Piotr Wróbel

Hibernate – sprawozdanie

Spis treści

1. Konfiguracja 1

2. Klasa Product 1

3. Klasa Supplier 6

4. Odwrócenie relacji 9

5. Relacja dwukierunkowa 15

6. Klasa Category 18

7. Relacja wiele-do-wielu 24

8. JPA 30

9. Kaskady 33

10. Embedded class 35

11. Dziedziczenie 38

11.1. Jedna tabela na całą hierarchię 38

11.2. Tabele łączone 42

11.3. Jedna tabela na klasę 44

12. Aplikacja do zamawiania produktów 46

1. Konfiguracja

Zgodnie z poleceniami w instrukcji do ćwiczenia, skonfigurowano serwer *Apache Derby* i utworzono na nim bazę danych *DB_Hibernate*. Wywołanie komendy *show tables* w programie ij przedstawia Rysunek 1.

2. Klasa Product

W celu dodania do bazy danych tabeli *Products* utworzono klasę *Product*. Zawiera ona pola *productID*, *productName* i *unitsOnStock*. Dodano odpowiednie anotacje – @Entity, @Table ustawiającą odpowiednią nazwę tabeli, oraz @Id dla atrybutu productID. Zapewniono także autoinkrementację klucza głównego adnotacją @GeneratedValue. Rozpatrywano możliwość ustawienia klucza głównego na pole *productName*, ale utrudniałoby to ewentualne rozbudowanie modelu (przykładowo nie można by było mieć produktu dostarczanego przez różnych dostawców). Zawarto bezparametrowy konstruktor wymagany przez Hibernate do odczytywania danych oraz konstruktor paramterowy do zapisu nowych produktów w następnym kroku. Kod klasy przedstawiono poniżej.

```
import javax.persistence.*;

```

```
wersja ij 10.14
ij> connect 'jdbc:derby://127.0.0.1/BD_Hibernate';
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                     |

23 wierszy wybranych
```

Rysunek 1. Wywołanie komendy *show tables* na nowo utworzonej bazie danych

```java
@Entity
@Table(name = "Products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productID;
    private String productName;
    private int unitsOnStock;

    public Product() {
    }

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

W celu poprawnego zmigrowania modelu, dodano wpis do pliku konfiguracyjnego .xml, po modyfikacji ma on następującą postać:

```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/BD_Hibernate</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">create</property>
        <mapping class="Product"></mapping>
    </session-factory>
</hibernate-configuration>
```

W klasie Main zmodyfikowano wygenerowany automatycznie przy tworzeniu projektu z Hibernate kod do następującej postaci:

```java
public class Main {
    private static final SessionFactory ourSessionFactory;

    static {
        try {
            Configuration configuration = new Configuration();
            configuration.configure();

            ourSessionFactory = configuration.buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static Session getSession() throws HibernateException {
        return ourSessionFactory.openSession();
    }

    public static void main(final String[] arg) {
        final Session session = getSession();
        Transaction tx = session.beginTransaction();
        tx.commit();
        session.close();
    }
}
```

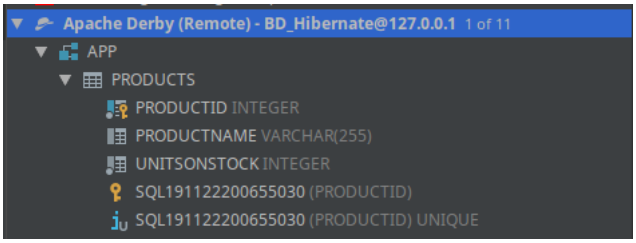
Po uruchomieniu programu zaobserwowano tworzenie nowej tabeli (w miejsce ewentualnej starej). Logi Hibernate'a przedstawia Rysunek 2. Po odświeżeniu modelu w DataGripie widoczna jest struktura tabeli (Rysunek 3) dla której wygenerowano schemat (Rysunek 4).

