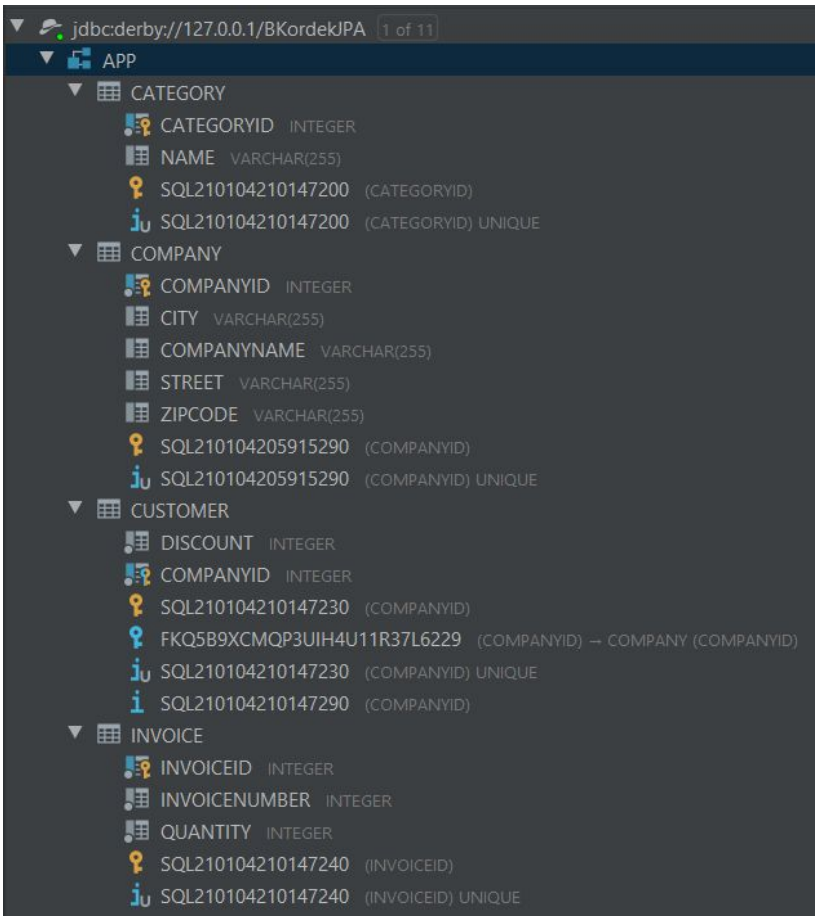


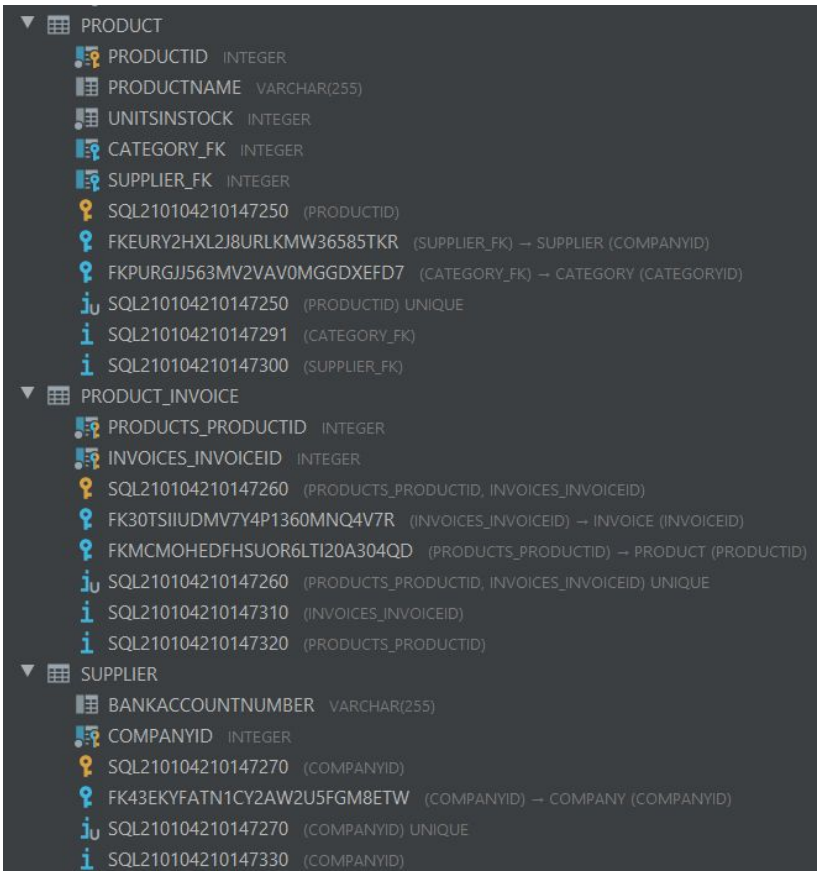
Uruchomienie programu

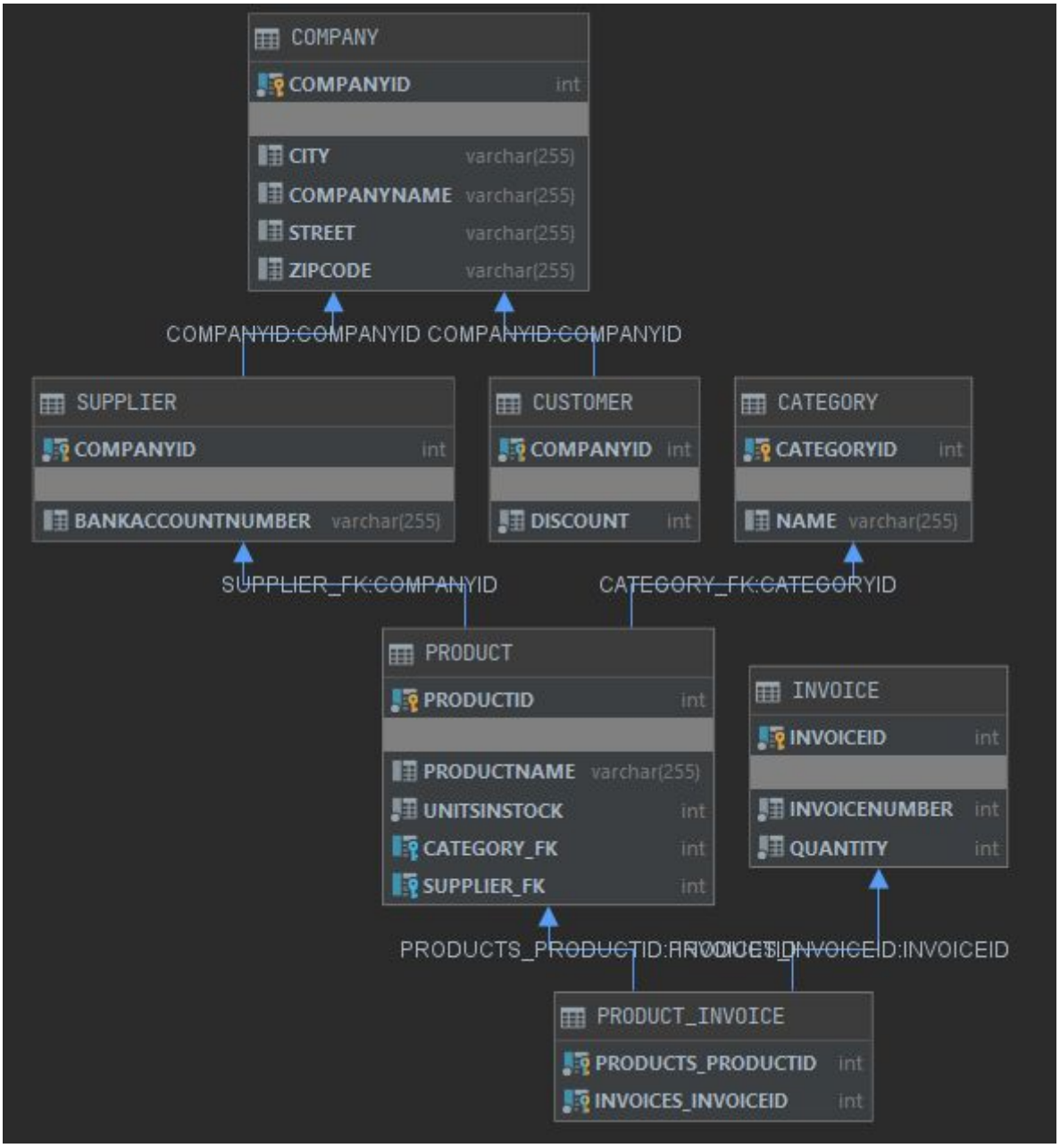


```
Hibernate:

    