Hibernate, JPA

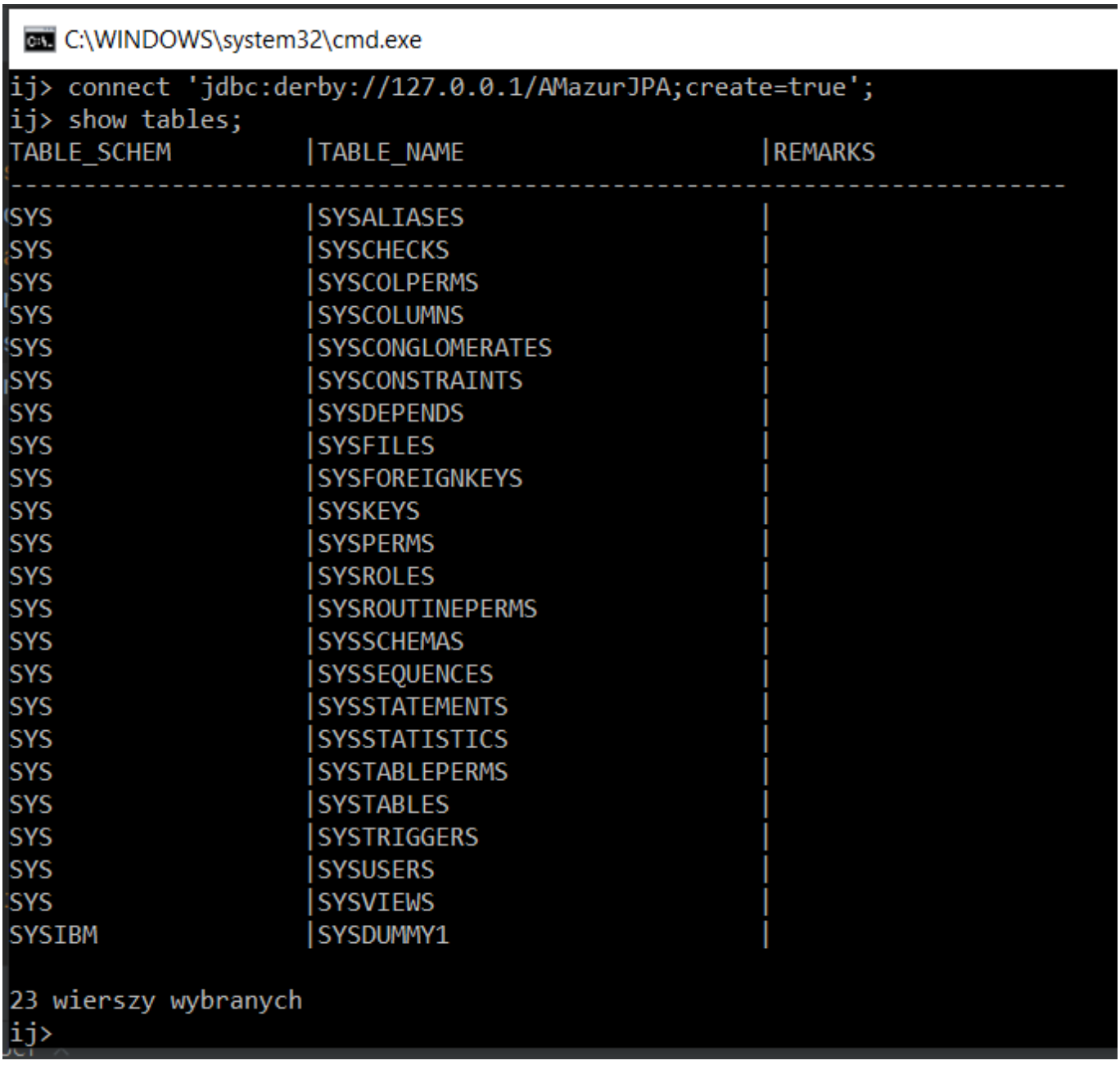
Aleksandra Mazur

Zadanie II. Basics

a) - h) Konfiguracja

Uruchomiono i podpięto się do serwera *Derby* oraz założono bazę `AMazurJPA`.

Polecenie *show tables* zwróciło poniższe dane.



i) Projekt Javowy

Utworzono projekt w Intellij o nazwie `AMazurJPAPractice`.

j), k) Klasa Product

Stworzono klasę `Product` z polami:

- public int ProductId
- public String ProductName
- public int UnitsOnStock

Uzupełniono w klasie elementy potrzebne do zmapowania jej do bazy danych.

Product.java

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int ProductId;
    public String ProductName;
    public int UnitsOnStock;

    public Product(String productName, int unitsOnStock) {
        ProductName = productName;
        UnitsOnStock = unitsOnStock;
    }

    public Product() {
        // for Hibernate
    }
}
```

l), m) Hibernate config

W pliku `hibernate.cfg.xml` uzupełniono potrzebne property.

hibernate.cfg.xml

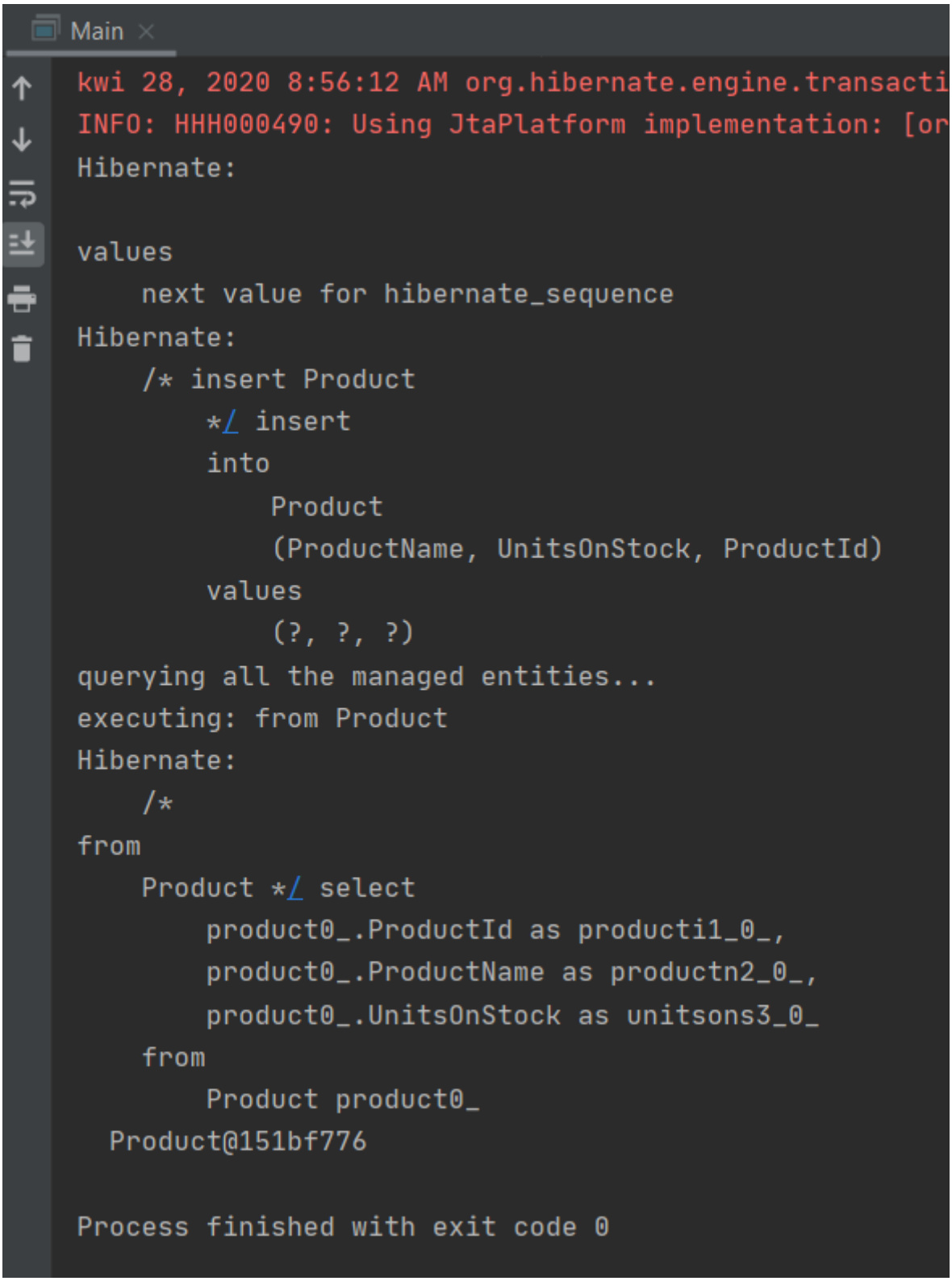
```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:derby://127.0.0.1/AMazurJPA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

Zadanie III. Dodanie produktu

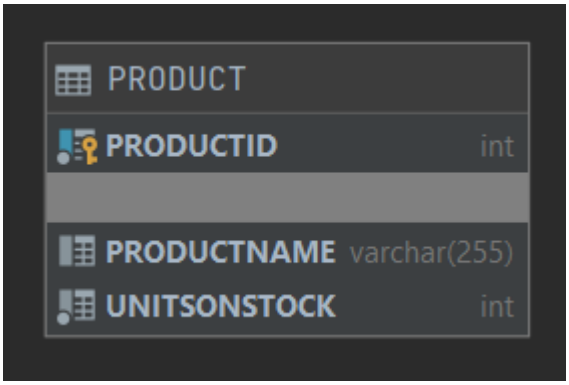
W klasie `Main` stworzono przykładowy produkt i zapisano go do bazy danych.

Main.java

```
7   import java.util.Map;
8
9   public class Main {
10      private static final SessionFactory ourSessionFactory;
11
12      static {
13          try {
14              Configuration configuration = new Configuration();
15              configuration.configure();
16
17              ourSessionFactory = configuration.buildSessionFactory();
18          } catch (Throwable ex) {
19              throw new ExceptionInInitializerError(ex);
20          }
21      }
22
23      public static Session getSession() throws HibernateException {
24          return ourSessionFactory.openSession();
25      }
26
27      public static void main(final String[] args) throws Exception {
28          Product product = new Product( productName: "Book",  unitsOnStock: 5);
29          final Session session = getSession();
30          Transaction transaction = session.beginTransaction();
31          session.save(product);
32          transaction.commit();
33          try {
34              System.out.println("querying all the managed entities...");
35              final Metamodel metamodel = session.getSessionFactory().getMetamodel();
36              for (EntityType<?> entityType : metamodel.getEntities()) {
37                  final String entityName = entityType.getName();
38                  final Query query = session.createQuery( s: "from " + entityName);
39                  System.out.println("executing: " + query.getQueryString());
40                  for (Object o : query.list()) {
41                      System.out.println("  " + o);
42                  }
43              }
44          } finally {
45              session.close();
46          }
47      }
48  }
```


Main
kwi 28, 2020 8:56:12 AM org.hibernate.engine.transacti
INFO: HHH000490: Using JtaPlatform implementation: [or
Hibernate:
values
next value for hibernate_sequence
Hibernate:
/* insert Product
*/ insert
into
Product
(ProductName, UnitsOnStock, 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_.UnitsOnStock as unitsons3_0_
from
Product product0_
Product@151bf776
Process finished with exit code 0

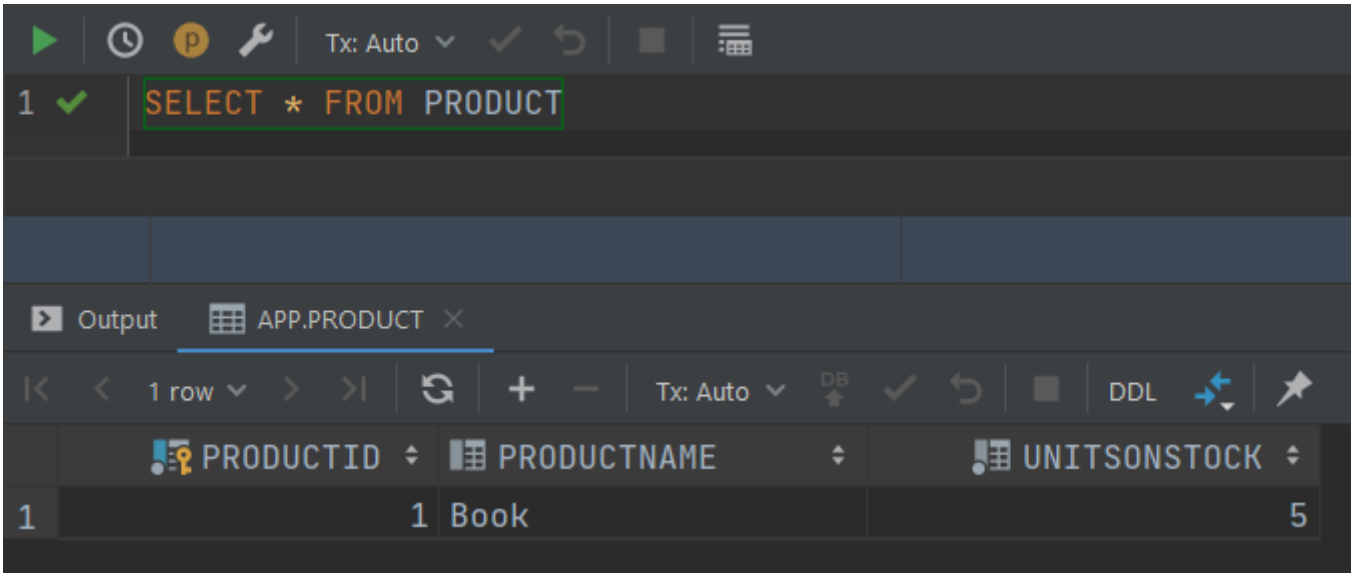
Schemat w bazie danych wygląda następująco:



Skrypt tworzący tabelę `Product`:

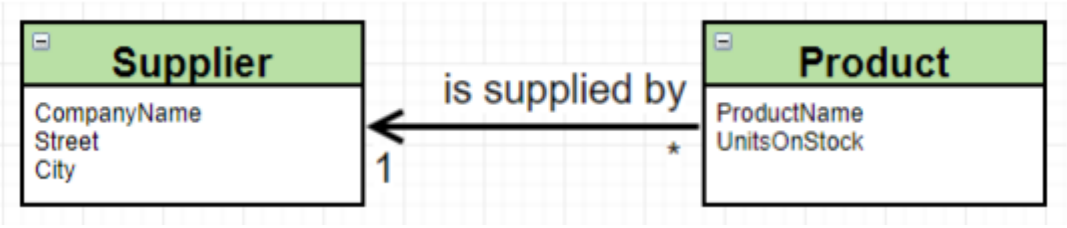
```
create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000081-120fc126-0171-bf75-6e31-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null
);
```

Jak widać produkt dodał się poprawnie:



Zadanie IV. Klasa Supplier

Zmodyfikowano model wprowadzając pojęcie dostawcy jak poniżej.



Klasa `Supplier` zawiera pola:

- public int SupplierId
- public String CompanyName
- public String Street
- public String City

Main.java Supplier.java

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int SupplierId;
    public String CompanyName;
    public String Street;
    public String City;

    public Supplier(String companyName, String street, String city) {
        CompanyName = companyName;
        Street = street;
        City = city;
    }

    public Supplier() {
    }
}
```

Zmodyfikowano klasę `Product`, dodając do niej pole `Supplier` i funkcję `setSupplier(Supplier supplier)` przypisującą danego dostawcę do produktu.

Main.java | Supplier.java | Product.java

```java
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int ProductId;
    public String ProductName;
    public int UnitsOnStock;

    @ManyToOne
    public Supplier supplier;

    public Product(String productName, int unitsOnStock) {
        ProductName = productName;
        UnitsOnStock = unitsOnStock;
    }

    public Product() {
        // for Hibernate
    }

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

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

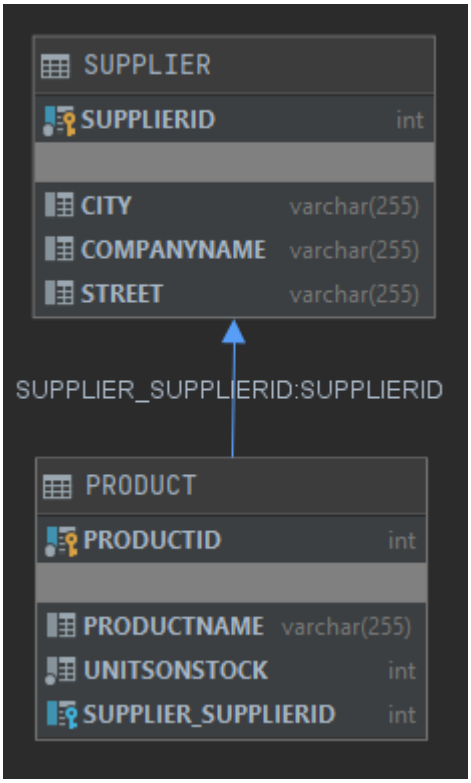
Do pliku `hibernate.cfg.xml` dodano Suppliera.

```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.url">jdbc:derby://127.0.0.1/AMazurJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
    </session-factory>
</hibernate-configuration>
```

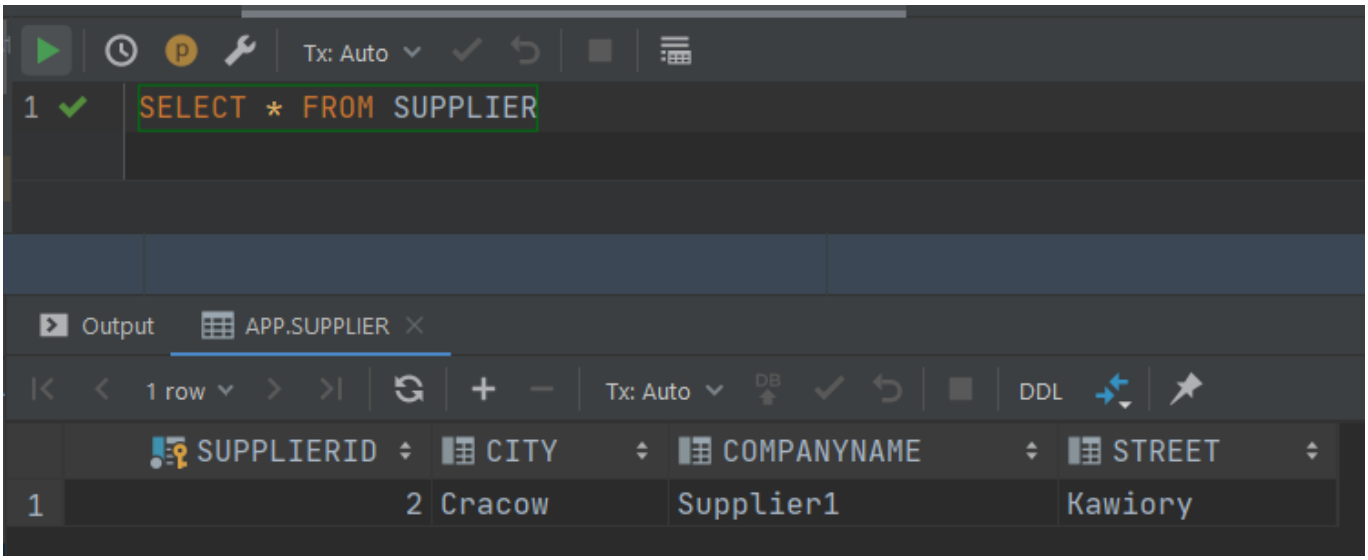
Utworzono nowego dostawcę i przypisano go do wcześniej dodanego produktu.

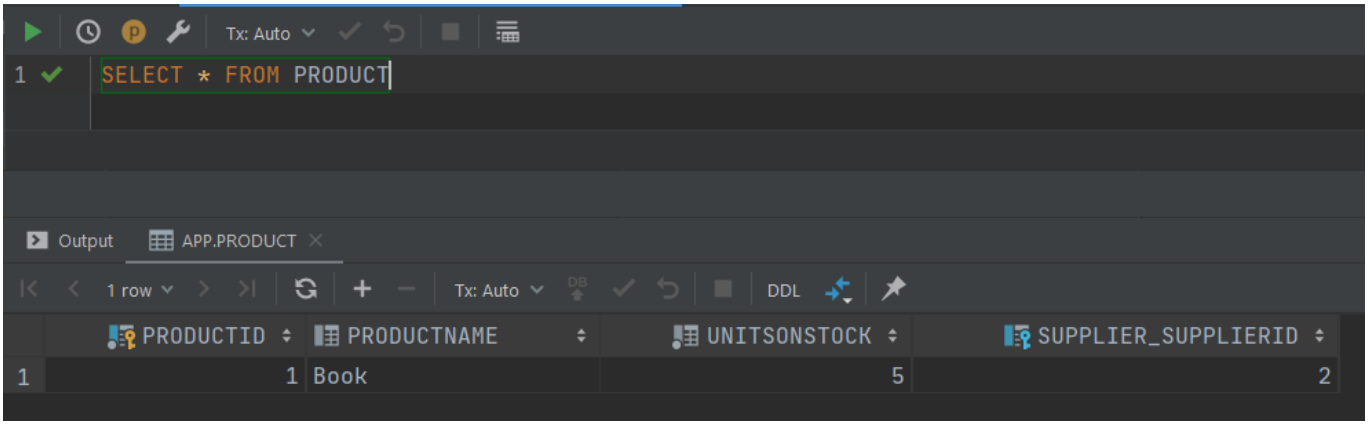
```java
    public static void main(final String[] args) throws Exception {
        final Session session = getSession();
        Transaction transaction = session.beginTransaction();
        Supplier supplier = new Supplier( companyName: "Supplier1",  street: "Kawiory",  city: "Cracow");
        Product foundProduct = session.get(Product.class,  serializable: 1);
        foundProduct.setSupplier(supplier);
        session.save(supplier);
        session.save(foundProduct);
        transaction.commit();

        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();
        }
    }
}
```

Schemat bazy danych wygląda następująco:



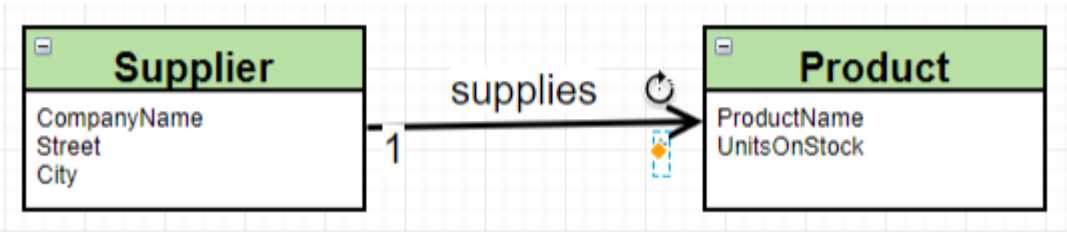
Jak widać dane dodały się poprawnie.