```
Hibernate:

    drop table Products
lis 22, 2019 8:06:54 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$Connecti
Hibernate:

    drop sequence hibernate_sequence restrict
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
lis 22, 2019 8:06:55 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$Connecti
Hibernate:

    create table Products (
       productID integer not null,
        productName varchar(255),
        unitsOnStock integer not null,
        primary key (productID)
    )
lis 22, 2019 8:06:55 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]

Process finished with exit code 0
```

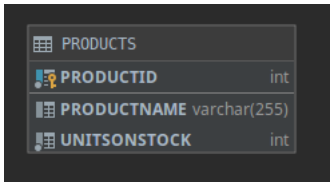
Rysunek 2. Log Hibernate dotyczący tworzenia tabeli *Products*

Następnie, w celu odpytania użytkownika o dane dodawanego produktu, zmodyfikowano funkcję main:

```java
public static void main(final String[] arg) {
  Scanner inputScanner = new Scanner(System.in);

  System.out.print("Enter product name: ");
```



Rysunek 3. Struktura tabeli *Products* widziana w DataGripie



Rysunek 4. Schemat tabeli *Products* z DataGripa

```
  String productName = inputScanner.nextLine();

  System.out.print("Enter units on stock number: ");
  int unitsOnStock =  Integer.parseInt(inputScanner.nextLine());

  Product addedProduct = new Product(productName, unitsOnStock);

  final Session session = getSession();
  Transaction tx = session.beginTransaction();
  session.save(addedProduct);
  tx.commit();
  session.close();
}
```

W konfiguracyjnym pliku .xml zmieniono ustawienie *hbm2ddl.auto* z *create* na *update*, aby nie usuwać starej i nie tworzy nowej tabeli za każdym razem. Przykładowe wywołanie przedstawia Rysunek 5. Wynik wywołania polecenia *select* na tabeli *Products* potwierdzający zapis nowych danych przedstawia Rysunek 6.

```
Enter product name: T-shirt
Enter units on stock number: 12
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    /* insert Product
        */ insert
        into
            Products
            (productName, unitsOnStock, productID)
        values
            (?, ?, ?)

Process finished with exit code 0
```

Rysunek 5. Log Hibernate dotyczący dodania nowego wiersza do tabeli *Products*

1 row

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	T-shirt	12

Rysunek 6. Wynik polecenia *select* wywołanego na tabeli *Products*

3. Klasa Supplier

Dodano nową klasę *Supplier* zawierającą pola *supplierID*, *companyName*, *address* i *city*:

```
import javax.persistence.*;

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

    public Supplier() {
    }

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

W konfiguracyjnym pliku XML ustawiono, analogicznie jak w poprzednim punkcie, odpowiednie mapowanie klasy. Aby utworzyć powiązanie między tabelami, do klasy Product dodano jedno nowe pole z adnotacją @ManyToOne oraz z dodatkowym getterem i setterem (będą przydatne przy modyfikacji utworzonego wcześniej produktu). Dodany kod ma następującą postać:

```
@ManyToOne
private Supplier supplier;

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

public Supplier getSupplier() {
  return supplier;
}
```

Zmodyfikowano funkcję main, tak aby czytano od użytkownika dane nowego dostawcy i przypisano je do dodanego poprzednio produktu:

```
public static void main(final String[] arg) {
  Scanner inputScanner = new Scanner(System.in);

  System.out.print("Enter company name: ");
  String companyName = inputScanner.nextLine();
  System.out.print("Enter company street: ");
  String street = inputScanner.nextLine();
  System.out.print("Enter company city: ");
  String city = inputScanner.nextLine();

  Supplier addedSupplier = new Supplier(companyName, street, city);

  final Session session = getSession();
  Transaction tx = session.beginTransaction();

```

```
    session.save(addedSupplier);
    Product firstProduct = session.get(Product.class, 1);
    firstProduct.setSupplier(addedSupplier);

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

Uruchomiono program i dodano nowego dostawcę. Logi Hibernate przedstawia Rysunek 7 i 8, wynik wywołania polecenia *select* na tabeli *Products* Rysunek 9 a dla tabeli *Suppliers* Rysunek 10. Stan bazy danych odczytany z DataGripa zawiera Rysunek 11, a schemat bazy danych Rysunek 12.

```
Hibernate:

    alter table Products
       add column supplier_supplierID integer
Hibernate:

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

    alter table Products
       add constraint FK8b7ydyv2hnxqvuepd10uvatrk
       foreign key (supplier_supplierID)
       references Suppliers
```

Rysunek 7. Log Hibernate dotyczący modyfikacji tabeli *Products* i dodania tabeli *Suppliers*

```
Enter company name: Hurtownia odzieży sp. z.o.o.
Enter company street: Pawia 4
Enter company city: Kraków
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    select
        product0_.productID as productI1_0_0_,
        product0_.productName as productN2_0_0_,
        product0_.supplier_supplierID as supplier4_0_0_,
        product0_.unitsOnStock as unitsOnS3_0_0_,
        supplier1_.supplierID as supplier1_1_1_,
        supplier1_.city as city2_1_1_,
        supplier1_.companyName as companyN3_1_1_,
        supplier1_.street as street4_1_1_
    from
        Products product0_
    left outer join
        Suppliers supplier1_
            on product0_.supplier_supplierID=supplier1_.supplierID
    where
        product0_.productID=?
Hibernate:
    /* insert Supplier
        */ insert
        into
            Suppliers
            (city, companyName, street, supplierID)
        values
            (?, ?, ?, ?)
Hibernate:
    /* update
        Product */ update
            Products
        set
            productName=?,
            supplier_supplierID=?,
            unitsOnStock=?
        where
            productID=?