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







Zawartość tabeli *COMPANY*

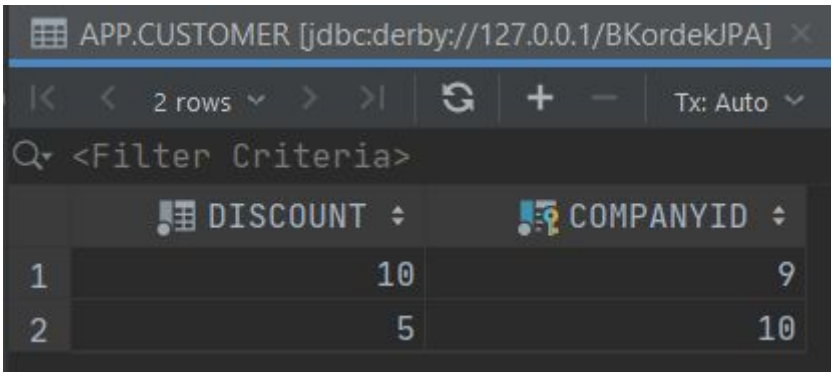
APP.COMPANY [jdbc:derby://127.0.0.1/BKordekJPA]

4 rows | Tx: Auto | DDL | Comma-...d (CSV)

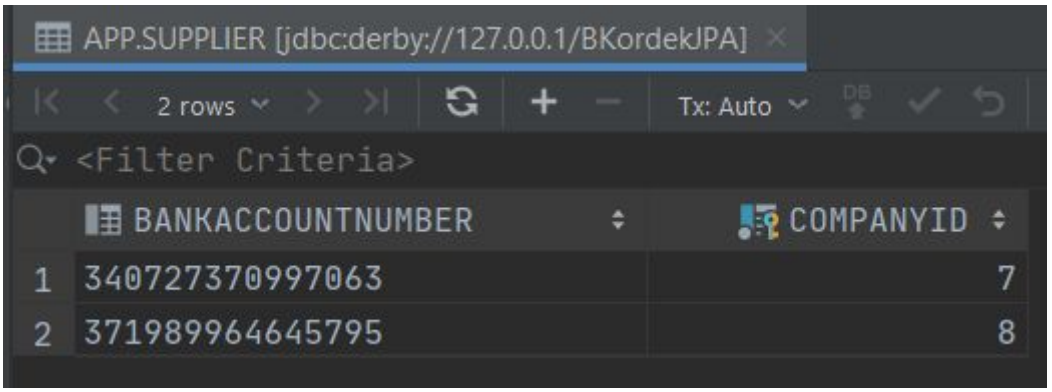
<Filter Criteria>

	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*



Zawartość tabeli *SUPPLIER*



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.