Zadanie V. Odwrócona relacja Supplier - Product

Odwrócono relację zgodnie z poniższym schematem.



a) Wariant z tabelą łącznikową

Usunięto z klasy `Product` wcześniej dodane pole `Supplier`.

```java
import javax.persistence.*;

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int ProductId;
    public String ProductName;
    public int UnitsOnStock;


    public Product(String productName, int unitsOnStock) {
        ProductName = productName;
        UnitsOnStock = unitsOnStock;
    }

    public Product() {
        // for Hibernate
    }

}
```

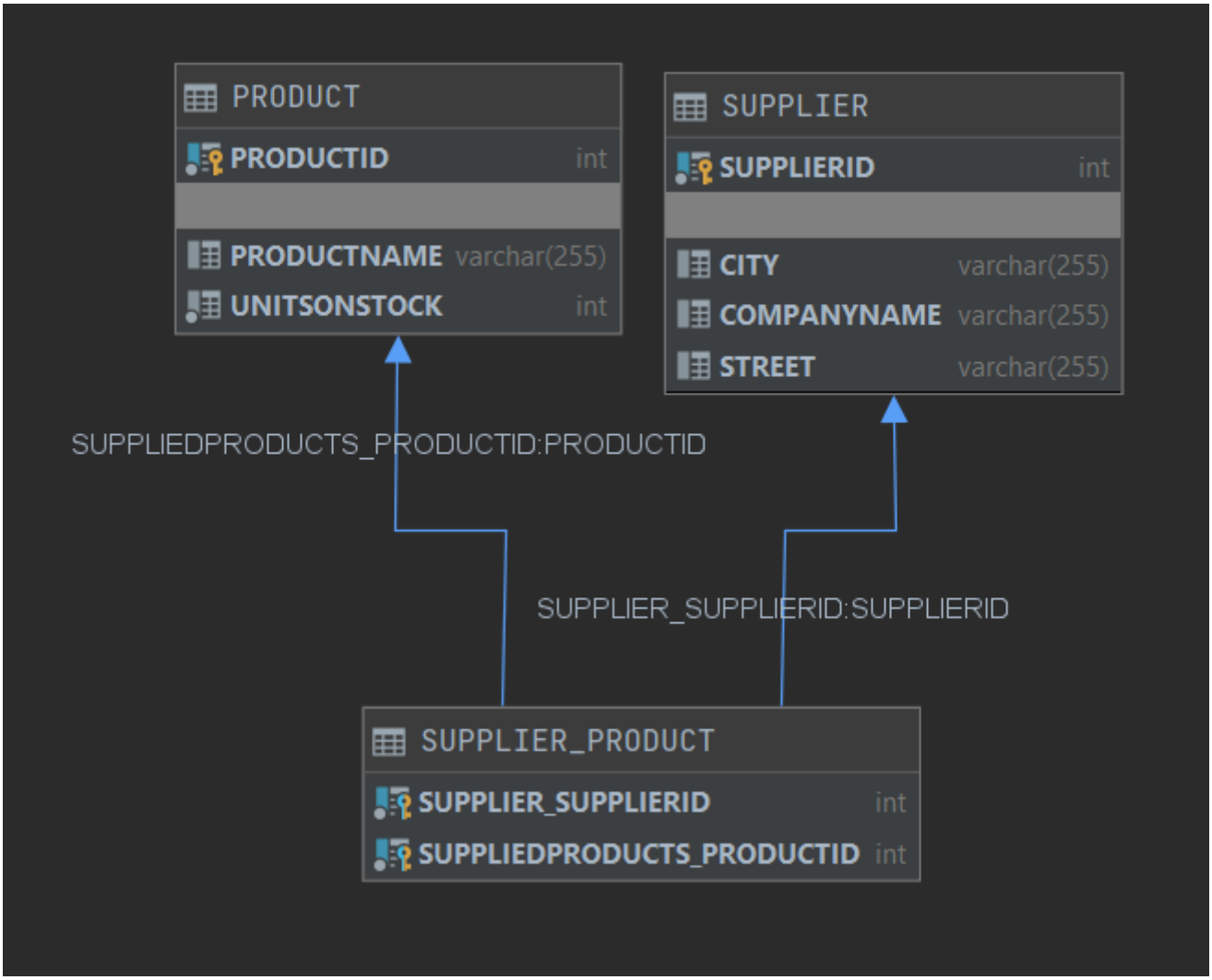
Do klasy `Supplier` dodano zbiór produktów i metodę `addProduct(Product product)` dodającą dany produkt do zbioru produktów dostarczanych przez danego dostawcę.

```java
import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int SupplierId;
    public String CompanyName;
    public String Street;
    public String City;

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


    public Supplier(String companyName, String street, String city) {
        CompanyName = companyName;
        Street = street;
        City = city;
    }

    public Supplier() {
    }

    public Supplier(String companyName, String street, String city, Product product) {
        CompanyName = companyName;
        Street = street;
        City = city;
        this.suppliedProducts.add(product);
    }

    public void addProduct(Product product){
        this.suppliedProducts.add(product);
    }
}
```

Dodano kilka produktów i dostawcę, a następnie przypisano utworzone produkty do zbioru produktów dostarczanych przez dostawcę.

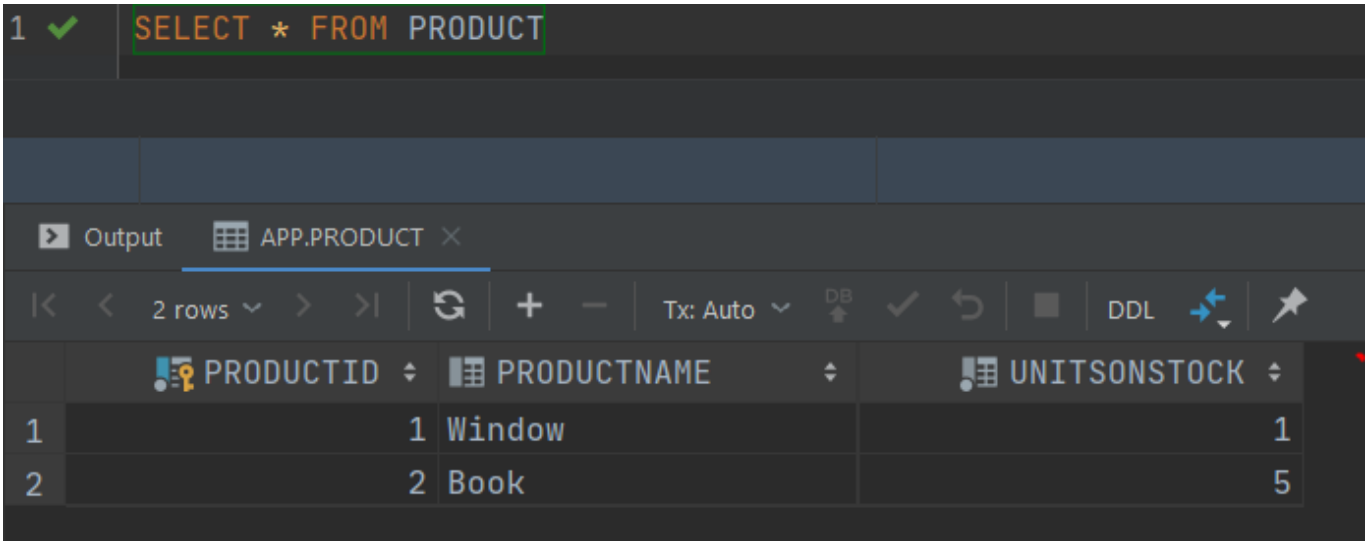
```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction transaction = session.beginTransaction();
    Supplier supplier = new Supplier( companyName: "Supplier1",  street: "Aleje",  city: "Warsaw");
    Product product = new Product( productName: "Window",  unitsOnStock: 1);
    Product product2 = new Product( productName: "Book",  unitsOnStock: 5);
    supplier.addProduct(product);
    supplier.addProduct(product2);
    session.save(product);
    session.save(product2);
    session.save(supplier);
    transaction.commit();

    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();
    }
}
```

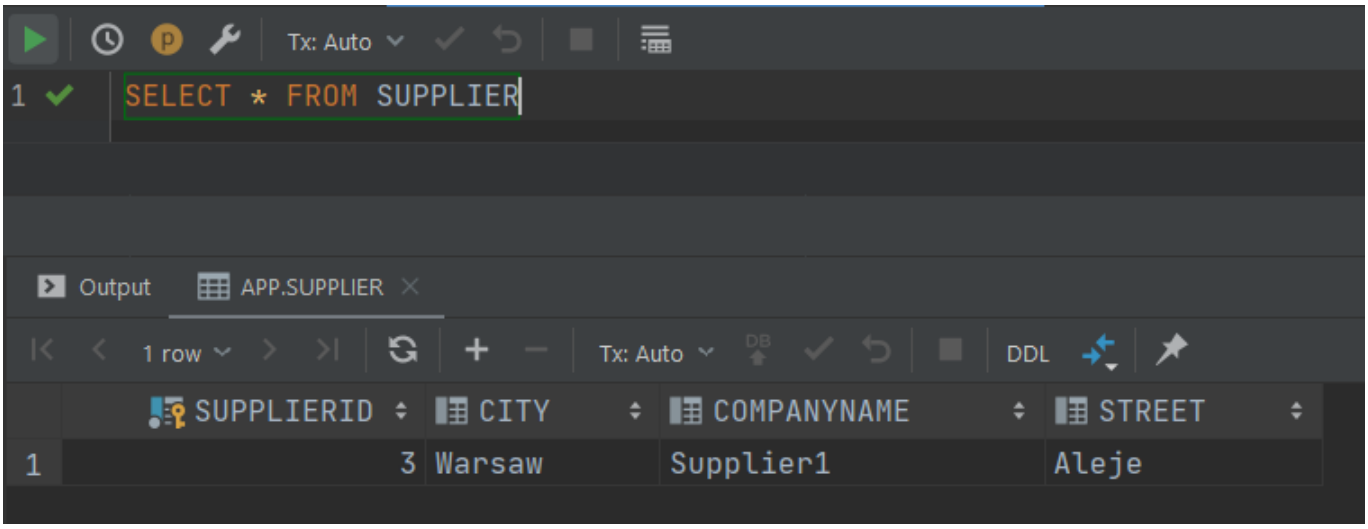
Schemat bazy danych wygląda następująco:

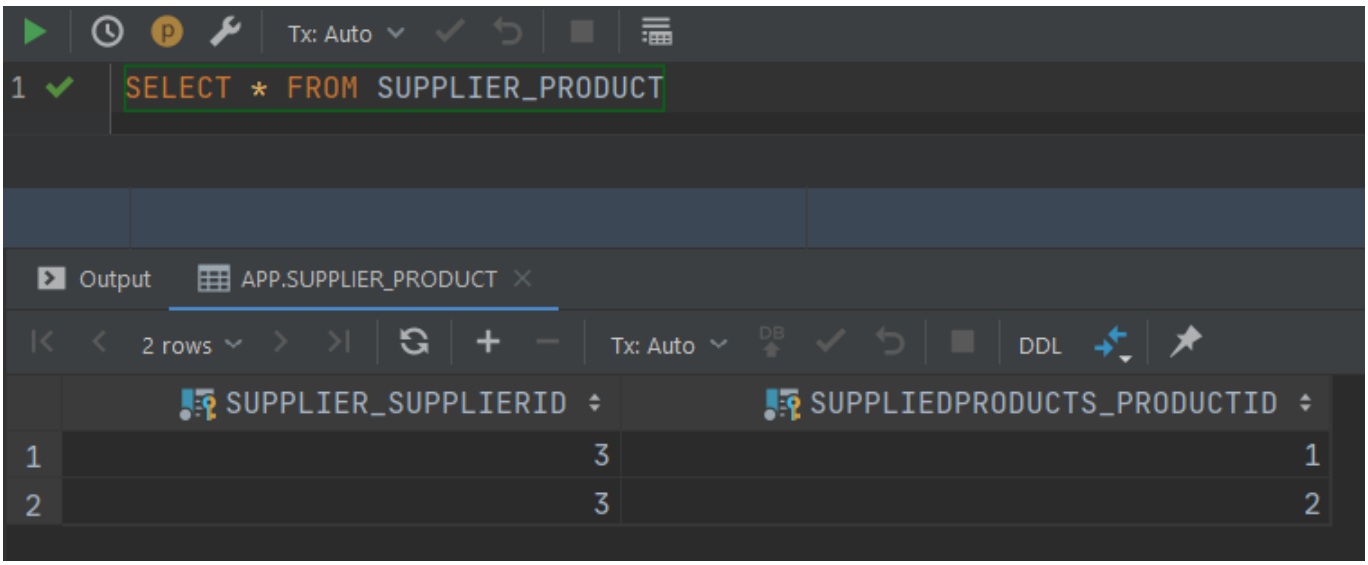


Jak widać powyżej, dodała się tabela łącznikowa `SUPPLIER_PRODUCT`.

Dodane dane:





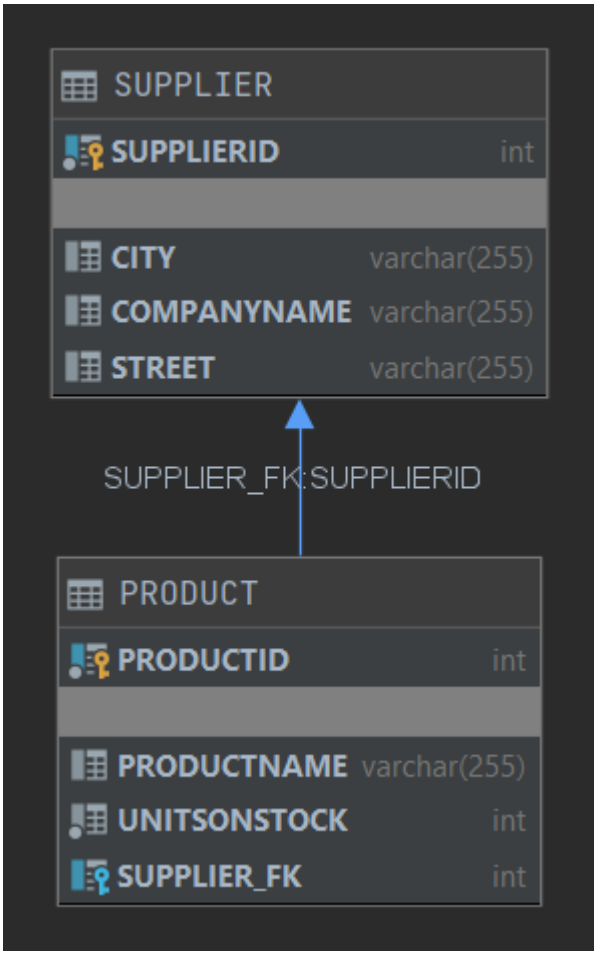


b) Wariant bez tabeli łącznikowej

W klasie `Supplier` nad zbiorem produktów dopisano *@JoinColumn(name="Supplier_FK")*.

```java
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int SupplierId;
    public String CompanyName;
    public String Street;
    public String City;

    @OneToMany
    @JoinColumn(name="Supplier_FK")
    public Set<Product> suppliedProducts = new HashSet<>();
```

Schemat bazy danych wygląda jak poniżej.



Jak widać nie dodała się tabela łącznikowa. Zamiast niej w tabeli `Product` jest pole `SUPPLIER_FK`.

Skrypt generujący tabele:

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000001-9ac8804c-0171-c07e-3180-ffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000000-ea5bc046-0171-c07e-3180-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    SUPPLIER_FK  INTEGER
        constraint FKVE96QACVSR1A50RGWL94ENRU
            references SUPPLIER
);

create index "SQL0000000003-e95a01d1-0171-c07e-3180-ffffc71c8945"
    on PRODUCT (SUPPLIER_FK);
```

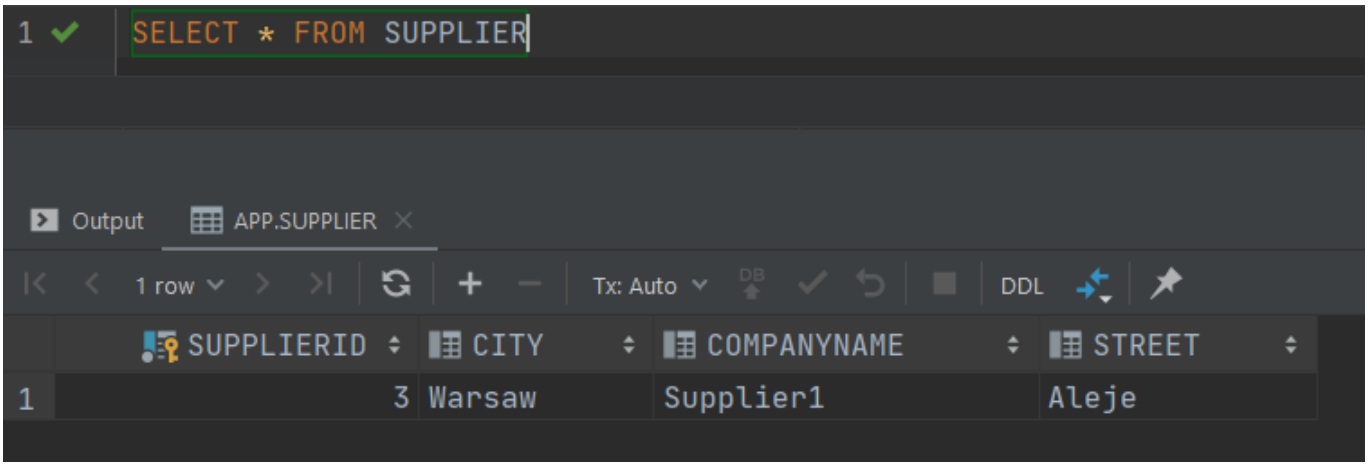
Dane dodane do bazy:

```
1 ✔  SELECT * FROM PRODUCT
```

Output | APP.PRODUCT

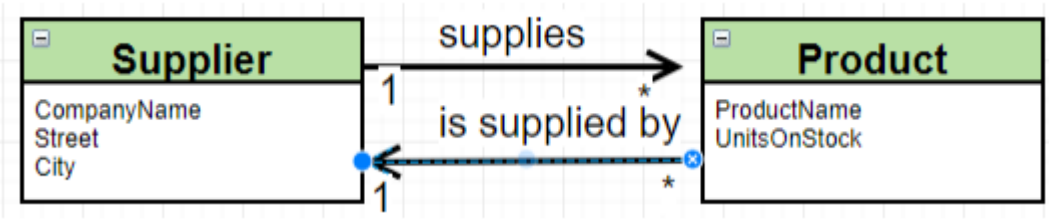
2 rows | Tx: Auto | DDL

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK
1	1	Window	1	3
2	2	Book	5	3


1
SELECT * FROM SUPPLIER
Output
APP.SUPPLIER
1 row
Tx: Auto
DDL
SUPPLIERID
CITY
COMPANYNAME
STREET
1
3
Warsaw
Supplier1
Aleje

Zadanie VI. Relacja dwustronna

Zamodelowano relację dwustronną jak poniżej.



Metody dodane/zmodyfikowane w klasie `Supplier`:

- `suppliesProduct(Product product)` zwracająca prawdę, jeżeli dostawca ma dany produkt w zbiorze dostarczanych produktów i fałsz w przeciwnym przypadku.
- `addProduct(Product product)` dodająca produkt do zbioru dostarczanych produktów i przypisująca dostawcę produktowi

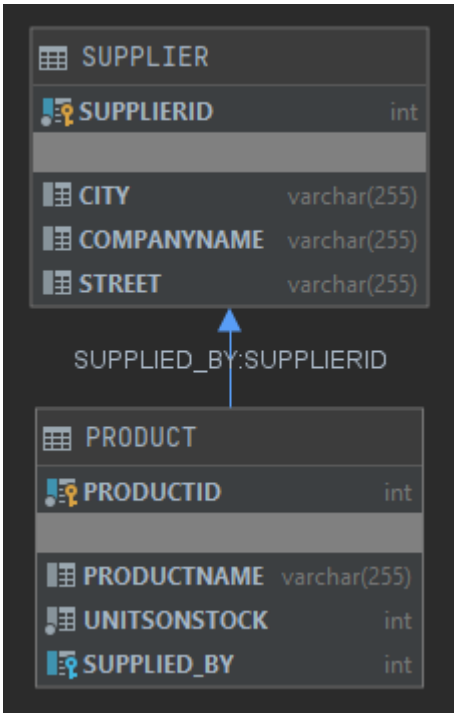
```java
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int SupplierId;
    public String CompanyName;
    public String Street;
    public String City;