Process finished with exit code 0
```

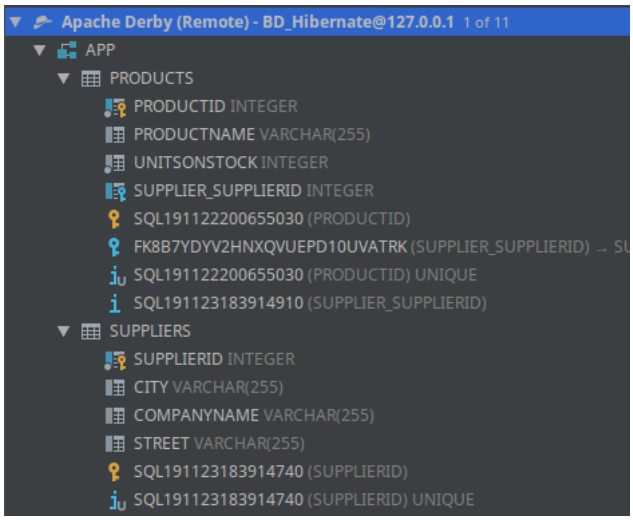
Rysunek 8. Log Hibernate dotyczący modyfikacji danych

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	1	T-shirt	12	101

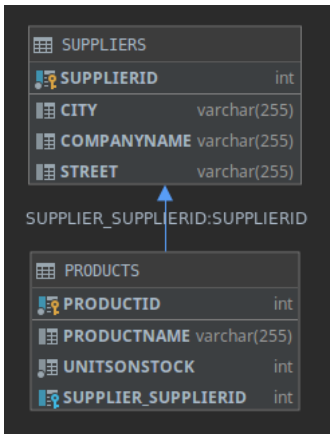
Rysunek 9. Wynik polecenia *select* wywołanego na tabeli *Products*

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	101	Kraków	Hurtownia odzieży sp. z.o.o.	Pawia 4

Rysunek 10. Wynik polecenia *select* wywołanego na tabeli *Suppliers*



Rysunek 11. Struktura bazy danych widziana w DataGripie



Rysunek 12. Schemat bazy danych

4. Odwrócenie relacji

Odwrócono relację między tabelami *Products* i *Suppliers* poprzez usunięcie pól dodanych do klasy Product w poprzednim kroku i dodaniu nowego pola do klasy Suppliers (wraz z pomocniczą metodą):

```
@OneToMany
private Set<Product> suppliedProducts;

public void addProductToList(Product addedProduct) {
  this.suppliedProducts.add(addedProduct);
}
```

Zmodyfikowano metodę main, tak aby czytać dane dostawcy i produktów od użytkownika, a następnie poprawnie je dodać do bazy danych:

```
public static void main(final String[] arg) {
  Scanner inputScanner = new Scanner(System.in);

  System.out.print("Enter company name: ");
  String companyName = inputScanner.nextLine();
  System.out.print("Enter company street: ");
  String street = inputScanner.nextLine();
  System.out.print("Enter company city: ");
  String city = inputScanner.nextLine();

  Supplier addedSupplier = new Supplier(companyName, street, city);

  System.out.print("Enter number of supplied products: ");
  int prodNumber = Integer.parseInt(inputScanner.nextLine());

  final Session session = getSession();
  Transaction tx = session.beginTransaction();

  for(int i = 0; i < prodNumber; i++) {
    System.out.print("Enter product name: ");
    String productName = inputScanner.nextLine();
    System.out.print("Enter units on stock: ");
    int unitsOnStock = Integer.parseInt(inputScanner.nextLine());

    Product nextProduct = new Product(productName, unitsOnStock);
    session.save(nextProduct);
    addedSupplier.addProductToList(nextProduct);
  }
  session.save(addedSupplier);