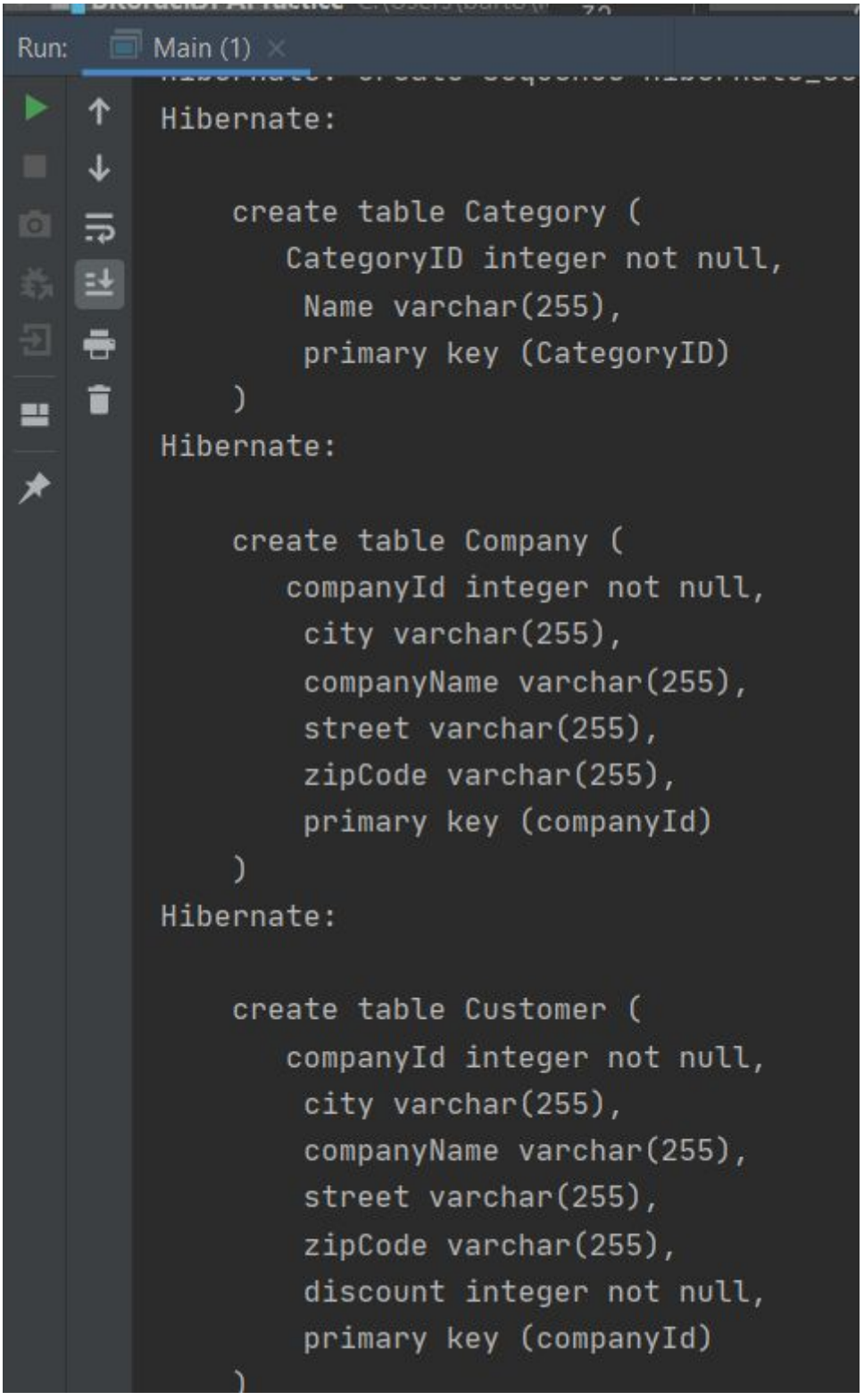
c. Table per Class

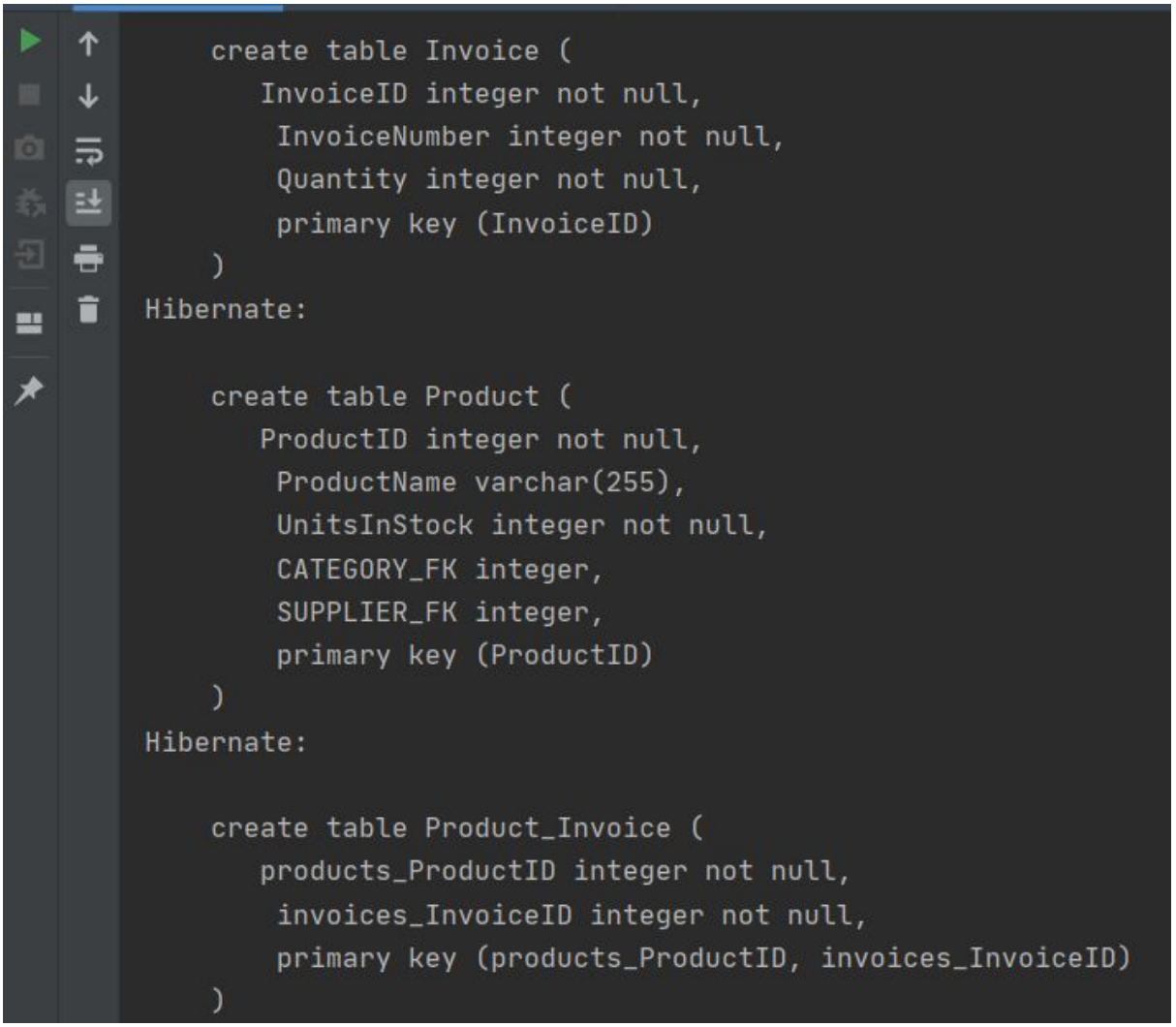
Modyfikacja klasy *Company* - zmiana typu dziedziczenia na ***TABLE_PER_CLASS***

Company.java

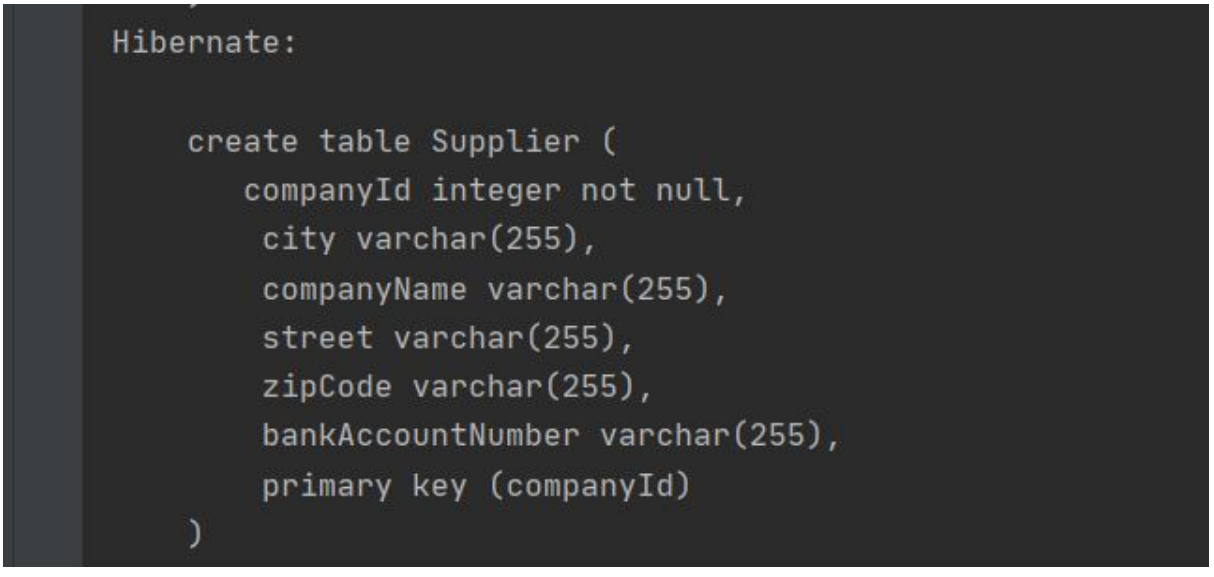
```java
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int companyId;
    private String companyName;
    private String street;
    private String city;