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


    public Supplier(String companyName, String street, String city) {
        CompanyName = companyName;
        Street = street;
        City = city;
    }

    public Supplier() {
    }

    public Supplier(String companyName, String street, String city, Product product) {
        CompanyName = companyName;
        Street = street;
        City = city;
        this.suppliedProducts.add(product);
    }

    public void addProduct(Product product){
        this.suppliedProducts.add(product);
        product.setSupplier(this);
    }

    public boolean suppliesProduct(Product product){
        return suppliedProducts.contains(product);
    }
```

Klasa `Product`:

Dodano do klasy pole `Supplier` i metodę `setSupplier(Supplier supplier)` przypisującą dostawcę do produktu oraz dodającą produkt do zbioru produktów dostarczanych przez dostawcę, jeżeli nie należy on jeszcze do tego zbioru.

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int ProductId;
    public String ProductName;
    public int UnitsOnStock;

    @ManyToOne
    @JoinColumn(name="SUPPLIED_BY")
    public Supplier supplier;


    public Product(String productName, int unitsOnStock) {
        ProductName = productName;
        UnitsOnStock = unitsOnStock;
    }

    public Product() {
        // for Hibernate
    }

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

    public void setSupplier(Supplier supplier){
        this.supplier = supplier;
        if (!supplier.suppliesProduct(this)){
            supplier.addProduct(this);
        }
    }
```

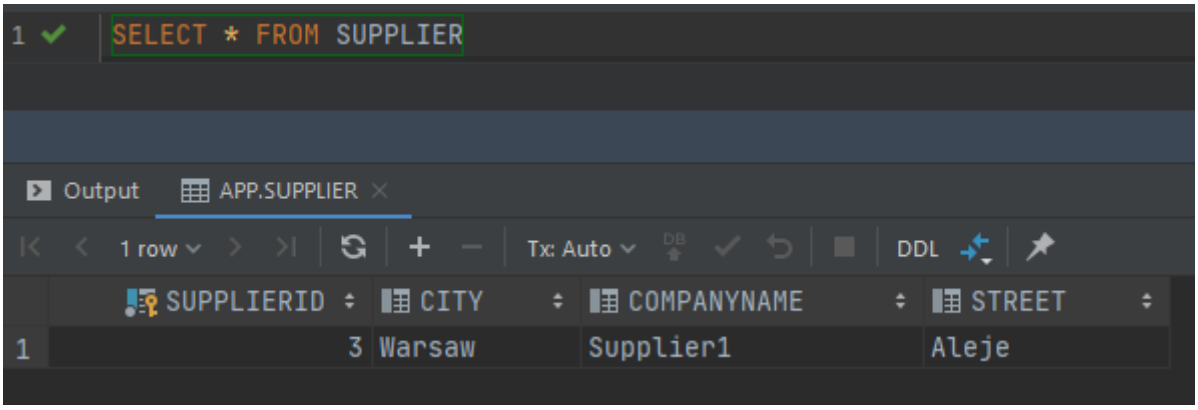
Schemat bazy danych wygląda następująco.

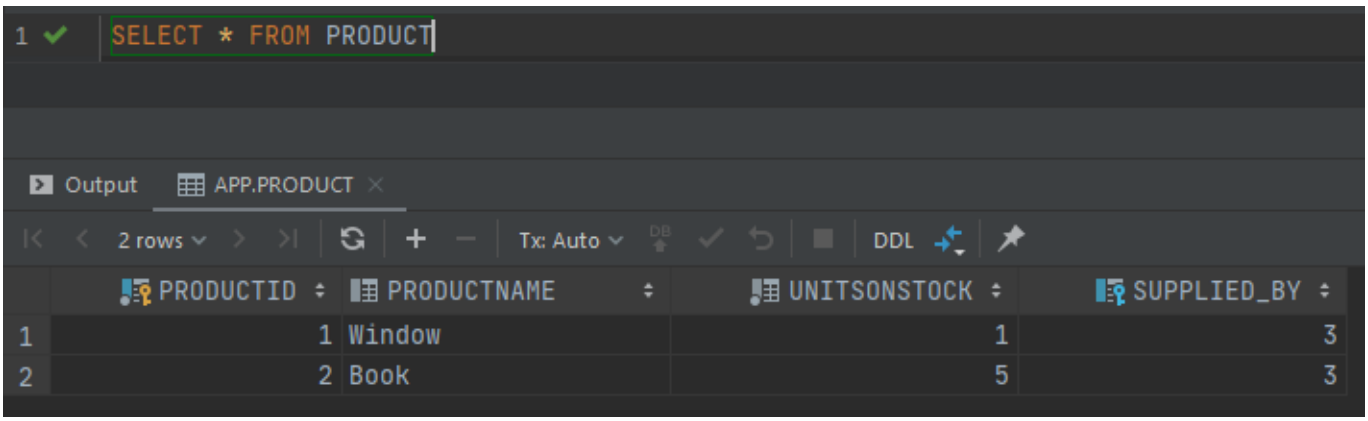


Skrypt tworzący tabele:

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000005-55410238-0171-c07e-3180-ffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000004-a1f24232-0171-c07e-3180-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create index "SQL0000000006-0128423f-0171-c07e-3180-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

Dodane dane:


1
SELECT * FROM SUPPLIER
Output
APP.SUPPLIER
1 row
Tx: Auto
DDL
SUPPLIERID
CITY
COMPANYNAME
STREET
1
3
Warsaw
Supplier1
Aleje


1
SELECT * FROM PRODUCT
Output
APP.PRODUCT
2 rows
Tx: Auto
DDL
PRODUCTID
PRODUCTNAME
UNITSONSTOCK
SUPPLIED_BY
1
1
Window
1
3
2
2
Book
5
3

Zadanie VII. Klasa Category

a) Modyfikacja modelu

Dodano klasę `Category` z polami:

- private int CategoryId
- private String Name
- private List <Product> Products

Klasa ta posiada metodę `addProduct(Product product)`, która dodaje dany produkt do listy produktów bieżącej kategorii oraz przypisuje kategorię produktowi.

```java
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CategoryId;
    private String Name;

    @OneToMany
    @JoinColumn(name="CATEGORY")
    private List<Product> Products = new ArrayList<>();


    public Category(String name) { Name = name; }

    public Category() {}

    public String getName() { return Name; }

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

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

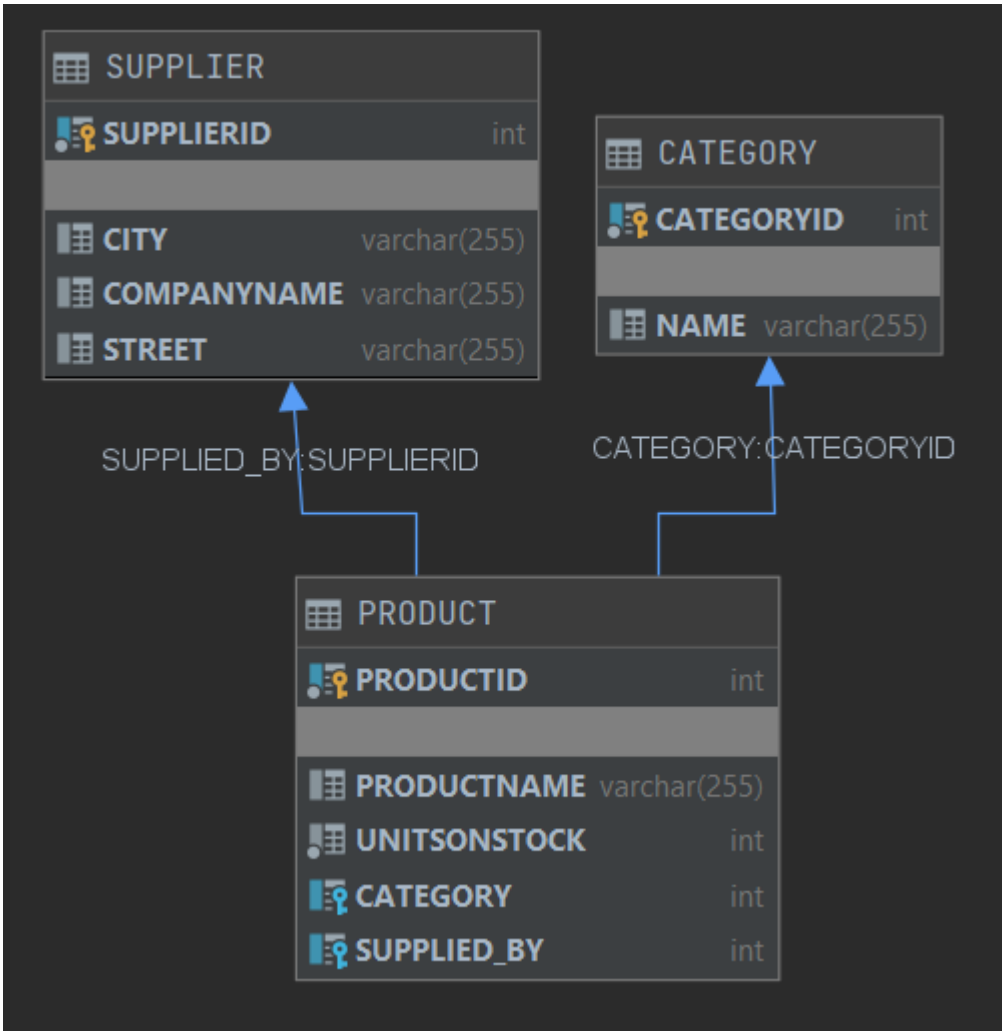
Do pliku `hibernate.cfg.xml` dodano Category.

```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.url">jdbc:derby://127.0.0.1/AMazurJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">create</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
        <mapping class="Category"></mapping>
    </session-factory>
</hibernate-configuration>
```

Zmodyfikowano klasę `Product`, dodając wskazanie na kategorię, do której należy oraz metodę `setCategory(Category category)`, przypisującą daną kategorię produktowi i dodającą produkt do listy produktów kategorii, jeśli jeszcze nie został tam dodany. Zmieniono również dostępność atrybutów we wszystkich klasach na prywatne.

```java
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductId;
    private String ProductName;
    private int UnitsOnStock;

    @ManyToOne
    @JoinColumn(name="SUPPLIED_BY")
    private Supplier supplier;

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

    public Product(String productName, int unitsOnStock) {...}

    public Product() {}

    public Product(String productName, int unitsOnStock, Supplier supplier) {...}

    public void setSupplier(Supplier supplier){...}

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

    public void setCategory(Category category){
        this.category = category;
        if (!category.getProducts().contains(this))
            category.addProduct(this);
    }
```

Schemat bazy wygląda następująco:



Skrypt generujący tabele:

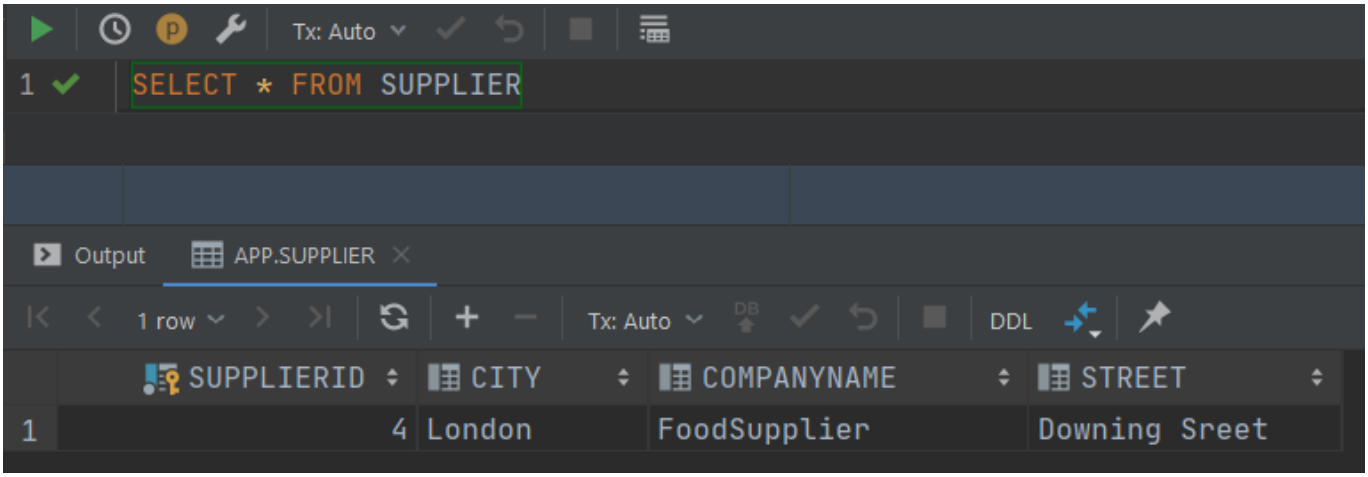
```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000007-c6970388-0171-c07e-3180-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000009-da248395-0171-c07e-3180-ffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000008-f4ca438f-0171-c07e-3180-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create index "SQL0000000010-b86e839c-0171-c07e-3180-ffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000011-1c0c03a0-0171-c07e-3180-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

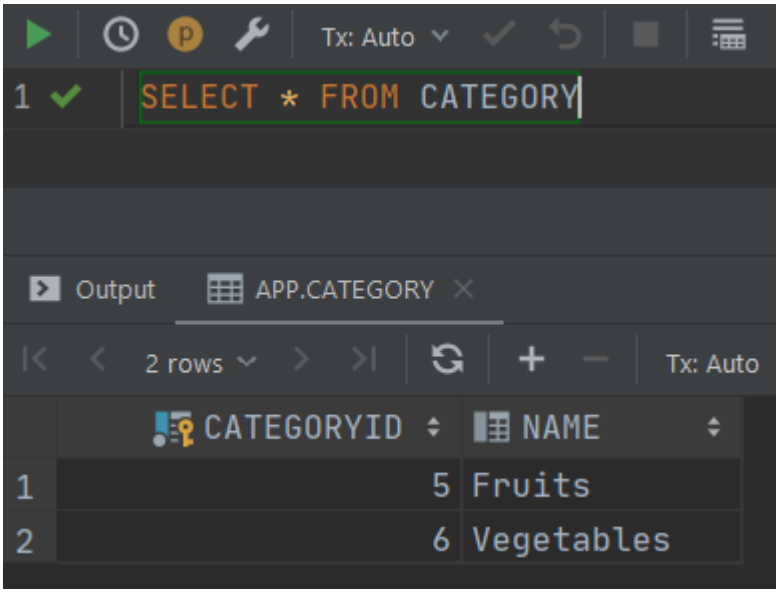
b), c) Dodanie danych

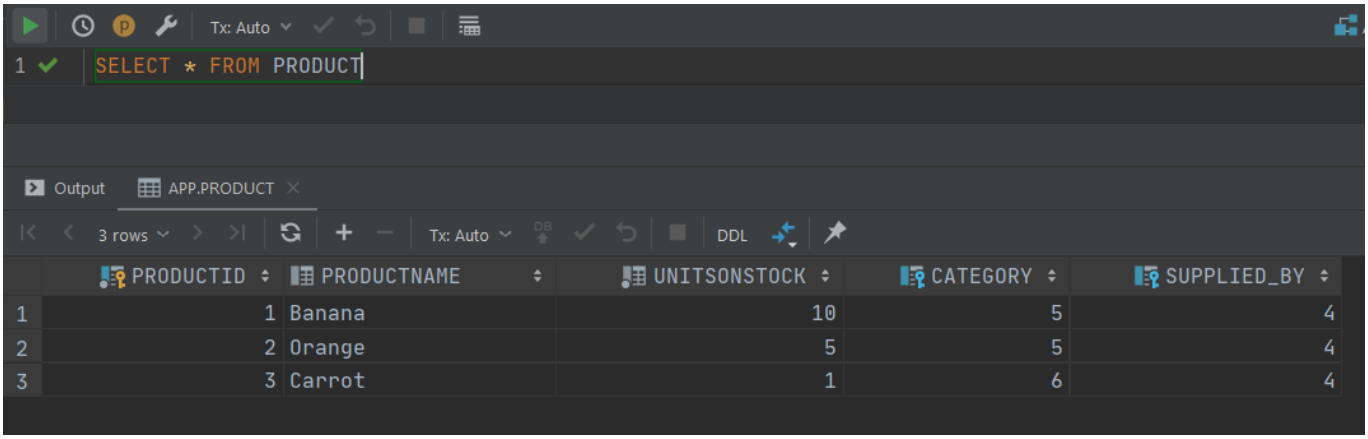
W `Mainie` dodano dostawcę, kilka produktów i kategorii jak poniżej.

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

    try {

        Transaction transaction = session.beginTransaction();
        Supplier supplier = new Supplier( companyName: "FoodSupplier",  street: "Downing Sreet",  city: "London");
        Product product1 = new Product( productName: "Banana",  unitsOnStock: 10);
        Product product2 = new Product( productName: "Orange",  unitsOnStock: 5);
        Product product3 = new Product( productName: "Carrot",  unitsOnStock: 1);
        Category category1 = new Category( name: "Fruits");
        Category category2 = new Category( name: "Vegetables");

        supplier.addProduct(product1);
        supplier.addProduct(product2);
        supplier.addProduct(product3);

        category1.addProduct(product1);
        category1.addProduct(product2);
        category2.addProduct(product3);

        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(supplier);
        session.save(category1);
        session.save(category2);
        transaction.commit();

        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();
```

Po uruchomieniu powyższego kodu baza danych wygląda następująco:

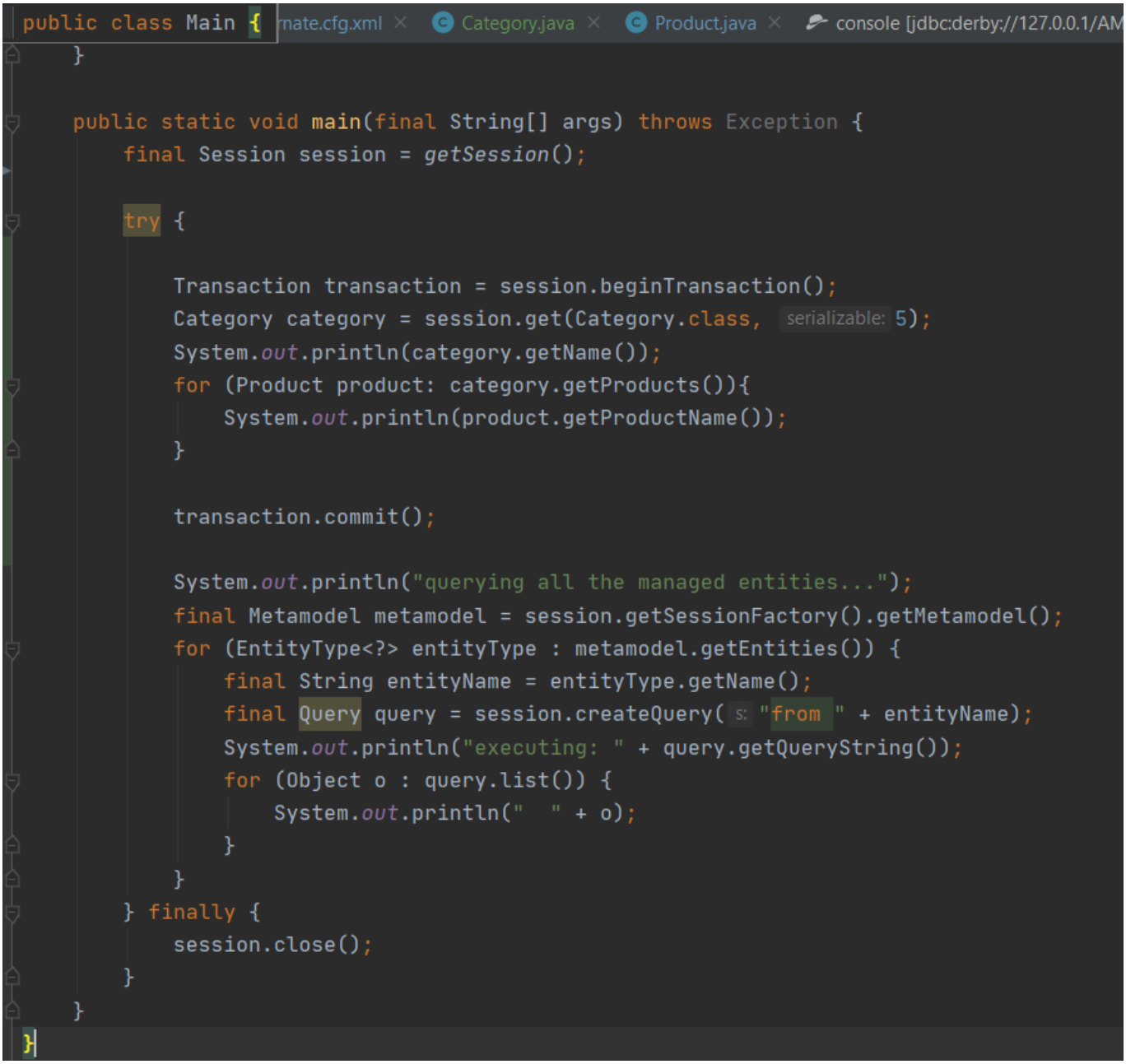






d) Wyciągnięcie danych z bazy

- Produkty z wybranej kategorii:



```
Fruits
Hibernate:
    select
        products0_.CATEGORY as category4_1_0_,
        products0_.ProductId as producti1_1_0_,
        products0_.ProductId as producti1_1_1_,
        products0_.ProductName as productn2_1_1_,
        products0_.UnitsOnStock as unitsons3_1_1_,
        products0_.CATEGORY as category4_1_1_,
        products0_.SUPPLIED_BY as supplied5_1_1_,
        supplier1_.SupplierId as supplier1_2_2_,
        supplier1_.City as city2_2_2_,
        supplier1_.CompanyName as companyn3_2_2_,
        supplier1_.Street as street4_2_2_
    from
        Product products0_
    left outer join
        Supplier supplier1_
            on products0_.SUPPLIED_BY=supplier1_.SupplierId
    where
        products0_.CATEGORY=?
Banana
Orange
```

- Kategoria, do której należy wybrany produkt:

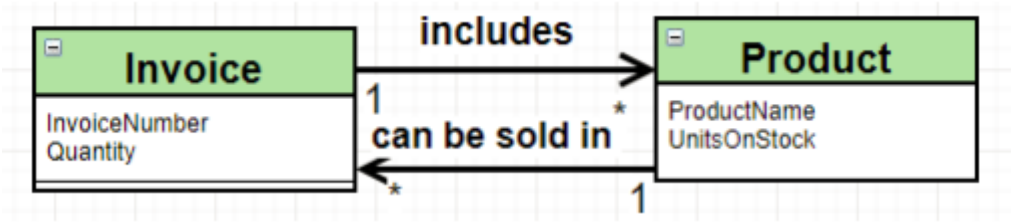
```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();

    try {

        Transaction transaction = session.beginTransaction();
        Product product = session.get(Product.class, serializable: 3);
        System.out.println(product.getProductName());
        System.out.println(product.getCategory().getName());

        transaction.commit();

        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();
    }
}
```

```
Hibernate:
    select
        product0_.ProductId as producti1_1_0_,
        product0_.ProductName as productn2_1_0_,
        product0_.UnitsOnStock as unitsons3_1_0_,
        product0_.CATEGORY as category4_1_0_,
        product0_.SUPPLIED_BY as supplied5_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=category1_.CategoryId
    left outer join
        Supplier supplier2_
            on product0_.SUPPLIED_BY=supplier2_.SupplierId
    where
        product0_.ProductId=?
Carrot
Vegetables
```

Zadanie VIII. Klasa Invoice (relacje wiele-do wielu)

a) Modyfikacja modelu

Zamodelowano relację wiele-do-wielu jak poniżej.



Stworzono klasę `Invoice` o atrybutach:

- private int InvoiceNumber
- private int Quantity
- private Set <Product> includesProducts

Dodano do niej między innymi metodę `addProduct(Product product)`, dodającą produkt do zbioru produktów faktury, fakturę do zbioru faktur danego produktu i zwiększającą ilość produktów na fakturze o 1.

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

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

    public Invoice(int quantity) { Quantity = quantity; }

    public Invoice() {}

    public void addProduct(Product product){
        includesProducts.add(product);
        product.getCanBeSoldIn().add(this);
        this.Quantity += 1;
    }

    public int getQuantity() { return Quantity; }

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

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

Do klasy `Product` dodano zbiór faktur, które zawierają bieżący produkt:

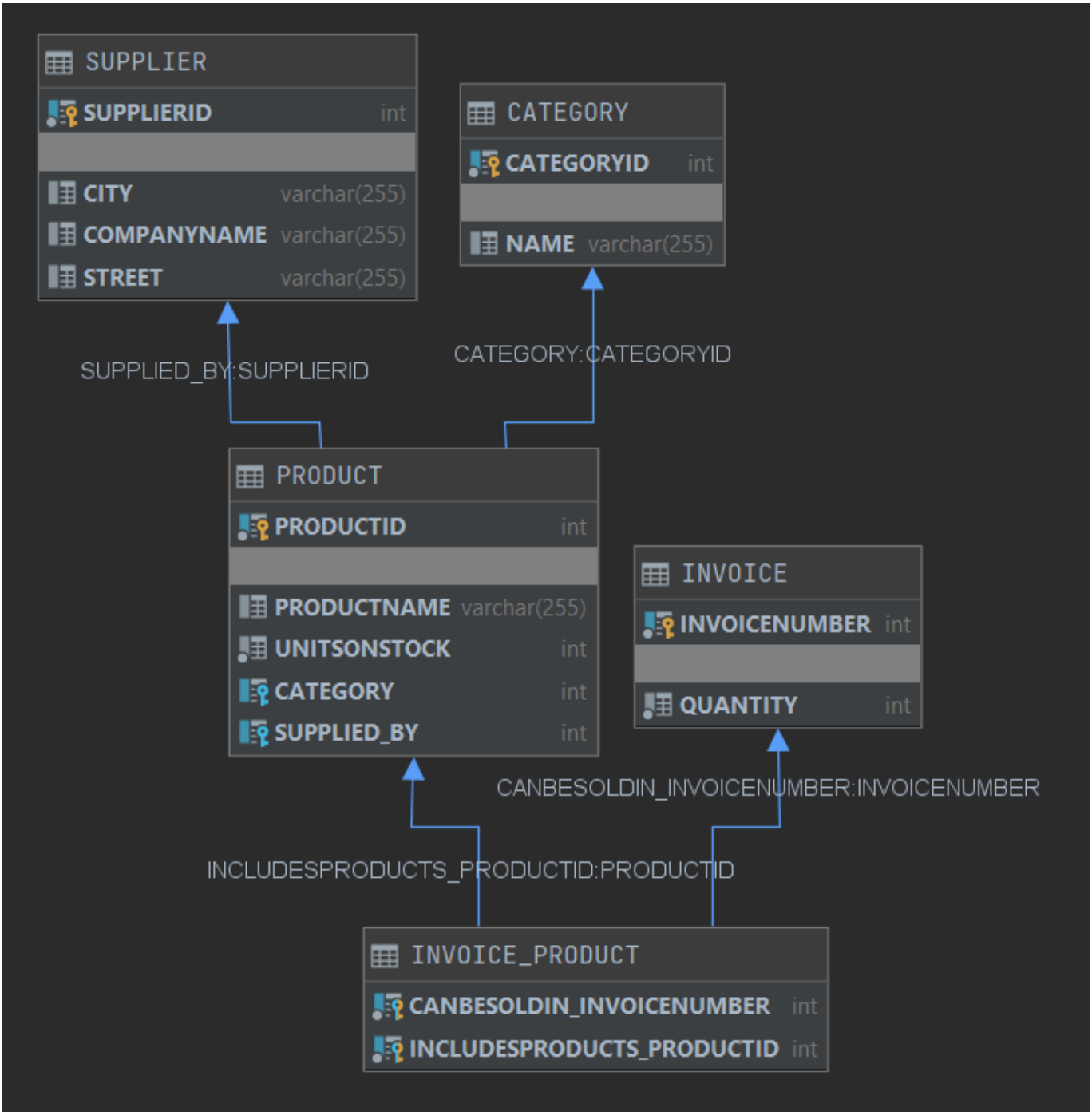
- private Set<Invoice> canBeSoldIn

```java
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductId;
    private String ProductName;
    private int UnitsOnStock;

    @ManyToOne
    @JoinColumn(name="SUPPLIED_BY")
    private Supplier supplier;

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

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

Do pliku `hibernate.cfg.xml` dodano Invoice.

```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.url">jdbc:derby://127.0.0.1/AMazurJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"></mapping>
        <mapping class="Supplier"></mapping>
        <mapping class="Category"></mapping>
        <mapping class="Invoice"></mapping>
    </session-factory>
</hibernate-configuration>
```

Schemat bazy dancyh wygląda następująco:



Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000018-cccb420a-0171-c670-619d-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000019-7fde0210-0171-c670-619d-ffffc71c8945"
            primary key,
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000022-194c4222-0171-c670-619d-ffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000021-a61e821c-0171-c670-619d-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000020-f2f9c216-0171-c670-619d-ffffc71c8945"
```

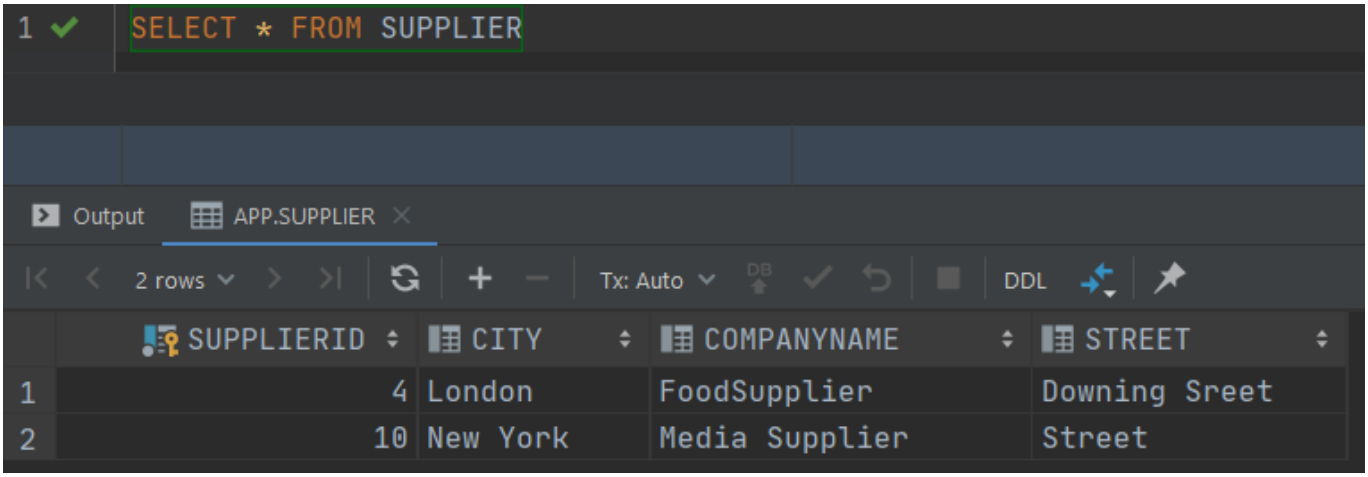
```
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000023-cc830228-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000024-2eac822c-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000025-90da0230-0171-c670-619d-ffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000026-f30b8234-0171-c670-619d-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

Dodanie obiektów do bazy

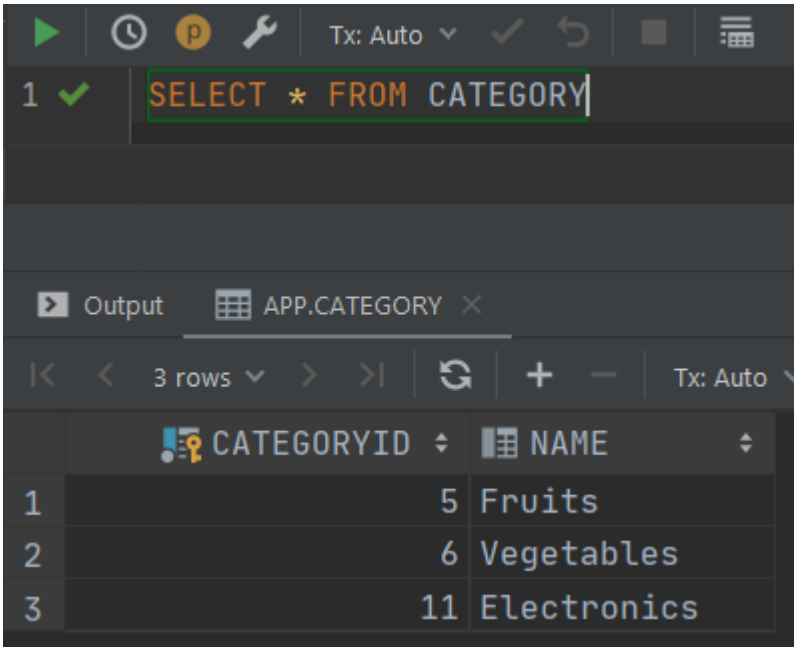
Dodano kilka produktów, faktur, dostawcę i kategorię.

```
Transaction transaction = session.beginTransaction();
Product product1 = new Product( productName: "TV", unitsOnStock: 2);
Product product2 = new Product( productName: "Computer", unitsOnStock: 1);
Product product3 = new Product( productName: "Telephone", unitsOnStock: 4);
Supplier supplier = new Supplier( companyName: "Media Supplier", street: "Street", city: "New York");
Category category = new Category( name: "Electronics");
Invoice invoice1 = new Invoice( quantity: 0);
Invoice invoice2 = new Invoice( quantity: 0);

supplier.addProduct(product1);
supplier.addProduct(product2);
supplier.addProduct(product3);
category.addProduct(product1);
category.addProduct(product2);
category.addProduct(product3);
invoice1.addProduct(product1);
invoice1.addProduct(product2);
invoice2.addProduct(product1);
invoice2.addProduct(product2);
invoice2.addProduct(product3);

session.save(product1);
session.save(product2);
session.save(product3);
session.save(supplier);
session.save(category);
session.save(invoice1);
session.save(invoice2);
transaction.commit();
```

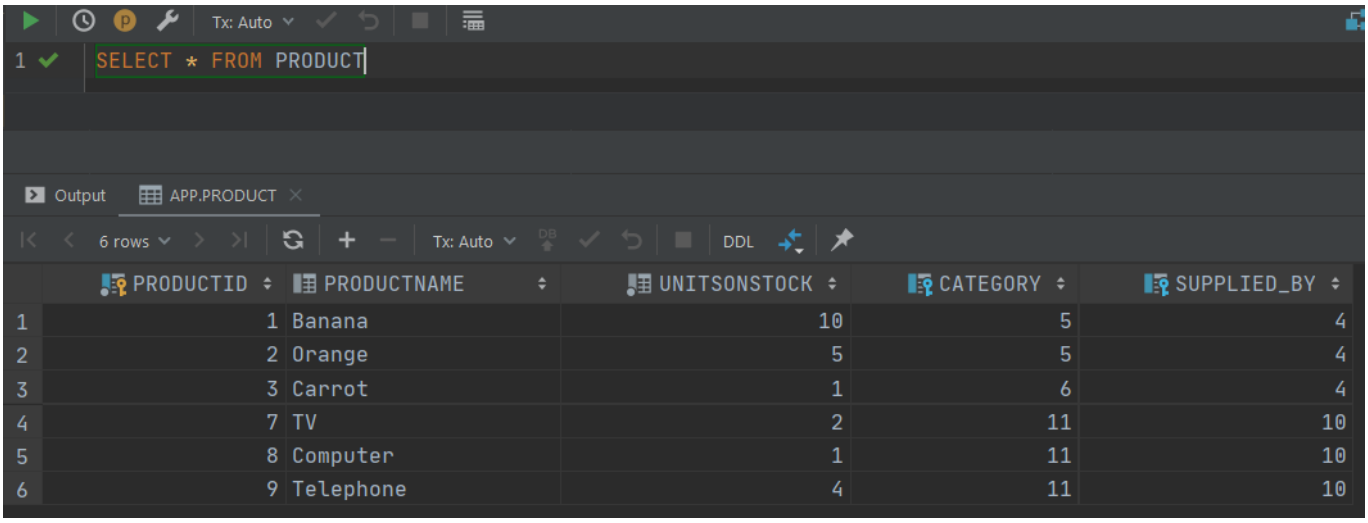
Zawartość bazy danych po uruchomieniu powyższego kodu:



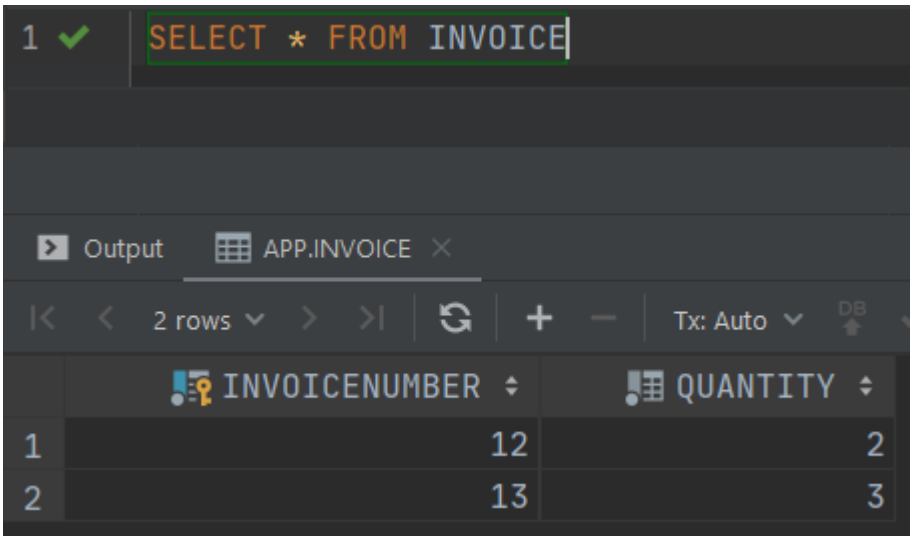
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	London	FoodSupplier	Downing Sreet
2	10	New York	Media Supplier	Street

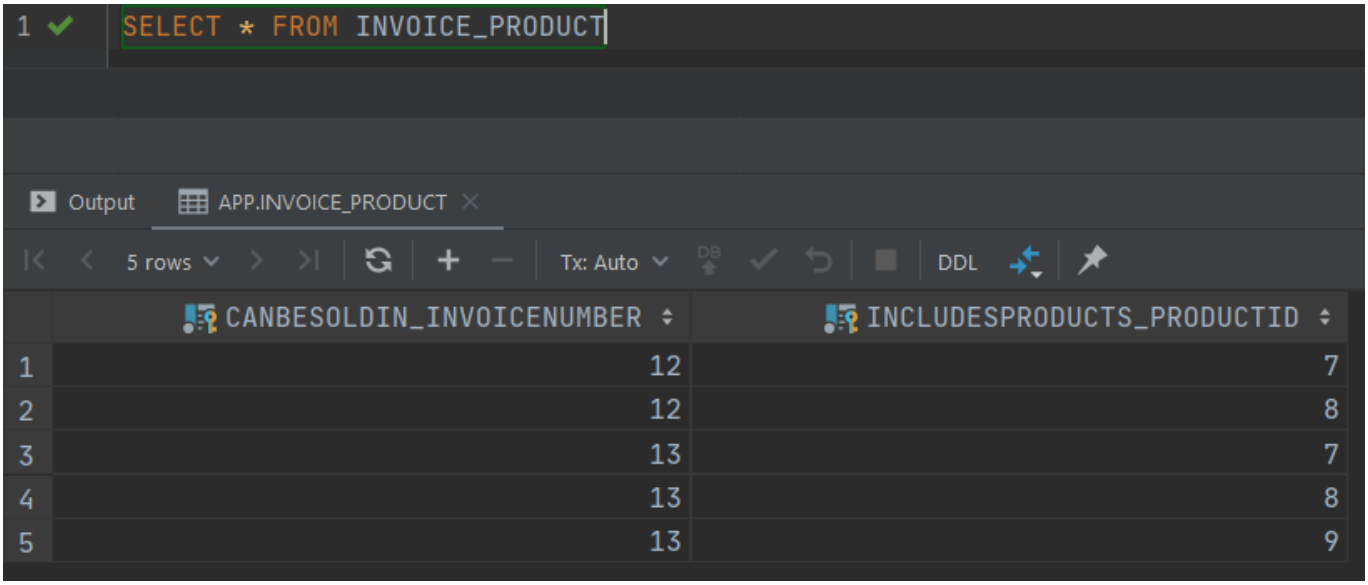


	CATEGORYID	NAME
1	5	Fruits
2	6	Vegetables
3	11	Electronics



	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	1	Banana	10	5	4
2	2	Orange	5	5	4
3	3	Carrot	1	6	4
4	7	TV	2	11	10
5	8	Computer	1	11	10
6	9	Telephone	4	11	10