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

Przed uruchomieniem programu zmodyfikowano w pliku konfiguracyjnym parametr *hbm2ddl.auto* na wartość *create*. Zaobserwowano tworzenie tabeli łącznikowej – log Hibernate'a zawiera Rysunek 13 i 14, wywołania polecenia *select* na poszczególnych tabelach Rysunek 15, 16 i 17, strukturę bazy danych Rysunek 18 a schemat Rysunek 19.

```
Hibernate:

    create table Products (
       productID integer not null,
        productName varchar(255),
        unitsOnStock integer not null,
        primary key (productID)
    )
Hibernate:

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

    create table Suppliers_Products (
       Supplier_supplierID integer not null,
        suppliedProducts_productID integer not null,
        primary key (Supplier_supplierID, suppliedProducts_productID)
    )
Hibernate:

    alter table Suppliers_Products
       add constraint UK_ek0aioj1bqlu2lj8p8hmhpcmm unique (suppliedProducts_productID)
Hibernate:

    alter table Suppliers_Products
       add constraint FKhcrf68p4mhlx6je6mkwnugus4
       foreign key (suppliedProducts_productID)
       references Products
Hibernate:

    alter table Suppliers_Products
       add constraint FKggamln27clbgftqypsfqqoqeo
       foreign key (Supplier_supplierID)
       references Suppliers
```

Rysunek 13. Log Hibernate dotyczący konstrukcji tabel z odwróconą zależnością

```
Enter company name: Hurtownia napojów SA
Enter company street: Wileńska 4
Enter company city: Warszawa
Enter number of supplied products: 2
Enter product name: Kawa czarna
Enter units on stock: 12
Hibernate:

values
    next value for hibernate_sequence
Enter product name: Herbata zielona
Enter units on stock: 44
Hibernate:

values
    next value for hibernate_sequence
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    /* insert Product
        */ insert
        into
            Products
            (productName, unitsOnStock, productID)
        values
            (?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Products
            (productName, unitsOnStock, productID)
        values
            (?, ?, ?)
Hibernate:
    /* insert Supplier
        */ insert
        into
            Suppliers
            (city, companyName, street, supplierID)
        values
            (?, ?, ?, ?)
Hibernate:
    /* insert collection
        row Supplier.suppliedProducts */ insert
        into
            Suppliers_Products
            (Supplier_supplierID, suppliedProducts_productID)
        values
            (?, ?)
Hibernate:
    /* insert collection
        row Supplier.suppliedProducts */ insert
        into
            Suppliers_Products
            (Supplier_supplierID, suppliedProducts_productID)
        values
            (?, ?)

Process finished with exit code 0
```

Rysunek 14. Log Hibernate dotyczący wstawiania nowych wartości do tabel

1 row | Tx: Auto

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	3	Warszawa	Hurtownia napojów SA	Wileńska 4

Rysunek 15. Wynik polecenia *select* na tabeli *Suppliers*

2 rows | Tx: Auto

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Kawa czarna	12
2	2	Herbata zielona	44