    private String zipCode;

    public Company(){

    }

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

    @Override
    public String toString() {
        return "Company{" +
                "companyId=" + companyId +
                ", companyName='" + companyName + '\'' +
                ", street='" + street + '\'' +
                ", city='" + city + '\'' +
                ", zipCode='" + zipCode + '\'' +
                '}';
```

Uruchomienie programu



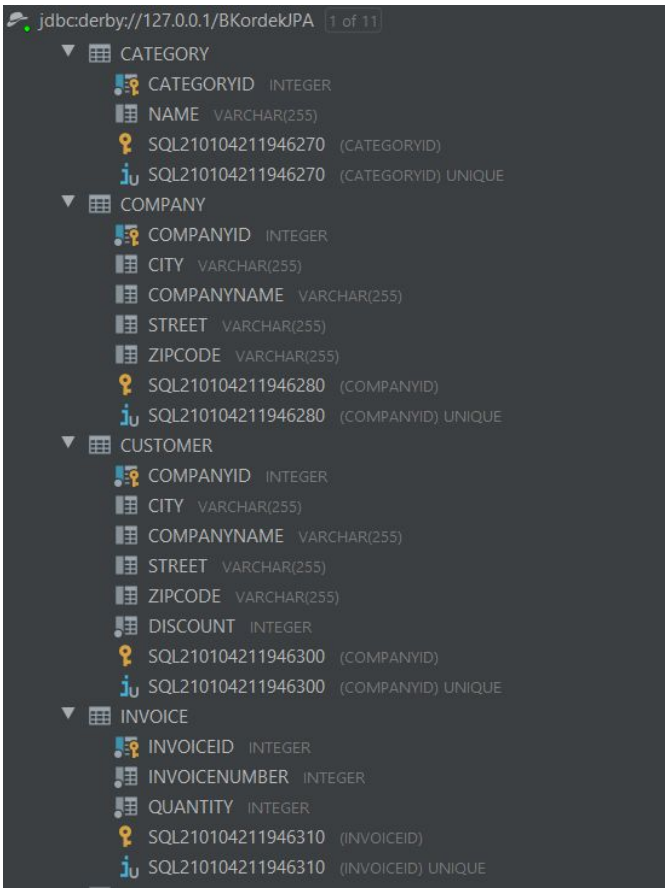


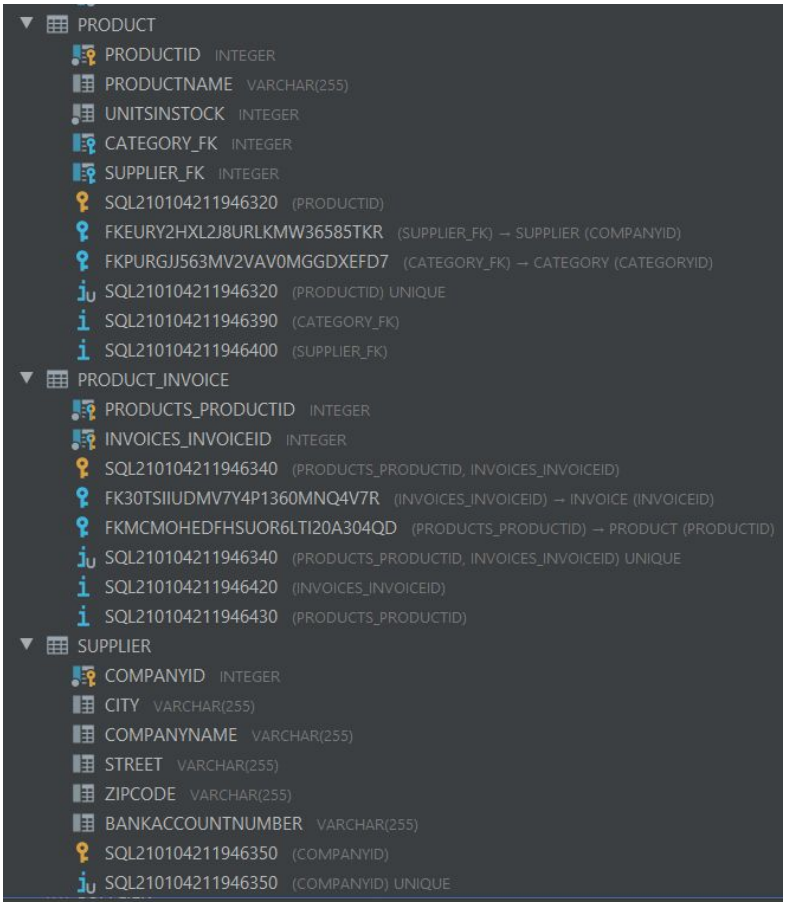
```
    create table Invoice (
       InvoiceID integer not null,
        InvoiceNumber integer not null,
        Quantity integer not null,
        primary key (InvoiceID)
    )
Hibernate:

    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)
    )
```