1
SELECT * FROM INVOICE
Output
APP.INVOICE
2 rows
Tx: Auto
INVOICENUMBER
QUANTITY
1 12 2
2 13 3


1
SELECT * FROM INVOICE_PRODUCT
Output
APP.INVOICE_PRODUCT
5 rows
Tx: Auto
DDL
CANBESOLDIN_INVOICENUMBER
INCLUDESPRODUCTS_PRODUCTID
1 12 7
2 12 8
3 13 7
4 13 8
5 13 9

b) Wypisanie produktów sprzedanych w ramach wybranej faktury

```
Transaction transaction = session.beginTransaction();
Invoice invoice = session.get(Invoice.class, serializable: 12);
for(Product product: invoice.getIncludesProducts())
    System.out.println(product.getProductName());


transaction.commit();
```

```
Hibernate:
    select
        canbesoldi0_.includesProducts_ProductId as includes2_2_0_,
        canbesoldi0_.canBeSoldIn_InvoiceNumber as canbesol1_2_0_,
        invoice1_.InvoiceNumber as invoicen1_1_1_,
        invoice1_.Quantity as quantity2_1_1_
    from
        Invoice_Product canbesoldi0_
    inner join
        Invoice invoice1_
            on canbesoldi0_.canBeSoldIn_InvoiceNumber=invoice1_.InvoiceNumber
    where
        canbesoldi0_.includesProducts_ProductId=?
TV
Computer
```

c) Wypisanie faktur w ramach, których był sprzedany wybrany produkt

```
Transaction transaction = session.beginTransaction();
Product product = session.get(Product.class, serializable: 8);
for (Invoice invoice: product.getCanBeSoldIn())
    System.out.println("Invoice number: "+ invoice.getInvoiceNumber());

transaction.commit();
```

```
Hibernate:
    select
        product0_.ProductId as producti1_3_0_,
        product0_.ProductName as productn2_3_0_,
        product0_.UnitsOnStock as unitsons3_3_0_,
        product0_.CATEGORY as category4_3_0_,
        product0_.SUPPLIED_BY as supplied5_3_0_,
        canbesoldi1_.includesProducts_ProductId as includes2_2_1_,
        invoice2_.InvoiceNumber as canbesol1_2_1_,
        invoice2_.InvoiceNumber as invoicen1_1_2_,
        invoice2_.Quantity as quantity2_1_2_,
        category3_.CategoryId as category1_0_3_,
        category3_.Name as name2_0_3_,
        supplier4_.SupplierId as supplier1_4_4_,
        supplier4_.City as city2_4_4_,
        supplier4_.CompanyName as companyn3_4_4_,
        supplier4_.Street as street4_4_4_
    from
        Product product0_
    left outer join
        Invoice_Product canbesoldi1_
            on product0_.ProductId=canbesoldi1_.includesProducts_ProductId
    left outer join
        Invoice invoice2_
            on canbesoldi1_.canBeSoldIn_InvoiceNumber=invoice2_.InvoiceNumber
    left outer join
        Category category3_
            on product0_.CATEGORY=category3_.CategoryId
    left outer join
        Supplier supplier4_
            on product0_.SUPPLIED_BY=supplier4_.SupplierId
    where
        product0_.ProductId=?
Invoice number: 13
Invoice number: 12
```

Zadanie IX-X. JPA

Stworzono plik `persistence.xml` w folderze `META-INF` oraz nowego maina `JPAMain` jak poniżej:

persistence.xml JPAMain.java

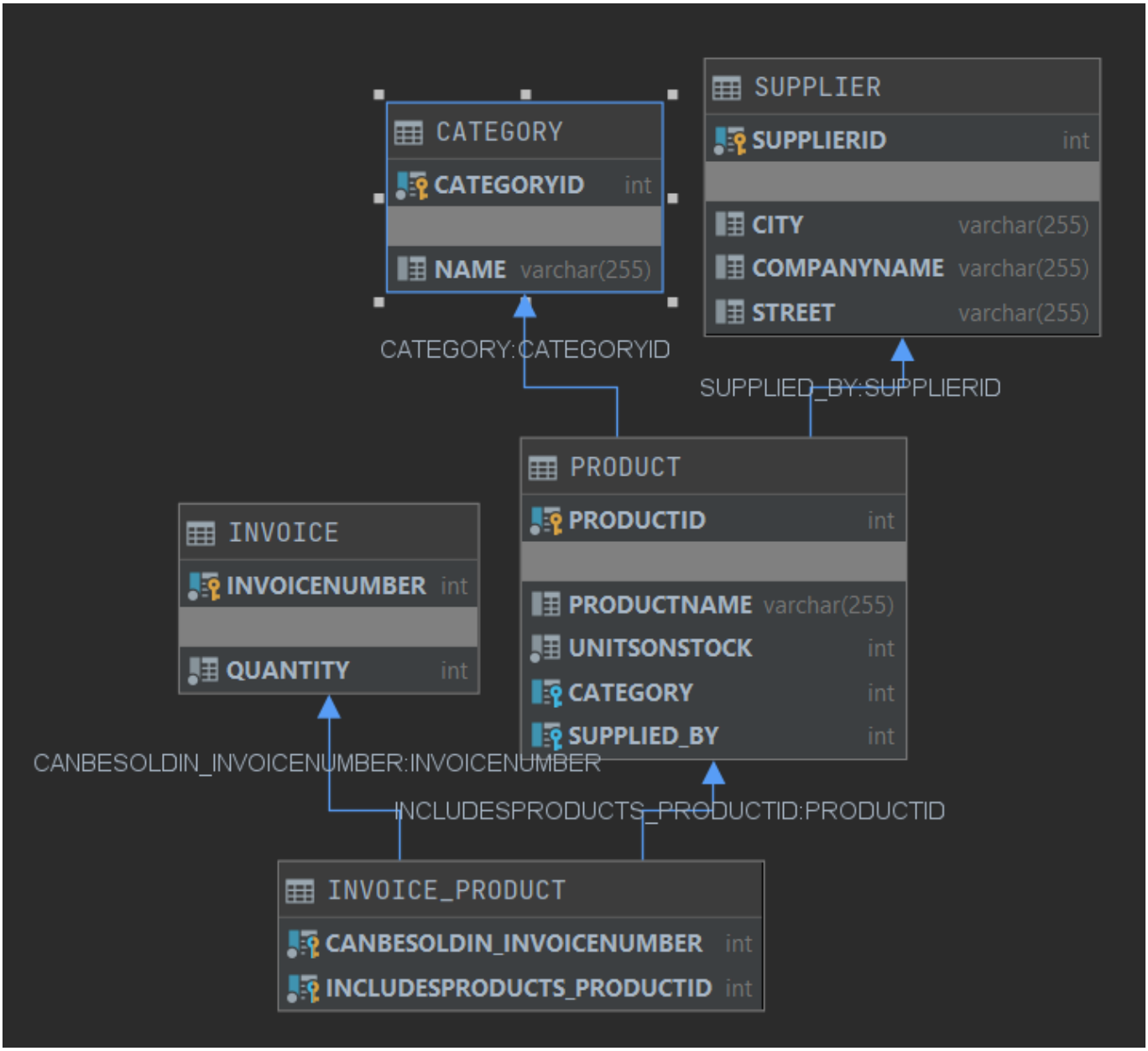
```xml
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
            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

persistence.xml × JPAMain.java ×

```
5
6   public class JPAMain {
7       private static EntityManagerFactory entityManagerFactory;
8
9       private static EntityManager getEntityManager(){
10          if (entityManagerFactory == null)
11              entityManagerFactory = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
12          return entityManagerFactory.createEntityManager();
13      }
14
15      public static void main(String argv[]){
16          EntityManager entityManager = getEntityManager();
17          EntityTransaction etx = entityManager.getTransaction();
18          etx.begin();
19
20          Supplier supplier = new Supplier( companyName: "Clothes Supplier",  street: "Street 2",  city: "Paris");
21          Product product1 = new Product( productName: "Jeans",  unitsOnStock: 6);
22          Product product2 = new Product( productName: "Skirt",  unitsOnStock: 10);
23          Product product3 = new Product( productName: "Shirt",  unitsOnStock: 20);
24          supplier.addProduct(product1);
25          supplier.addProduct(product2);
26          supplier.addProduct(product3);
27
28          entityManager.persist(product1);
29          entityManager.persist(product2);
30          entityManager.persist(product3);
31          entityManager.persist(supplier);
32
33          etx.commit();
34          entityManager.close();
35      }
36  }
```

Dodano do bazy dostawcę i kilka produktów.

Schemat bazy wygląda następująco:

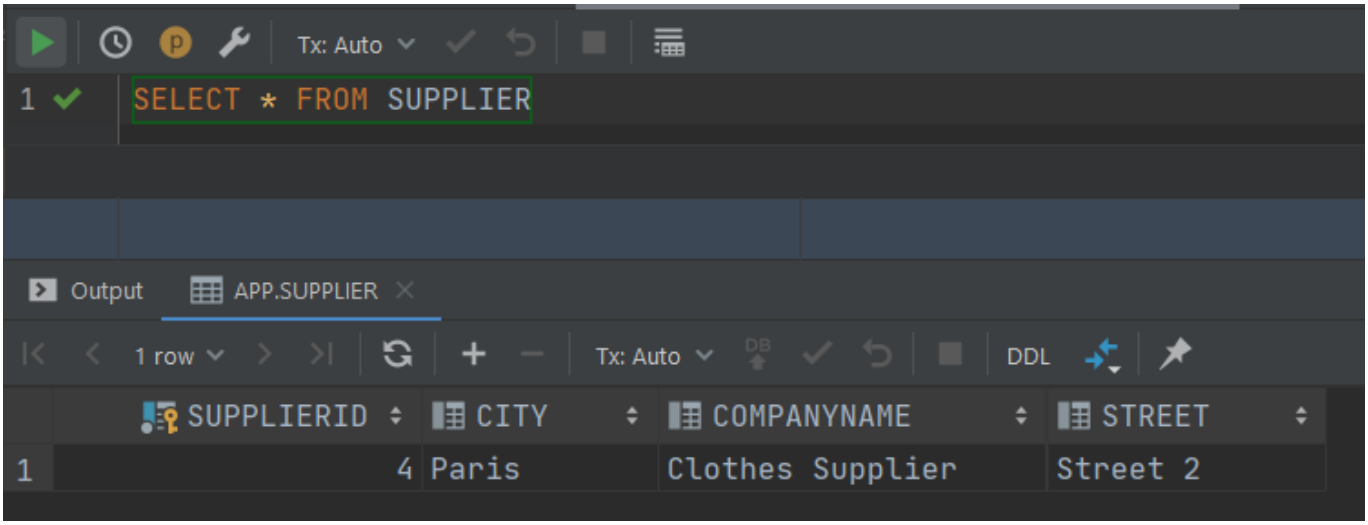


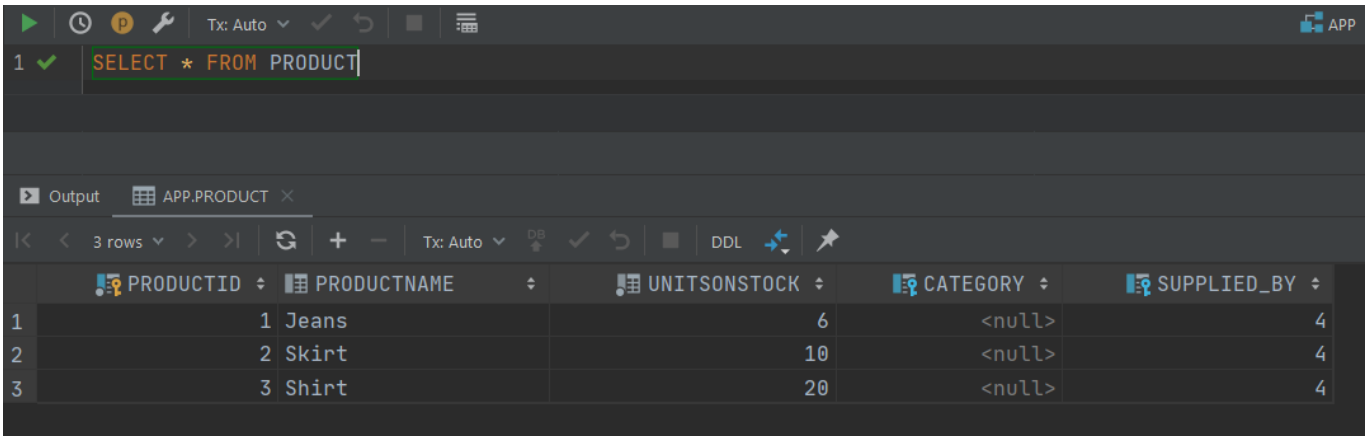
Skrypt generujący bazę:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000000-d7bd80dd-0171-c670-619d-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000001-f14580e4-0171-c670-619d-ffffc71c8945"
            primary key,
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000004-1e2700f9-0171-c670-619d-ffffc71c8945"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000003-247a40f2-0171-c670-619d-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000002-3ad9c0eb-0171-c670-619d-ffffc71c8945"
```

```
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000005-97e00100-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000006-61228105-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000007-6a6b410a-0171-c670-619d-ffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000008-83ba410f-0171-c670-619d-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

Obiekty dodane do bazy:





Zadanie XI. Kaskady

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

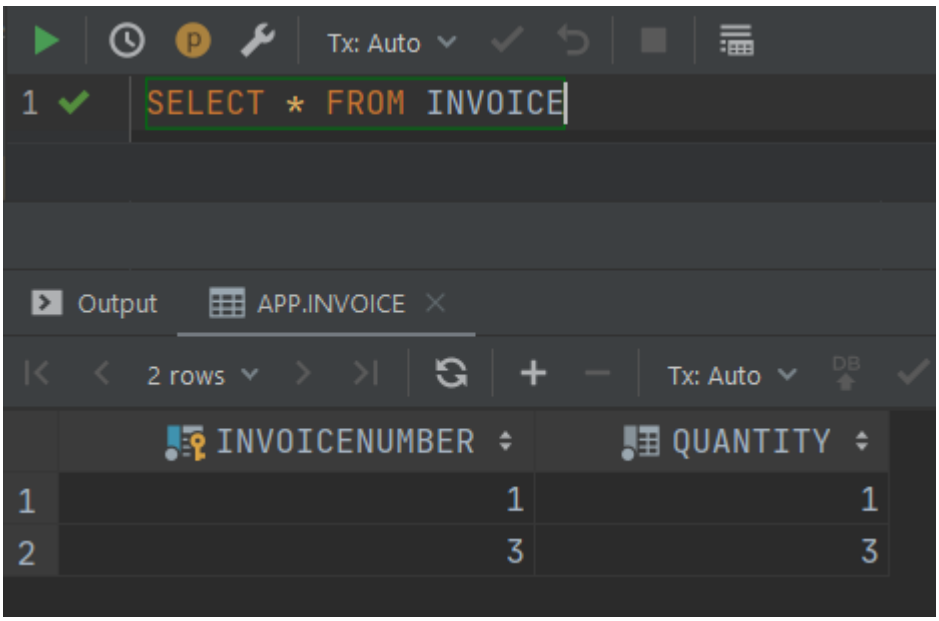
W klasie `Product` zbiór faktur wygląda następująco:

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

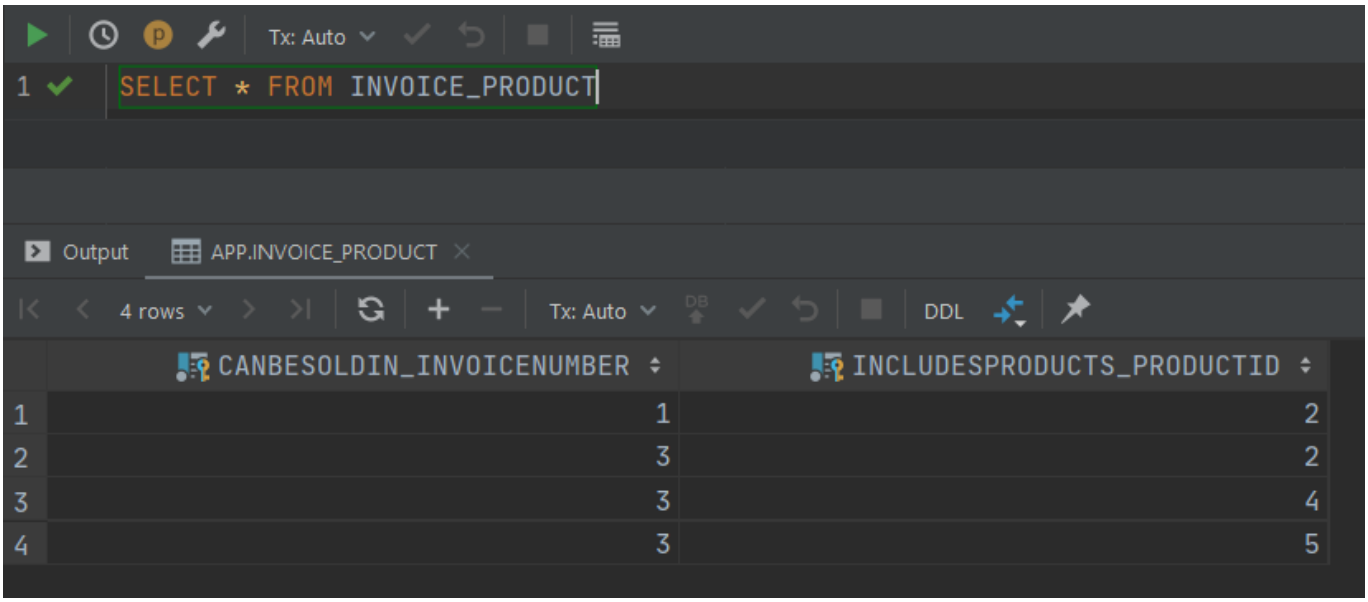
W klasie `Invoice` zbiór produktów:

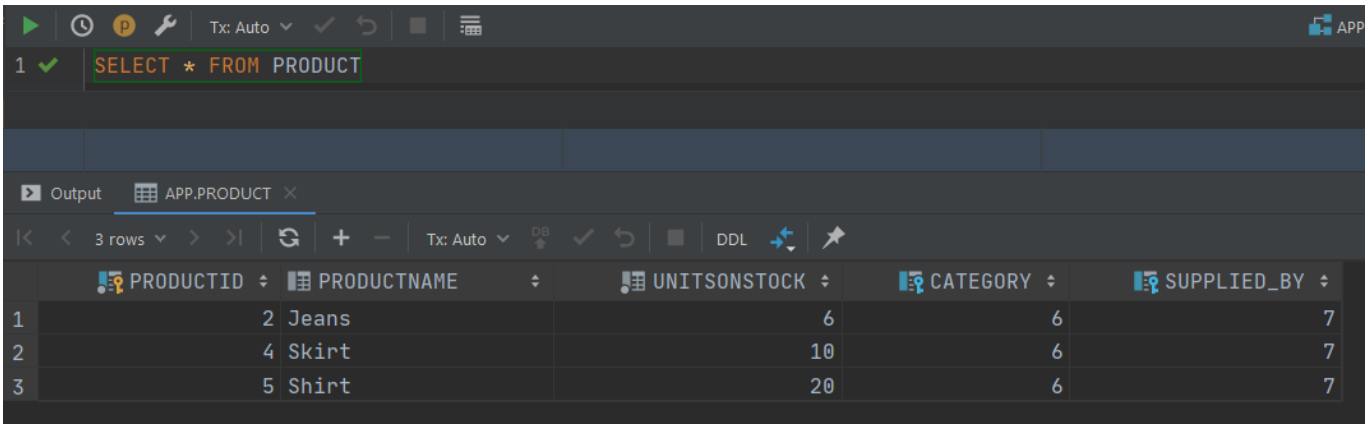
```
@ManyToMany(cascade = CascadeType.PERSIST)
private Set<Product> includesProducts = new HashSet<>();
```

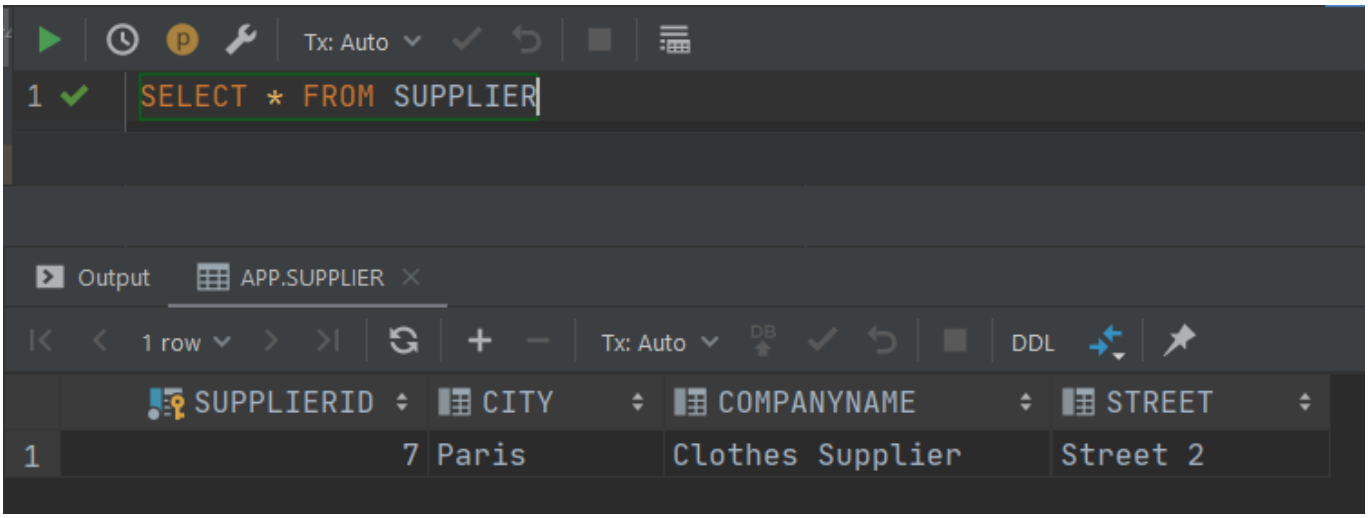
W klasie `JPAMain` stworzono kilka produktów, kategorię, dostawcę i faktury i dodano wszystkie rzeczy oprócz produktów do bazy.

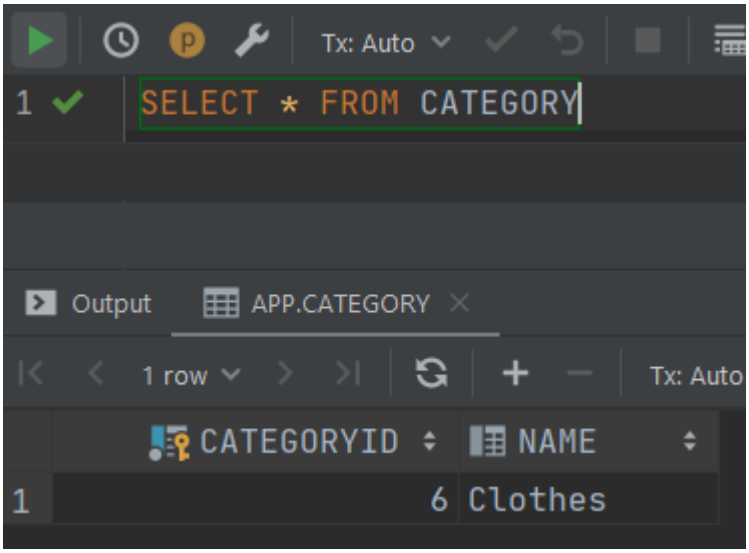


```
public static void main(String argv[]){
    EntityManager entityManager = getEntityManager();
    EntityTransaction etx = entityManager.getTransaction();
    etx.begin();

    Supplier supplier = new Supplier( companyName: "Clothes Supplier",  street: "Street 2",  city: "Paris");
    Category category = new Category( name: "Clothes");
    Product product1 = new Product( productName: "Jeans",  unitsOnStock: 6);
    Product product2 = new Product( productName: "Skirt",  unitsOnStock: 10);
    Product product3 = new Product( productName: "Shirt",  unitsOnStock: 20);
    supplier.addProduct(product1);
    supplier.addProduct(product2);
    supplier.addProduct(product3);
    category.addProduct(product1);
    category.addProduct(product2);
    category.addProduct(product3);

    Invoice invoice1 = new Invoice( quantity: 0);
    Invoice invoice2 = new Invoice( quantity: 0);

    invoice1.addProduct(product1);
    invoice2.addProduct(product1);
    invoice2.addProduct(product2);
    invoice2.addProduct(product3);

    entityManager.persist(invoice1);
    entityManager.persist(invoice2);
    entityManager.persist(category);
    entityManager.persist(supplier);

    etx.commit();
    entityManager.close();
}
```









Wszystkie obiekty (łącznie z produktami) dodały się prawidłowo do bazy.

Spróbowano również dodać wszystkie obiekty oprócz faktur jak poniżej.

```
public static void main(String argv[]){
    EntityManager entityManager = getEntityManager();
    EntityTransaction etx = entityManager.getTransaction();
    etx.begin();

    Supplier supplier = new Supplier( companyName: "Furniture Supplier",  street: "Street 3",  city: "Rome");
    Category category = new Category( name: "Furniture");
    Product product1 = new Product( productName: "Bed",  unitsOnStock: 3);
    Product product2 = new Product( productName: "Chair",  unitsOnStock: 15);
    Product product3 = new Product( productName: "Table",  unitsOnStock: 2);
    supplier.addProduct(product1);
    supplier.addProduct(product2);
    supplier.addProduct(product3);
    category.addProduct(product1);
    category.addProduct(product2);
    category.addProduct(product3);

    Invoice invoice1 = new Invoice( quantity: 0);
    Invoice invoice2 = new Invoice( quantity: 0);

    invoice1.addProduct(product1);
    invoice2.addProduct(product1);
    invoice2.addProduct(product2);
    invoice2.addProduct(product3);

    entityManager.persist(product1);
    entityManager.persist(product2);
    entityManager.persist(product3);
    entityManager.persist(category);
    entityManager.persist(supplier);

    etx.commit();
    entityManager.close();
}
```

```
SELECT PRODUCTID, PRODUCTNAME, UNITSONSTOCK, CATEGORY, SUPPLIED_BY, INVOICENUMBER, QUANTITY
FROM PRODUCT
JOIN INVOICE_PRODUCT IP on PRODUCT.PRODUCTID = IP.INCLUDESPRODUCTS_PRODUCTID
JOIN INVOICE I on IP.CANBESOLDIN_INVOICENUMBER = I.INVOICENUMBER
```

Output | Result 2

4 rows

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY	INVOICENUMBER	QUANTITY
1	1	Bed	3	6	7	2	1
2	1	Bed	3	6	7	3	3
3	4	Chair	15	6	7	3	3
4	5	Table	2	6	7	3	3

Obiekty (łącznie z fakturami) również dodały się prawidłowo. Kaskadowość zatem działa.

Zadanie XII. Embedded class

a), b) Modyfikacja modelu

Dodano do modelu nową klasę `Address` i "wbudowano" ją do tabeli dostawców.

Klasa ta zawiera następujące pola:

- private String country
- private String city
- private String street
- private String zipCode

Address.java

```java
import javax.persistence.Embeddable;

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

    public Address() {
    }

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

```

W klasie `Supplier` pola `City` i `Street` zastąpiono polem `Address`:

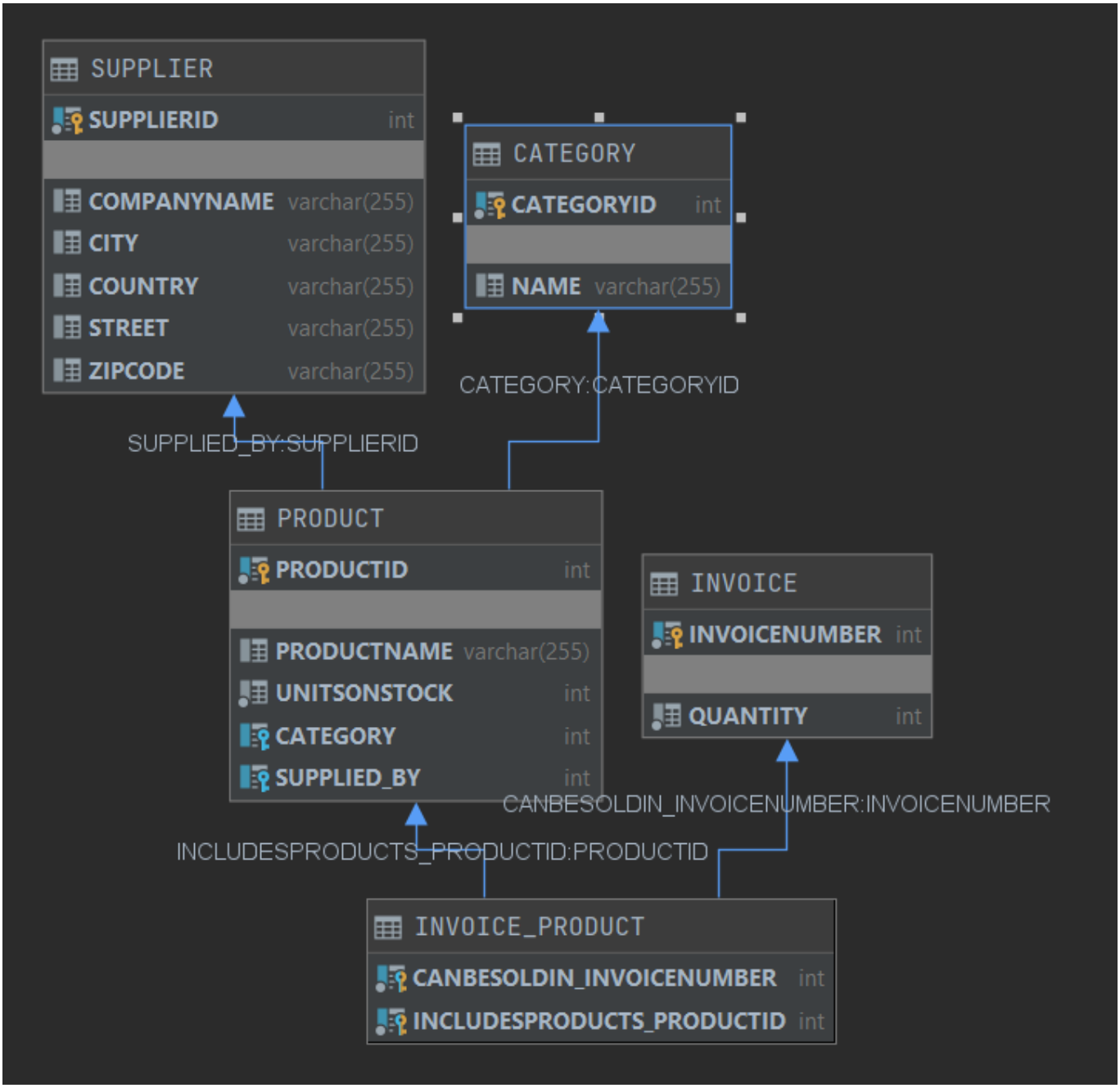
Supplier.java

```java
import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierId;
    private String CompanyName;

    @Embedded
    private Address address;

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

    public Supplier() {
    }

    public Supplier(String companyName, Address address, Product product) {
        CompanyName = companyName;
        this.address = address;
        this.suppliedProducts.add(product);
    }

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

Dodano do bazy kilku dostawców:

```java
etx.begin();

Address address1 = new Address( country: "England",  city: "London",  street: "Downing Street",  zipCode: "33-330");
Address address2 = new Address( country: "Italy",  city: "Rome",  street: "Street1",  zipCode: "33-395");
Supplier supplier1 = new Supplier( companyName: "Supplier1", address1);
Supplier supplier2 = new Supplier( companyName: "Supplier2", address2);
Supplier supplier3 = new Supplier( companyName: "Supplier3", address1);

entityManager.persist(supplier1);
entityManager.persist(supplier2);
entityManager.persist(supplier3);

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

Schemat bazy danych wygląda następująco:

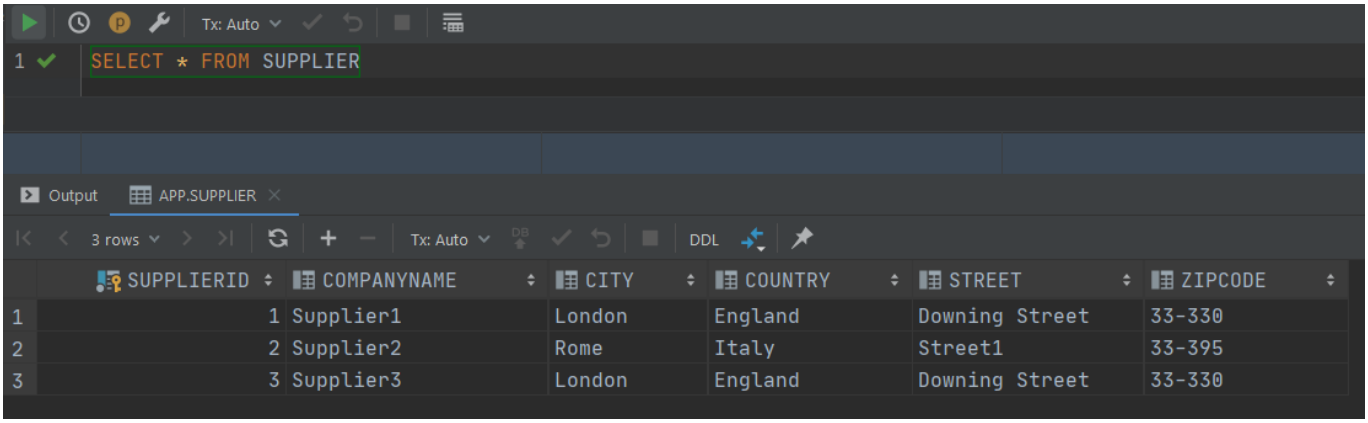


Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000054-5abe851c-0171-c670-619d-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000055-d26c4522-0171-c670-619d-ffffc71c8945"
            primary key,
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000058-a2f88535-0171-c670-619d-ffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255),
    CITY        VARCHAR(255),
    COUNTRY     VARCHAR(255),
    STREET      VARCHAR(255),
    ZIPCODE     VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000057-01e2c52e-0171-c670-619d-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
```

```
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000056-8a230528-0171-c670-619d-ffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000059-8acbc53b-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000060-d008453f-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000061-1548c543-0171-c670-619d-ffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000062-5a8d4547-0171-c670-619d-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

Dane dodały się poprawnie:



	SUPPLIERID	COMPANYNAME	CITY	COUNTRY	STREET	ZIPCODE
1	1	Supplier1	London	England	Downing Street	33-330
2	2	Supplier2	Rome	Italy	Street1	33-395
3	3	Supplier3	London	England	Downing Street	33-330

c), d) Modyfikacja modelu

Zmodyfikowano model w taki sposób, aby dane adresowe znajdowały się w klasie dostawców. Zmapowano to do dwóch osobnych tabel.

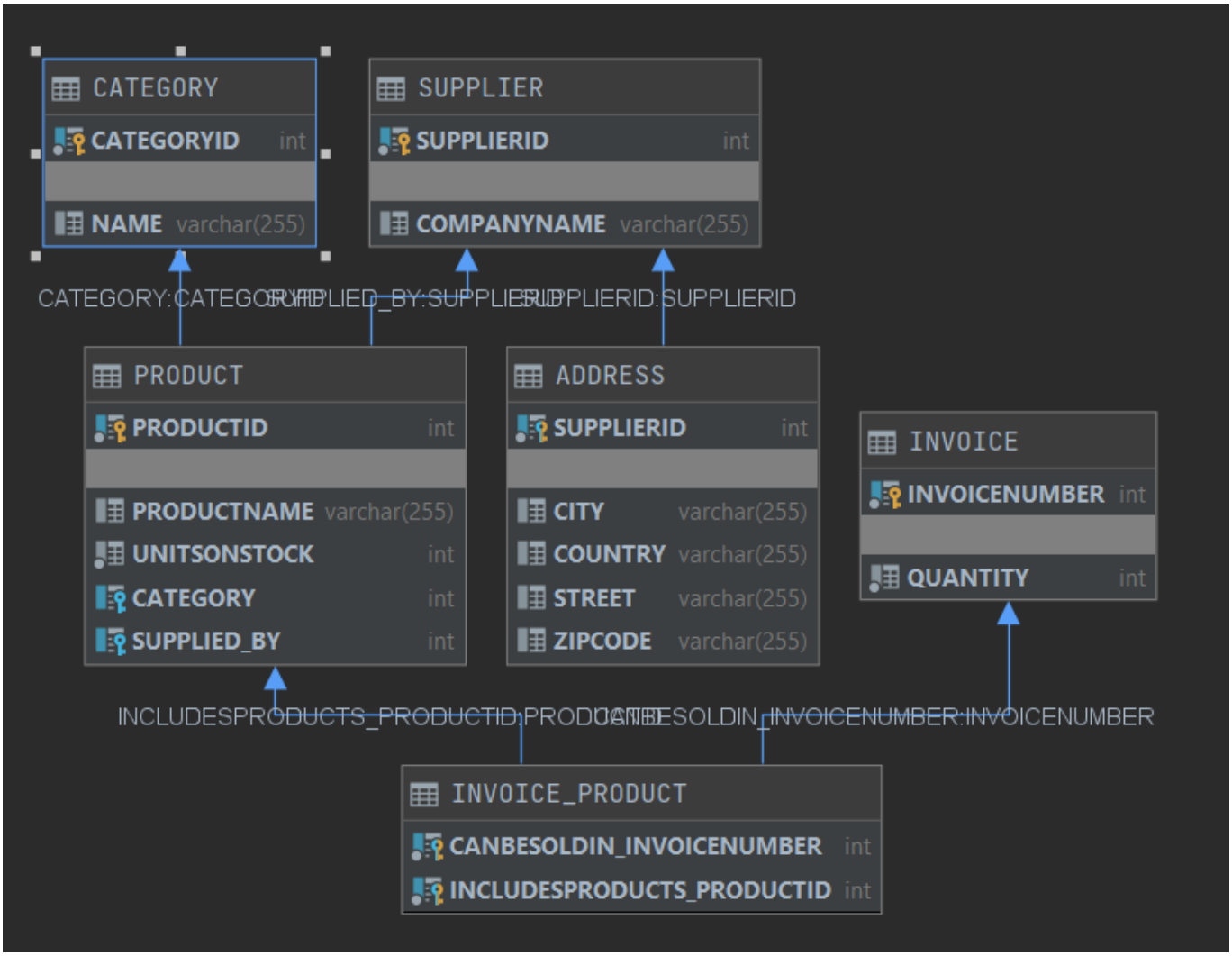
Klasa `Supplier`:

```
@Entity
@SecondaryTable(name="ADDRESS")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierId;
    private String CompanyName;

    @Column(table="ADDRESS")
    private String country;
    @Column(table="ADDRESS")
    private String city;
    @Column(table="ADDRESS")
    private String street;
    @Column(table="ADDRESS")
    private String zipCode;

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

Dodano kilku dostawców:

```
etx.begin();

Supplier supplier1 = new Supplier( companyName: "Supplier1",  country: "England",  city: "London",  street: "Downing Street",  zipCode: "33-330");
Supplier supplier2 = new Supplier( companyName: "Supplier2",  country: "Italy",  city: "Rome",  street: "Street1",  zipCode: "33-395");
Supplier supplier3 = new Supplier( companyName: "Supplier3",  country: "Poland",  city: "Cracow",  street: "Kawiory",  zipCode: "33-300");

entityManager.persist(supplier1);
entityManager.persist(supplier2);
entityManager.persist(supplier3);

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

Schemat bazy danych:

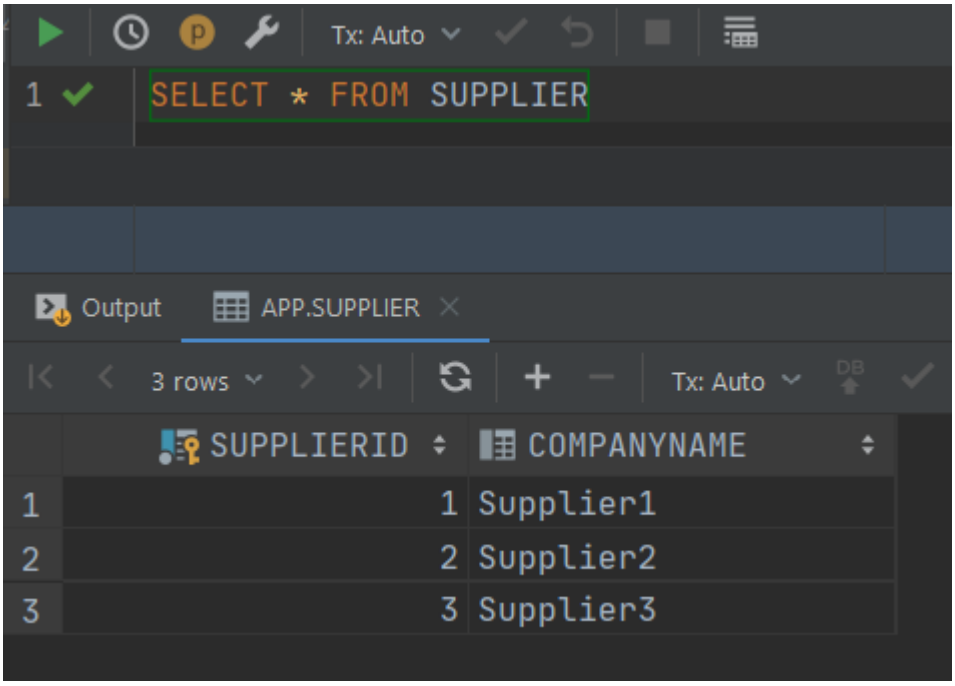


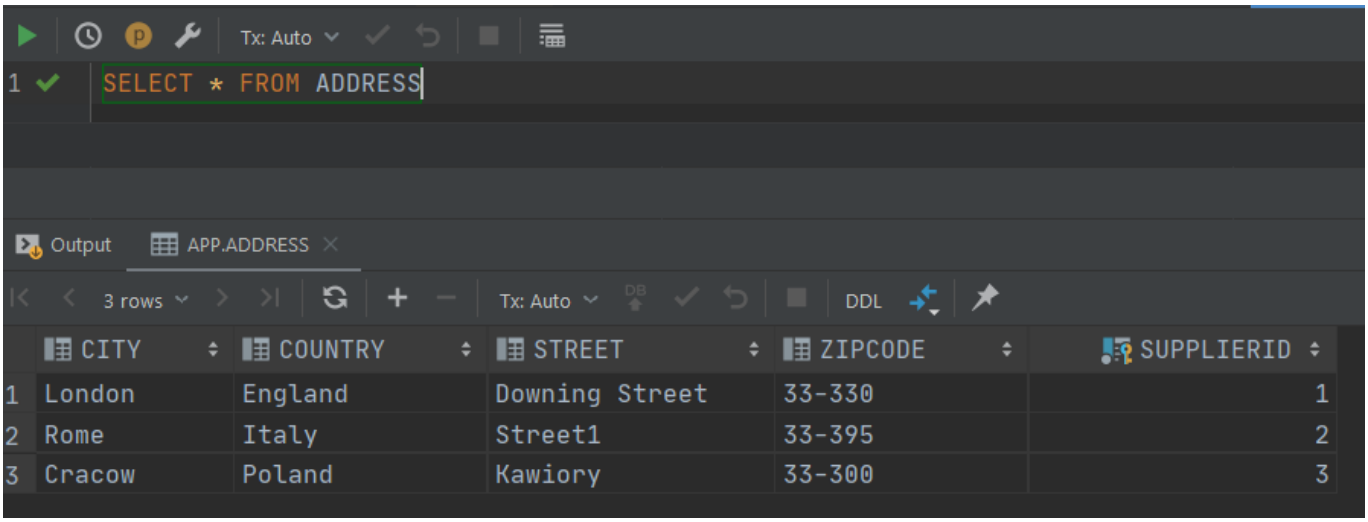
Skrypt generujący bazę:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000075-791dc68b-0171-c670-619d-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000076-e2f20691-0171-c670-619d-ffffc71c8945"
            primary key,
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000079-20a4c6a3-0171-c670-619d-ffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255)
);

create table ADDRESS
(
    CITY       VARCHAR(255),
    COUNTRY    VARCHAR(255),
    STREET     VARCHAR(255),
    ZIPCODE    VARCHAR(255),
    SUPPLIERID INTEGER not null
        constraint "SQL0000000074-8f528685-0171-c670-619d-ffffc71c8945"
            primary key
        constraint FKFIXSMY956R97J539UD6866JN5
            references SUPPLIER (SUPPLIERID)
);

create index "SQL0000000080-8a9d06a9-0171-c670-619d-ffffc71c8945"
    on ADDRESS (SUPPLIERID);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000078-36b5869d-0171-c670-619d-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
```

```
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (SUPPLIERID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000077-4ccf4697-0171-c670-619d-ffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000081-3af2c6ae-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000082-a1a246b2-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000083-0855c6b6-0171-c670-619d-ffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000084-6f0d46ba-0171-c670-619d-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

Obiekty dodane do bazy:

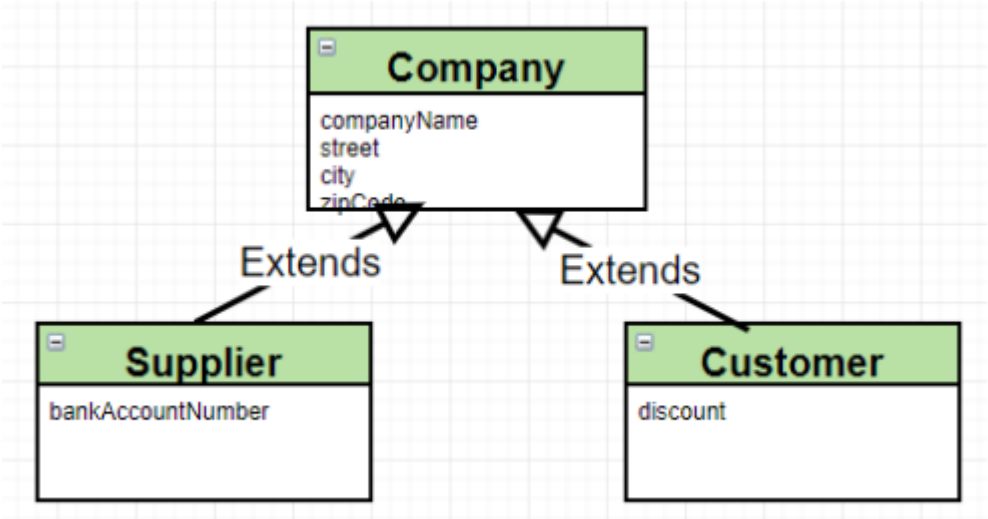




	CITY	COUNTRY	STREET	ZIPCODE	SUPPLIERID
1	London	England	Downing Street	33-330	1
2	Rome	Italy	Street1	33-395	2
3	Cracow	Poland	Kawiory	33-300	3

Zadanie XIII. Dziedziczenie

Wprowadzono do modelu następującą hierarchię:



a) SINGLE TABLE

Klasa Company:

```
@Entity
@SecondaryTable(name="ADDRESS")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyId;
    private String CompanyName;

    @Column(table="ADDRESS")
    private String country;
    @Column(table="ADDRESS")
    private String city;
    @Column(table="ADDRESS")
    private String street;
    @Column(table="ADDRESS")
    private String zipCode;

    public Company() {}

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

Klasa `Supplier`:

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

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

    public Supplier() {
    }

    public Supplier(String bankAccountNumber) {
        this.bankAccountNumber = bankAccountNumber;
    }

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

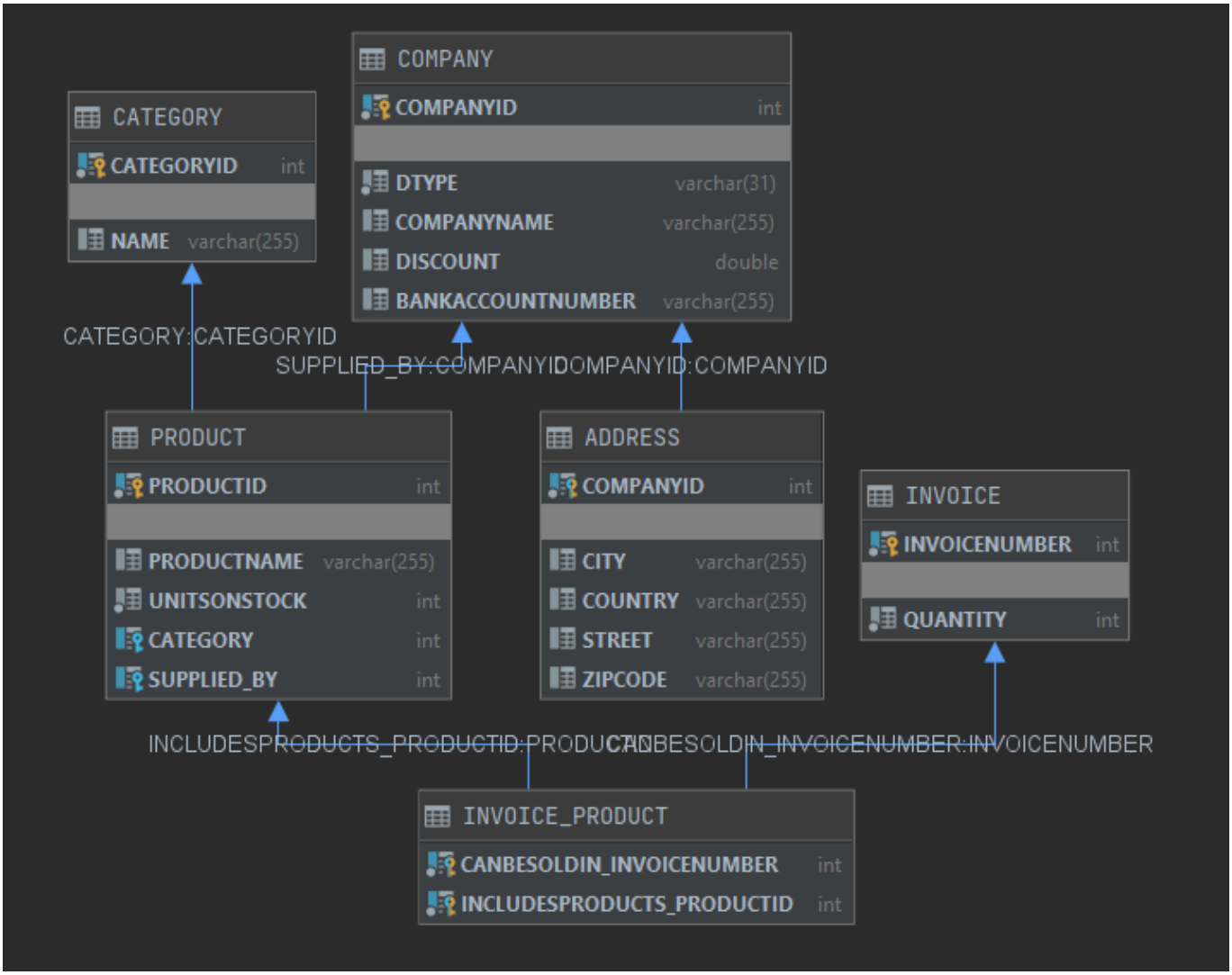
Klasa `Customer`:

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

    public Customer(){}
    public Customer(double discount) { this.discount = discount; }

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

    public double getDiscount() { return discount; }

    public void setDiscount(double discount) { this.discount = discount; }
}
```

Dodanie obiektów do bazy:

```java
etx.begin();

Supplier supplier1 = new Supplier( companyName: "Supplier1", country: "England", city: "London", street: "Downing Street", zipCode: "33-330", bankAccountNumber: "123456789");
Supplier supplier2 = new Supplier( companyName: "Supplier2", country: "Italy", city: "Rome", street: "Street1", zipCode: "33-395", bankAccountNumber: "987654321");
Customer customer1 = new Customer( companyName: "Customer1", country: "Poland", city: "Cracow", street: "Kawiory", zipCode: "33-300", discount: 0.5);
Customer customer2 = new Customer( companyName: "Customer2", country: "Poland", city: "Warsaw", street: "Street2", zipCode: "33-360", discount: 0.2);

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

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

Schemat bazy danych:

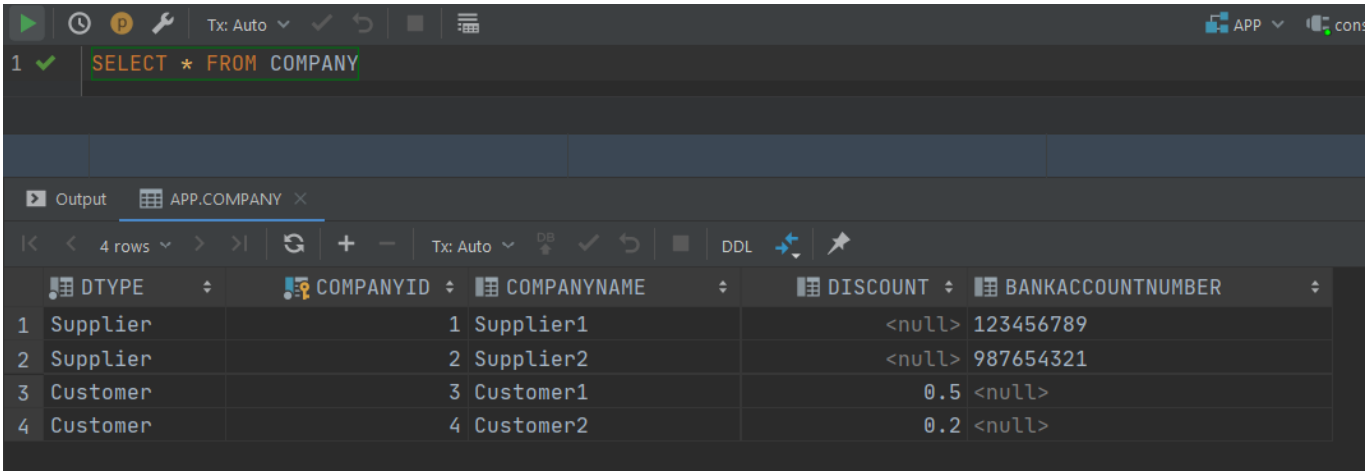


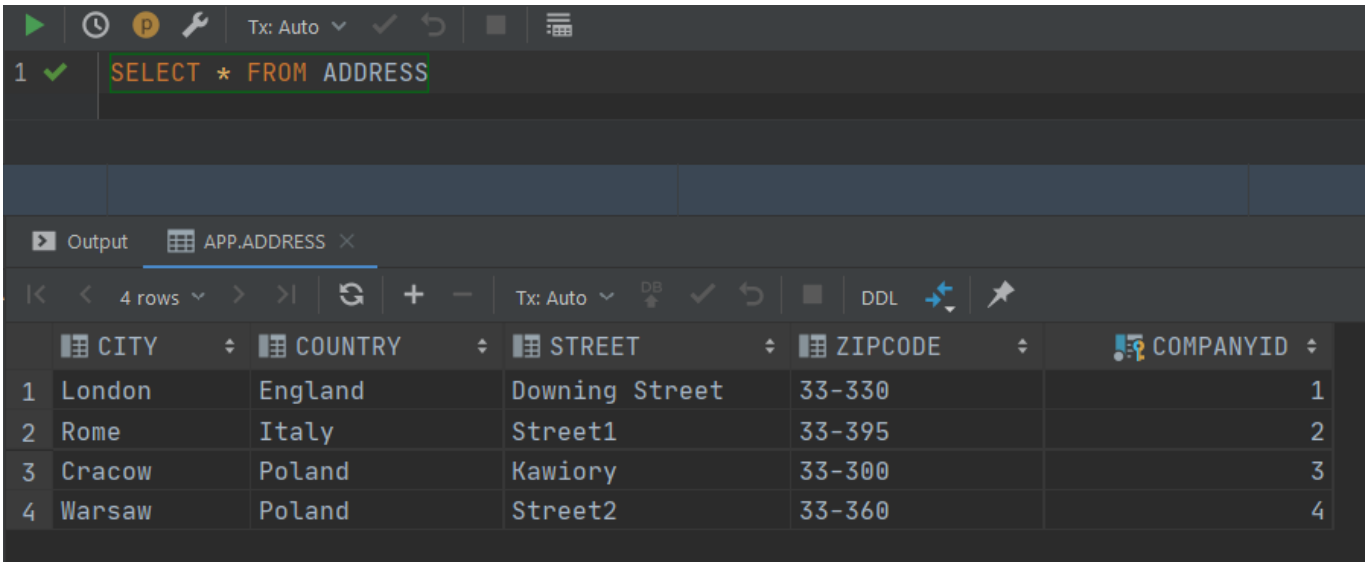
Skrypt generujący bazę danych:

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000108-0f9fc95b-0171-c670-619d-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table COMPANY
(
    DTYPE             VARCHAR(31) not null,
    COMPANYID         INTEGER     not null
        constraint "SQL0000000109-7dac0961-0171-c670-619d-ffffc71c8945"
            primary key,
    COMPANYNAME       VARCHAR(255),
    DISCOUNT          DOUBLE,
    BANKACCOUNTNUMBER VARCHAR(255)
);

create table ADDRESS
(
    CITY      VARCHAR(255),
    COUNTRY   VARCHAR(255),
    STREET    VARCHAR(255),
    ZIPCODE   VARCHAR(255),
    COMPANYID INTEGER not null
        constraint "SQL0000000107-219c8955-0171-c670-619d-ffffc71c8945"
            primary key
        constraint FKG1JYTBEOJJH2N1XV4JDOHNOQ2
            references COMPANY (COMPANYID)
);

create index "SQL0000000113-36370979-0171-c670-619d-ffffc71c8945"
    on ADDRESS (COMPANYID);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000110-ebc14967-0171-c670-619d-ffffc71c8945"
            primary key,
    QUANTITY      INTEGER not null
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000112-c806c973-0171-c670-619d-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
```

```
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY  INTEGER
        constraint FKORWD7TI86M4OHR0CRO5GFGLHG
            references COMPANY (COMPANYID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000111-d9df896d-0171-c670-619d-ffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000114-6a10c97e-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000115-d3904982-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000116-3d13c986-0171-c670-619d-ffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000117-a69b498a-0171-c670-619d-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

Obiekty w bazie danych:





	CITY	COUNTRY	STREET	ZIPCODE	COMPANYID
1	London	England	Downing Street	33-330	1
2	Rome	Italy	Street1	33-395	2
3	Cracow	Poland	Kawiory	33-300	3
4	Warsaw	Poland	Street2	33-360	4

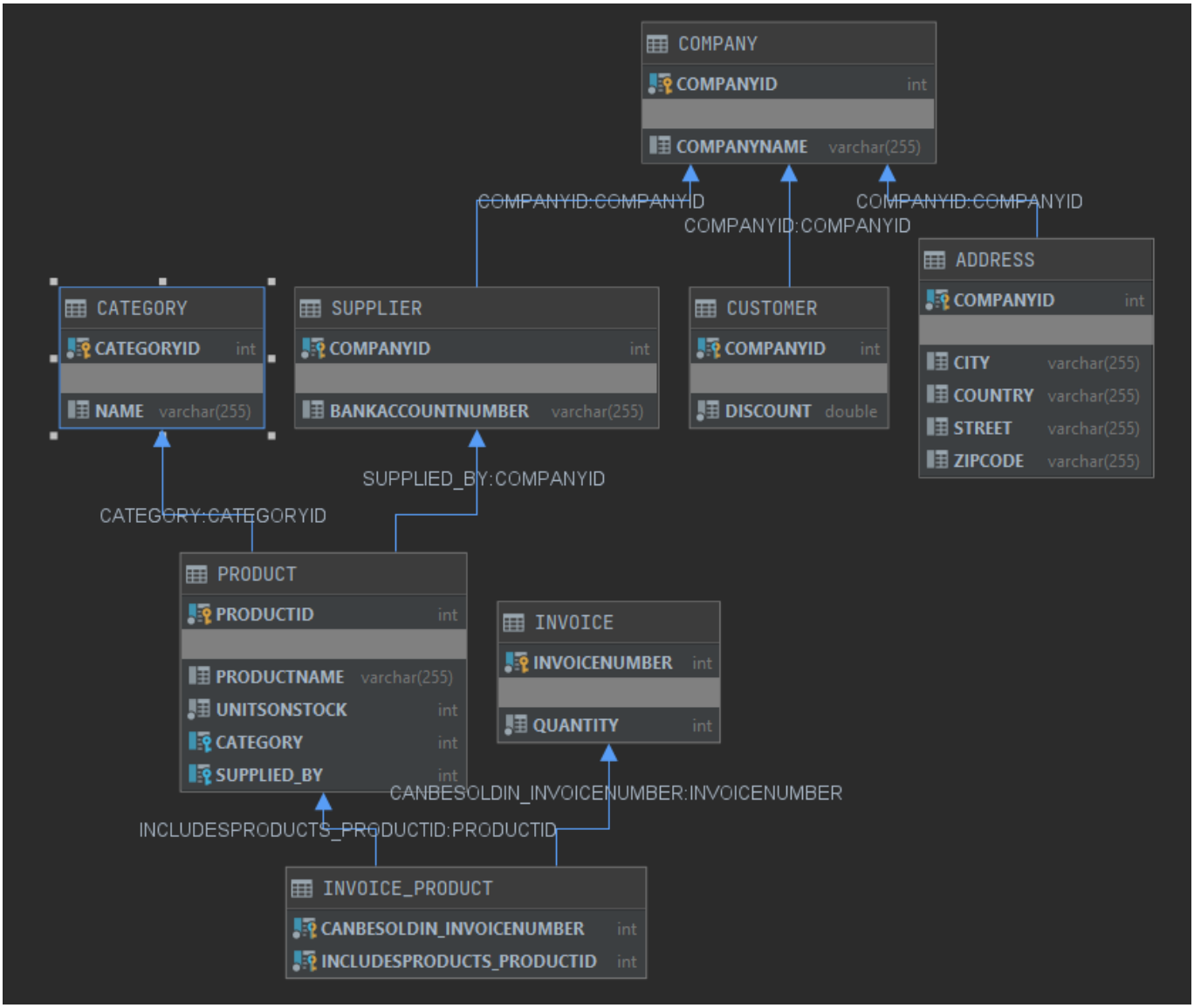
b) JOINED

Klasa `Company`:

```
@Entity
@SecondaryTable(name="ADDRESS")
@Inheritance(strategy = InheritanceType.JOINED)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyId;
    private String CompanyName;

    @Column(table="ADDRESS")
    private String country;
    @Column(table="ADDRESS")
    private String city;
    @Column(table="ADDRESS")
    private String street;
    @Column(table="ADDRESS")
    private String zipCode;
```

Klasa `Supplier` i `Customer` pozostały bez zmian. Do bazy dodano te same obiekty, co poprzednio.