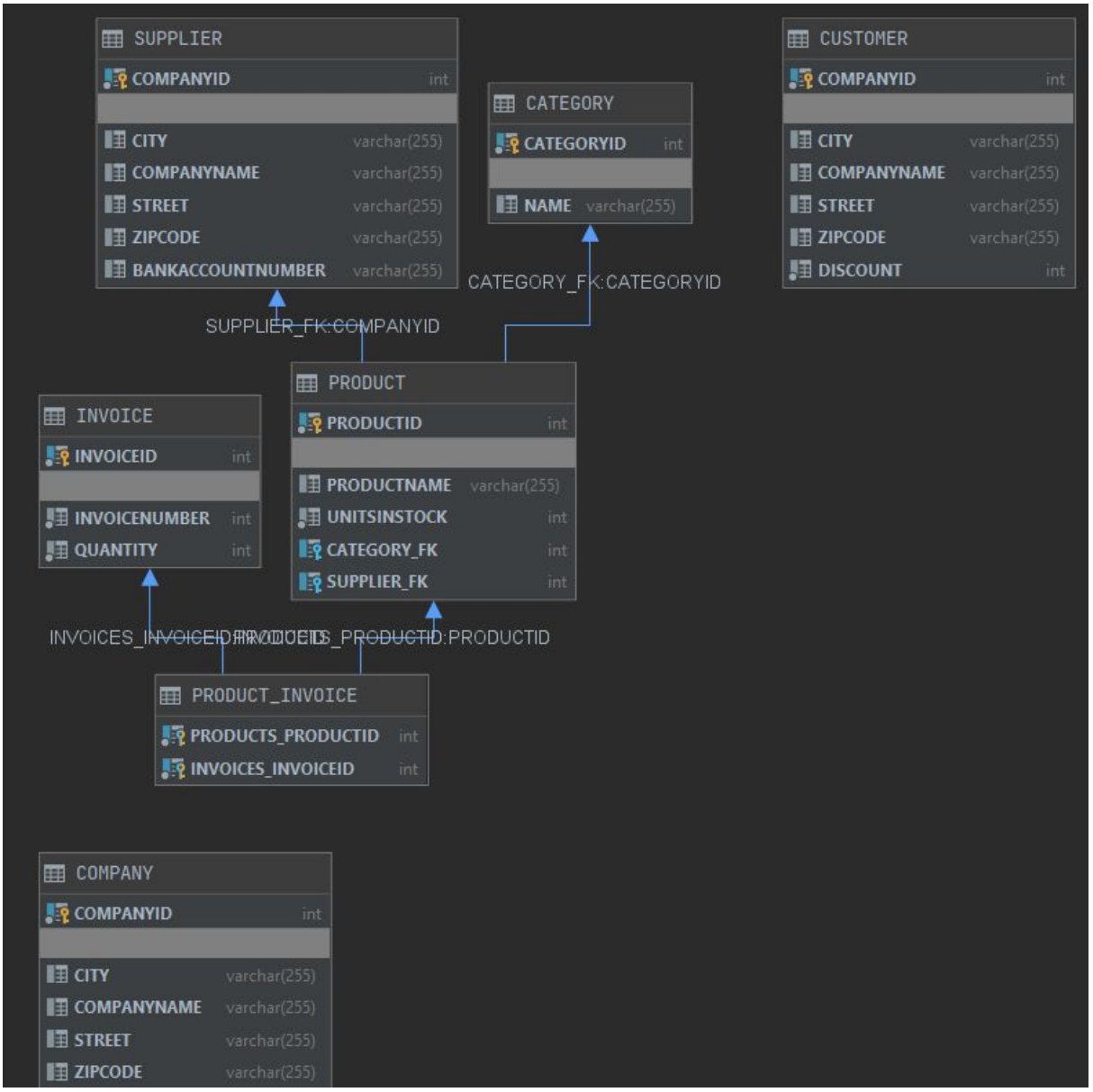


```
Hibernate:

    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)
    )
```

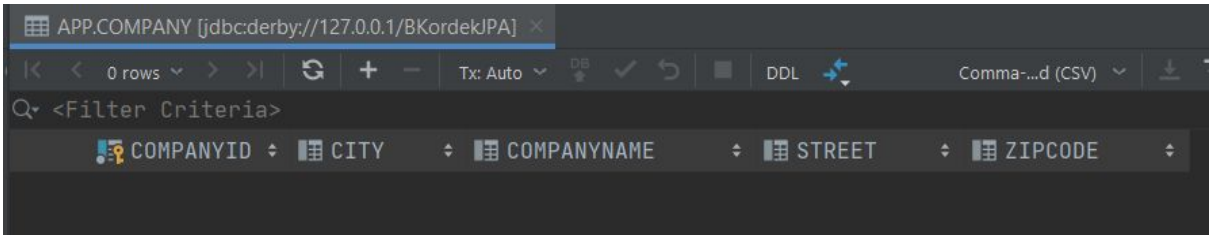
Schemat bazy danych



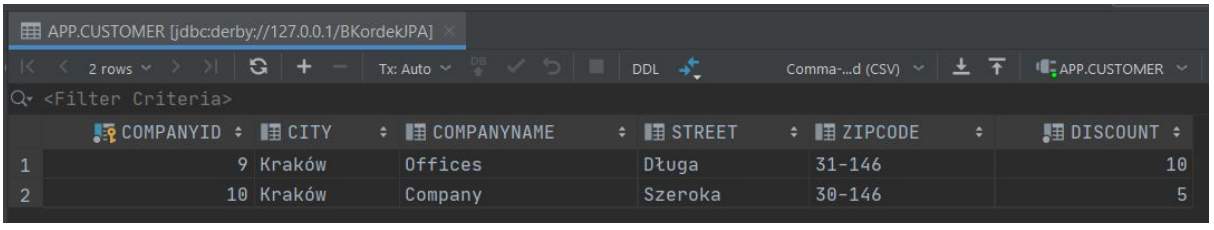




Zawartość tabeli *COMPANY* - pusta



Zawartość tabeli *CUSTOMER*



Zawartość tabeli *SUPPLIER*

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

2 rows | Tx: Auto | DDL | Comma-...d (CSV) | APP.SUPPLIER

<Filter Criteria>

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	7	Kraków	Tesco	Kapelanka	30-226	340727370997063
2	8	Kraków	Żabka	Sołtysowska	31-589	371989964645795

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=6, ProductName='Pizza', UnitsInStock=10, category=11 Mrożonki}, Product{ProductID=1, ProductName='Frytki
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.