Schemat bazy danych:

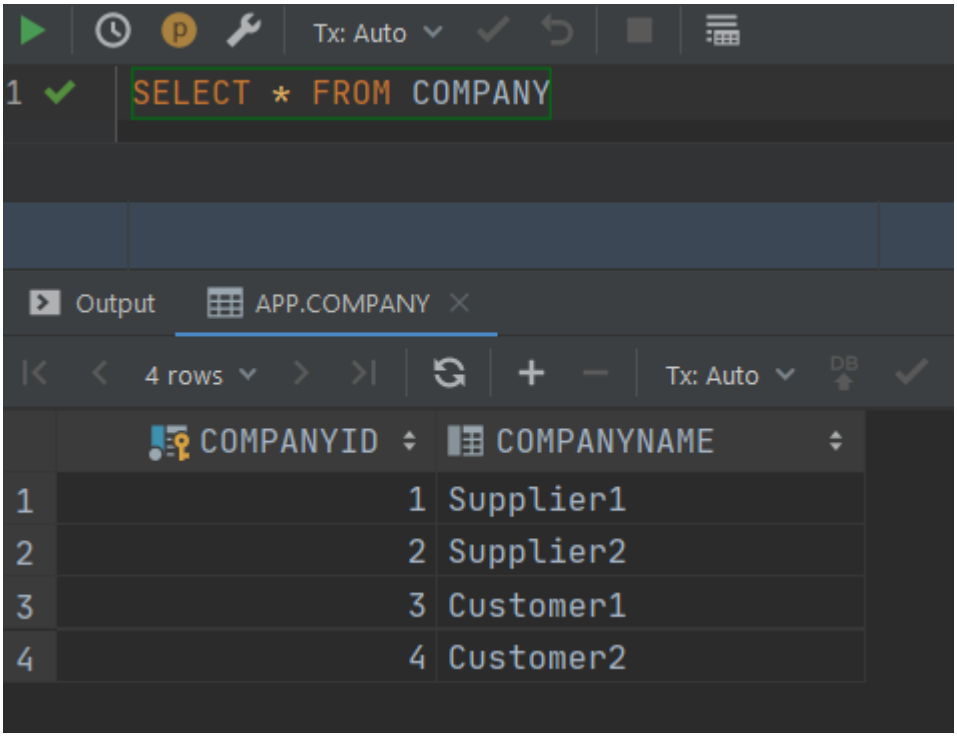


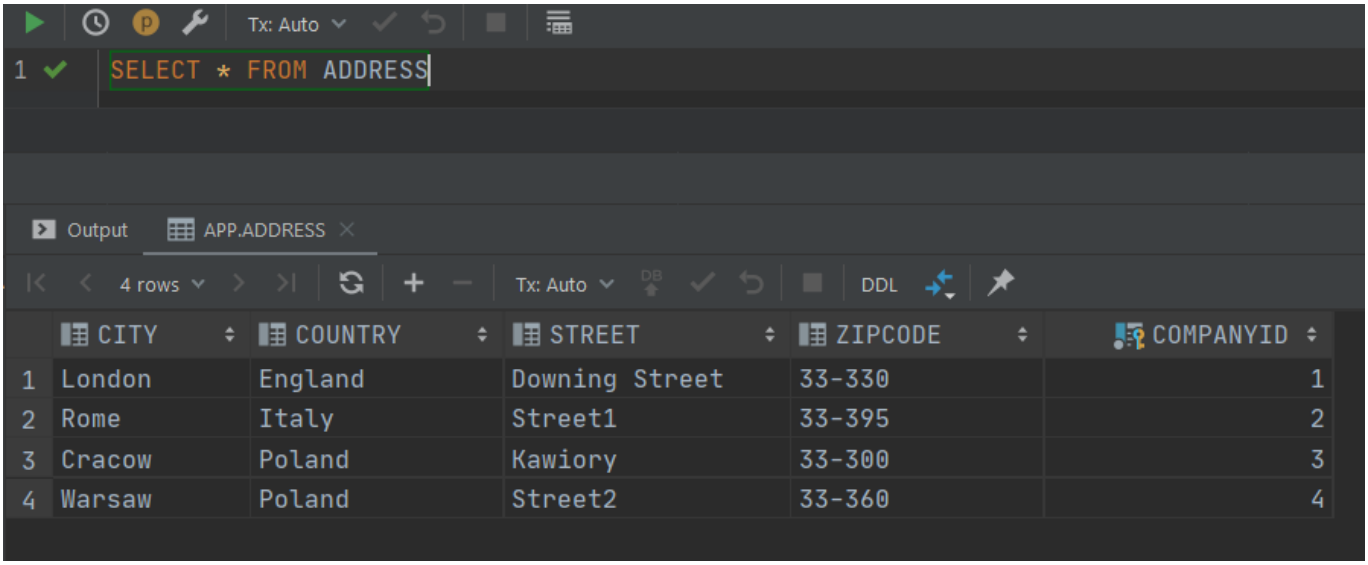
Skrypt generujący bazę danych:

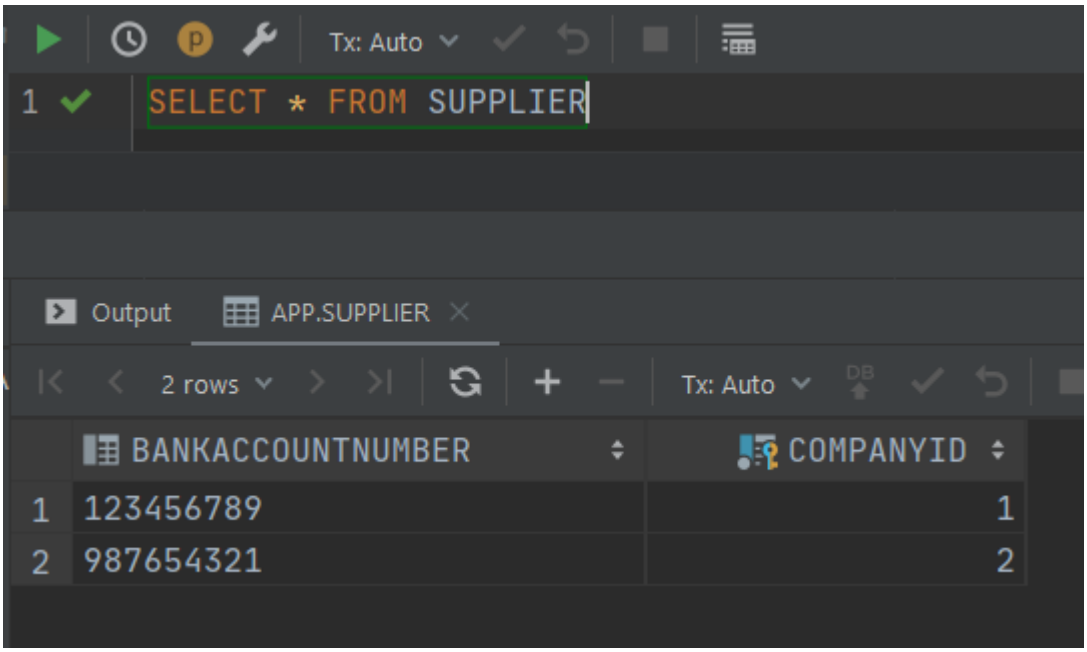
```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000134-2bff0bb9-0171-c670-619d-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table COMPANY
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000120-f3748a15-0171-c670-619d-ffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255)
);

create table ADDRESS
(
    CITY      VARCHAR(255),
    COUNTRY   VARCHAR(255),
    STREET    VARCHAR(255),
    ZIPCODE   VARCHAR(255),
    COMPANYID INTEGER not null
        constraint "SQL0000000133-ba6ecbb3-0171-c670-619d-ffffc71c8945"
            primary key
        constraint FKG1JYTBEOJJH2N1XV4JDOHNOQ2
            references COMPANY (COMPANYID)
);

create index "SQL0000000140-21fc0be1-0171-c670-619d-ffffc71c8945"
    on ADDRESS (COMPANYID);

create table CUSTOMER
(
    DISCOUNT  DOUBLE  not null,
    COMPANYID INTEGER not null
        constraint "SQL0000000135-e958cbc3-0171-c670-619d-ffffc71c8945"
            primary key
        constraint FKRR2PXJ29E8FMTV5S31R8FKX7K
            references COMPANY (COMPANYID)
);

create index "SQL0000000141-98d7cbe6-0171-c670-619d-ffffc71c8945"
    on CUSTOMER (COMPANYID);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000136-5b010bc9-0171-c670-619d-ffffc71c8945"
            primary key,
```

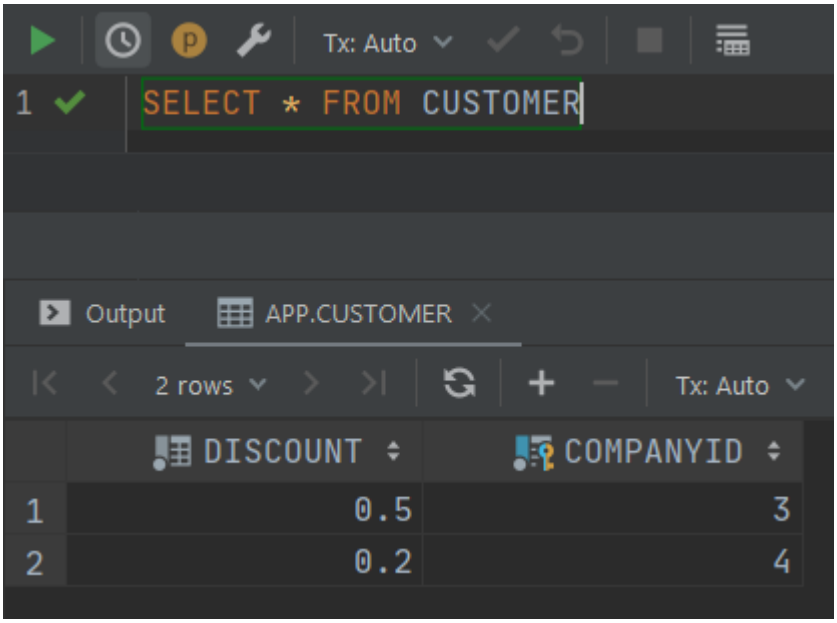
```
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    BANKACCOUNTNUMBER VARCHAR(255),
    COMPANYID         INTEGER not null
        constraint "SQL0000000139-b02fcbdb-0171-c670-619d-ffffc71c8945"
            primary key
        constraint FKNR3T1U2JVAA8PL6UPKPAXXABV
            references COMPANY (COMPANYID)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000138-be6c8bd5-0171-c670-619d-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (COMPANYID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000137-ccb24bcf-0171-c670-619d-ffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000142-dfb9cbeb-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);
create index "SQL0000000143-2ba64bef-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);
create index "SQL0000000144-7796cbf3-0171-c670-619d-ffffc71c8945"
    on PRODUCT (CATEGORY);
create index "SQL0000000145-c38b4bf7-0171-c670-619d-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
create index "SQL0000000146-0f83cbfb-0171-c670-619d-ffffc71c8945"
    on SUPPLIER (COMPANYID);
```

Obiekty w bazie danych:






Tx: Auto
1
SELECT * FROM SUPPLIER
Output
APP.SUPPLIER
2 rows
Tx: Auto
BANKACCOUNTNUMBER
COMPANYID
1 123456789 1
2 987654321 2


Tx: Auto
1
SELECT * FROM CUSTOMER
Output
APP.CUSTOMER
2 rows
Tx: Auto
DISCOUNT
COMPANYID
1 0.5 3
2 0.2 4

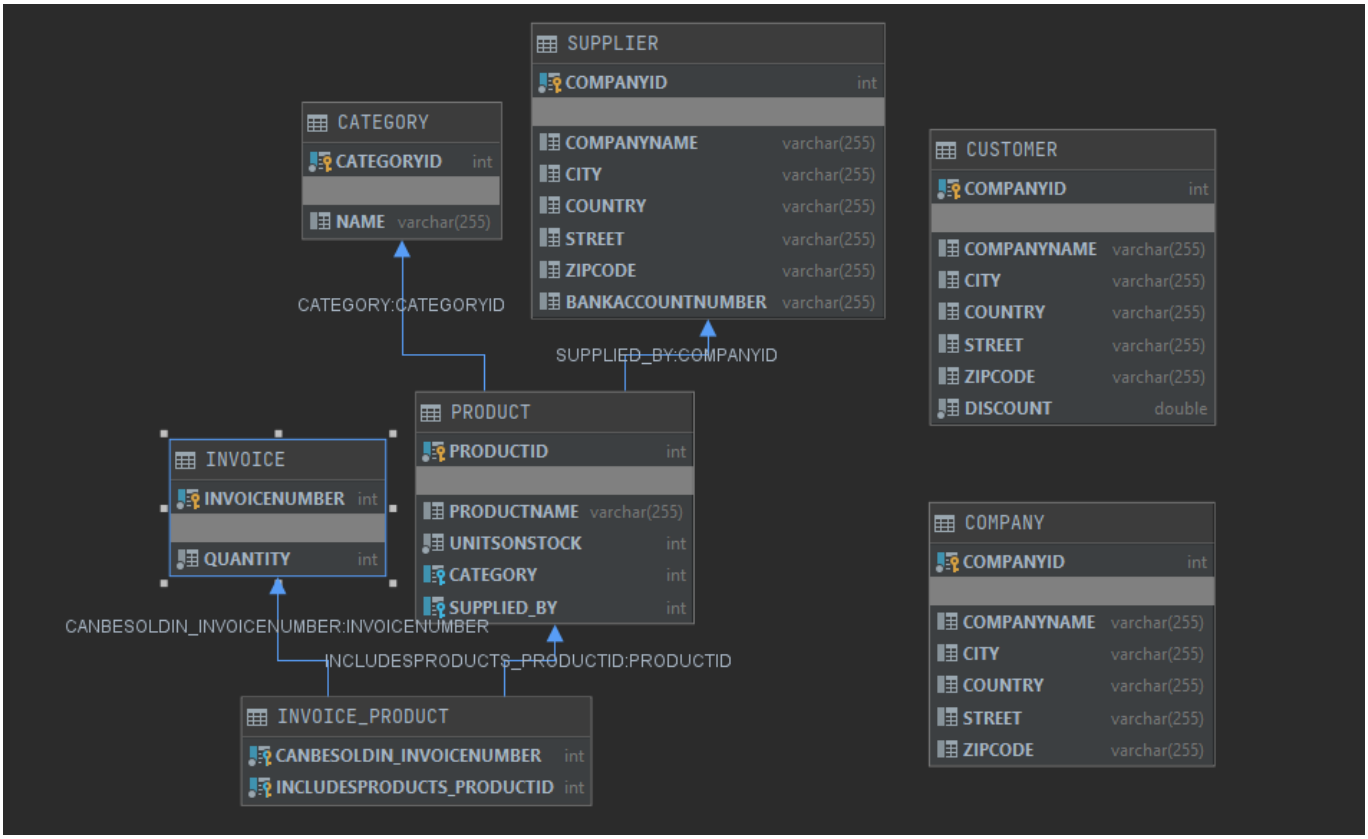
c) TABLE PER CLASS

Klasa `Company`:

```java
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyId;
    private String CompanyName;

    private String country;
    private String city;
    private String street;
    private String zipCode;

    public Company() {}

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

Klasa `Supplier` i `Customer` pozostały bez zmian. Do bazy dodano te same obiekty, co poprzednio.

Schemat bazy danych:

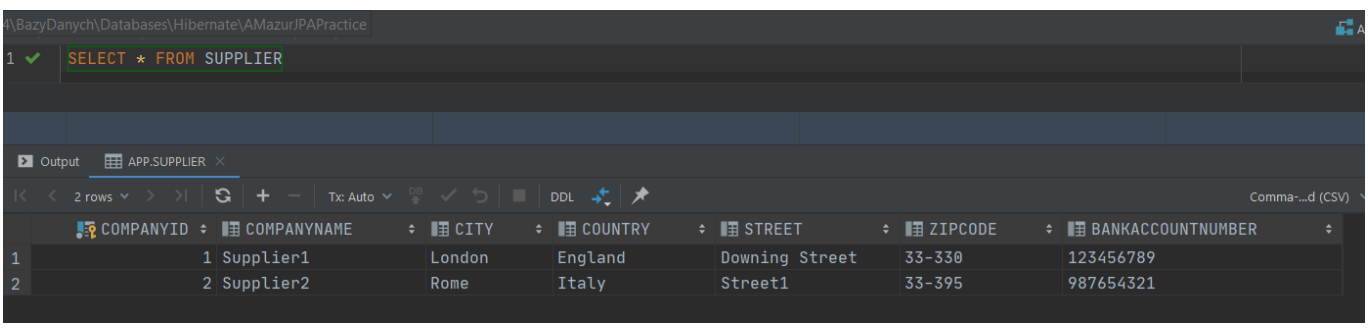


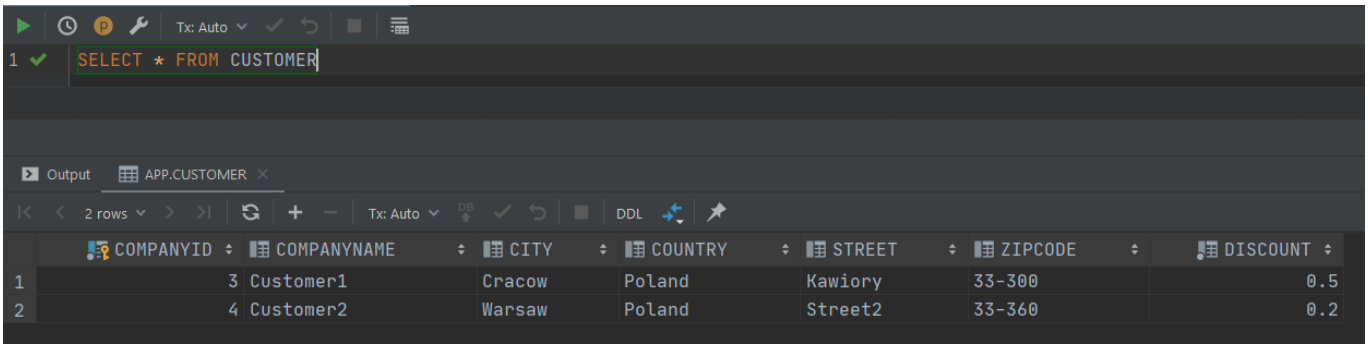
Skrypt generujący bazę danych:

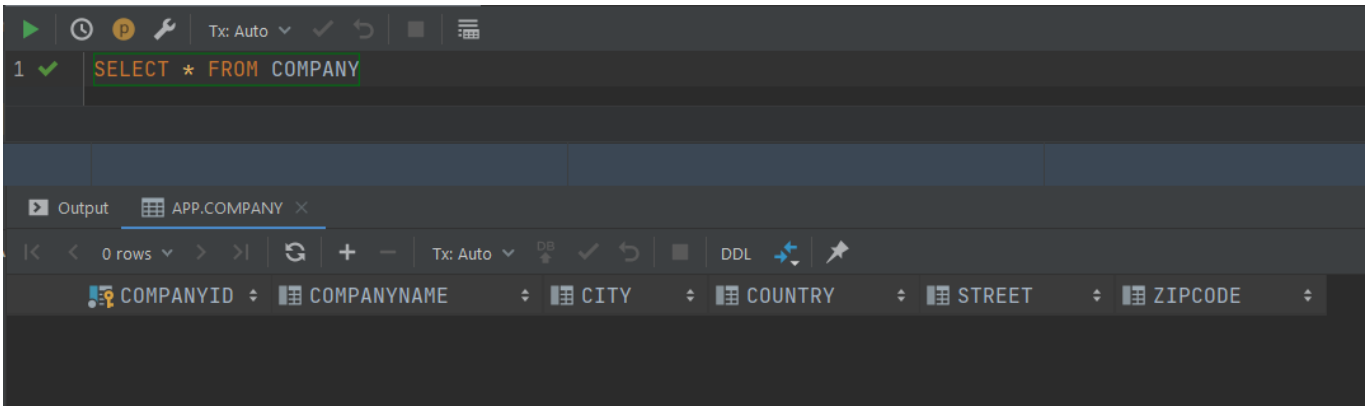
```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000193-d4429205-0171-c670-619d-ffffc71c8945"
            primary key,
    NAME       VARCHAR(255)
);

create table COMPANY
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000194-cf4dd20b-0171-c670-619d-ffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255),
    CITY        VARCHAR(255),
    COUNTRY     VARCHAR(255),
    STREET      VARCHAR(255),
    ZIPCODE     VARCHAR(255)
);

create table CUSTOMER
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000195-4a621211-0171-c670-619d-ffffc71c8945"
            primary key,
    COMPANYNAME VARCHAR(255),
    CITY        VARCHAR(255),
    COUNTRY     VARCHAR(255),
    STREET      VARCHAR(255),
    ZIPCODE     VARCHAR(255),
    DISCOUNT    DOUBLE  not null
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000196-c57f5217-0171-c670-619d-ffffc71c8945"
            primary key,
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    COMPANYID         INTEGER not null
        constraint "SQL0000000199-370d1229-0171-c670-619d-ffffc71c8945"
            primary key,
    COMPANYNAME       VARCHAR(255),
    CITY              VARCHAR(255),
    COUNTRY           VARCHAR(255),
    STREET            VARCHAR(255),
    ZIPCODE           VARCHAR(255),
```

```
    BANKACCOUNTNUMBER VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000198-bbd4d223-0171-c670-619d-ffffc71c8945"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER (COMPANYID)
);

create table INVOICE_PRODUCT
(
    CANBESOLDIN_INVOICENUMBER  INTEGER not null
        constraint FK3MT734UUBMOS8GVSXV85H0XXJ
            references INVOICE (INVOICENUMBER),
    INCLUDESPRODUCTS_PRODUCTID INTEGER not null
        constraint FKBX01IKXFWEBN63V0K8F4L2EDL
            references PRODUCT (PRODUCTID),
    constraint "SQL0000000197-c0a5921d-0171-c670-619d-ffffc71c8945"
        primary key (CANBESOLDIN_INVOICENUMBER, INCLUDESPRODUCTS_PRODUCTID)
);

create index "SQL0000000200-b24e522f-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (INCLUDESPRODUCTS_PRODUCTID);

create index "SQL0000000201-047ed233-0171-c670-619d-ffffc71c8945"
    on INVOICE_PRODUCT (CANBESOLDIN_INVOICENUMBER);

create index "SQL0000000202-56b35237-0171-c670-619d-ffffc71c8945"
    on PRODUCT (CATEGORY);

create index "SQL0000000203-a8ebd23b-0171-c670-619d-ffffc71c8945"
    on PRODUCT (SUPPLIED_BY);
```

Obiekty w bazie danych:




Tx: Auto
1
SELECT * FROM CUSTOMER
Output
APP.CUSTOMER
2 rows
Tx: Auto
DDL
COMPANYID
COMPANYNAME
CITY
COUNTRY
STREET
ZIPCODE
DISCOUNT
1 3 Customer1 Cracow Poland Kawiory 33-300 0.5
2 4 Customer2 Warsaw Poland Street2 33-360 0.2


Tx: Auto
1
SELECT * FROM COMPANY
Output
APP.COMPANY
0 rows
Tx: Auto
DDL
COMPANYID
COMPANYNAME
CITY
COUNTRY
STREET
ZIPCODE