Rysunek 16. Wynik polecenia *select* na tabeli *Products*

2 rows | Tx: Auto

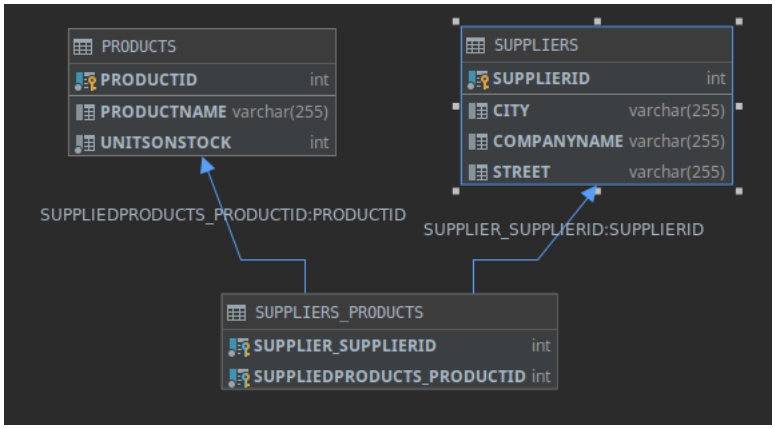
	SUPPLIER_SUPPLIERID	SUPPLIEDPRODUCTS_PRODUCTID
1	3	1
2	3	2

Rysunek 17. Wynik polecenia *select* na tabeli łącznikowej

Apache Derby (Remote) - BD_Hibernate@127.0.0.1 1 of 11

- PRODUCTID INTEGER
- PRODUCTNAME VARCHAR(255)
- UNITSONSTOCK INTEGER
- SQL191123210116560 (PRODUCTID)
- SQL191123210116560 (PRODUCTID) UNIQUE
- SUPPLIERS
  - SUPPLIERID INTEGER
  - CITY VARCHAR(255)
  - COMPANYNAME VARCHAR(255)
  - STREET VARCHAR(255)
  - SQL191123210116640 (SUPPLIERID)
  - SQL191123210116640 (SUPPLIERID) UNIQUE
- SUPPLIERS_PRODUCTS
  - SUPPLIER_SUPPLIERID INTEGER
  - SUPPLIEDPRODUCTS_PRODUCTID INTEGER
  - SQL191123210116720 (SUPPLIER_SUPPLIERID, SUPPLIEDPRODUC
  - SQL191123210116800 (SUPPLIEDPRODUCTS_PRODUCTID)
  - FKGGAMLN27CLBGFTQYPSFQQOQEO (SUPPLIER_SUPPLIERID) → S
  - FKHCRF68P4MHLX6JE6MKWNUGUS4 (SUPPLIEDPRODUCTS_PROD
  - SQL191123210116720 (SUPPLIER_SUPPLIERID, SUPPLIEDPRODUC
  - SQL191123210116800 (SUPPLIEDPRODUCTS_PRODUCTID) UNIQU
  - SQL191123210116850 (SUPPLIEDPRODUCTS_PRODUCTID)
  - SQL191123210116870 (SUPPLIER_SUPPLIERID)

Rysunek 18. Struktura bazy danych widziana w DataGripie

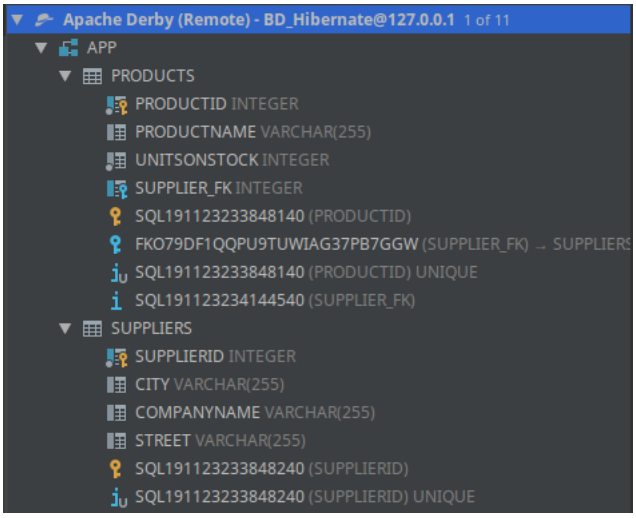


Rysunek 19. Schemat bazy danych

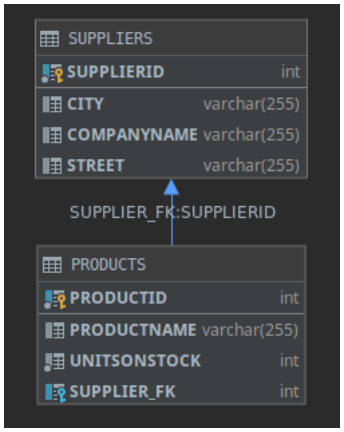
Brak tworzenia tabeli łącznikowej zapewniono dodając przed zbiorem produktów w klasie *Suppliers* adnotację @JoinColumn(name = "SUPPLIERS_FK"). Log Hibernate'a zawarto na Rysunku 20, strukturę bazy danych na Rysunku 21 a schemat bazy danych na Rsyunku 22.

```
Hibernate:

    create table Products (
       productID integer not null,
        productName varchar(255),
        unitsOnStock integer not null,
        SUPPLIER_FK integer,
        primary key (productID)
    )
Hibernate:

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

    alter table Products
       add constraint FKo79df1qqpu9tuwiag37pb7ggw
       foreign key (SUPPLIER_FK)
       references Suppliers
```

Rysunek 20. Log Hibernate dotyczący konstrukcji tabel z odwróconą zależnością



Rysunek 21. Struktura bazy danych widziana w DataGripie



Rysunek 22. Schemat bazy danych

5. Relacja dwukierunkowa

Dwukierunkowość relacji zapewniono dodając atrybut supplier do klasy Product i odpowiednio modyfikując metody ustawiające dostawcę/dodające produkt do listy dostarczanych. Klasa *Product* ma następującą postać:

```
@Entity
@Table(name = "Products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productID;
    private String productName;
    private int unitsOnStock;
    @ManyToOne
    @JoinColumn(name = "SUPPLIER_FK")
    private Supplier supplier;

    public Product() {
    }

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

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

a klasa *Supplier*:

```
@Entity
@Table(name = "Suppliers")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    private String companyName;
    private String street;
    private String city;
    @OneToMany(mappedBy = "supplier")
    private Set<Product> suppliedProducts;

    public Supplier() {
    }

    public Supplier(String companyName, String street, String city) {
        this.companyName = companyName;
        this.street = street;
        this.city = city;
        this.suppliedProducts = new HashSet<>();
    }

    public void addProductToList(Product addedProduct) {
        this.suppliedProducts.add(addedProduct);
        addedProduct.setSupplier(this);
    }

```

```
    public Set<Product> getSuppliedProducts() {
        return suppliedProducts;
    }
}
```

Log Hibernate przedstawia Rysunek 23, wyniki wywołania *select* na tabeli *Suppliers* i *Products* Rysunek 24 i 25, strukturę bazy danych Rysunek 26 a schemat bazy danych Rysunek 27.

```
Hibernate:

    create table Products (
       productID integer not null,
        productName varchar(255),
        unitsOnStock integer not null,
        SUPPLIER_FK integer,
        primary key (productID)
    )
Hibernate:

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

    alter table Products
       add constraint FKo79df1qqpu9tuwiag37pb7ggw
       foreign key (SUPPLIER_FK)
       references Suppliers
```

Rysunek 23. Log Hibernate dotyczący konstrukcji tabel z dwukierunkową zależnością

1 row | Tx: Auto

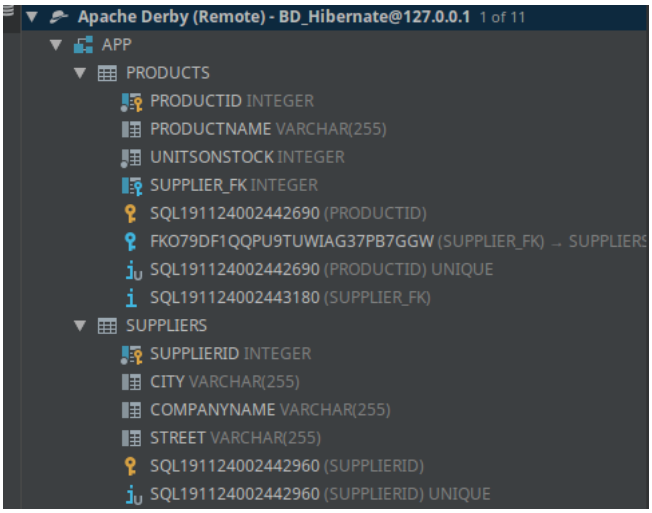
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	3	Warszawa	Hurtownia napojów SA	Wileńska 4a

Rysunek 24. Wynik polecenia *select* na tabeli *Suppliers*

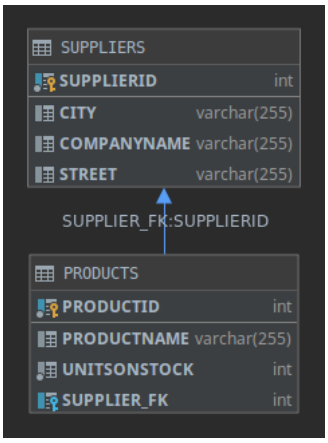
2 rows | Tx: Auto

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK
1	1	Kawa czarna	11	3
2	2	Herbata zielona	29	3

Rysunek 25. Wynik polecenia *select* na tabeli *Products*



Rysunek 26. Struktura bazy danych widziana w DataGripie



Rysunek 27. Schemat bazy danych

6. Klasa Category

Utworzono nową klasę *Category*:

```
@Entity
@Table(name = "Categories")
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int categoryID;
    private String categoryName;
    @OneToMany(mappedBy = "category")
    private Set<Product> products;

    public Category() {
    }

    public Category(String categoryName) {
        this.categoryName = categoryName;
        this.products = new HashSet<>();
    }

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

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

    @Override
    public String toString() {
        return categoryID + ": " + categoryName;
    }
}
```

Dodano także w klasie *Product* pomocnicze metody i mapowanie relacji w drugą stronę:

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

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

public Category getCategory() {
  return category;
}

@Override
public String toString() {
  return productID + ": " + productName + ", unitsOnStock: " + unitsOnStock;
}
```

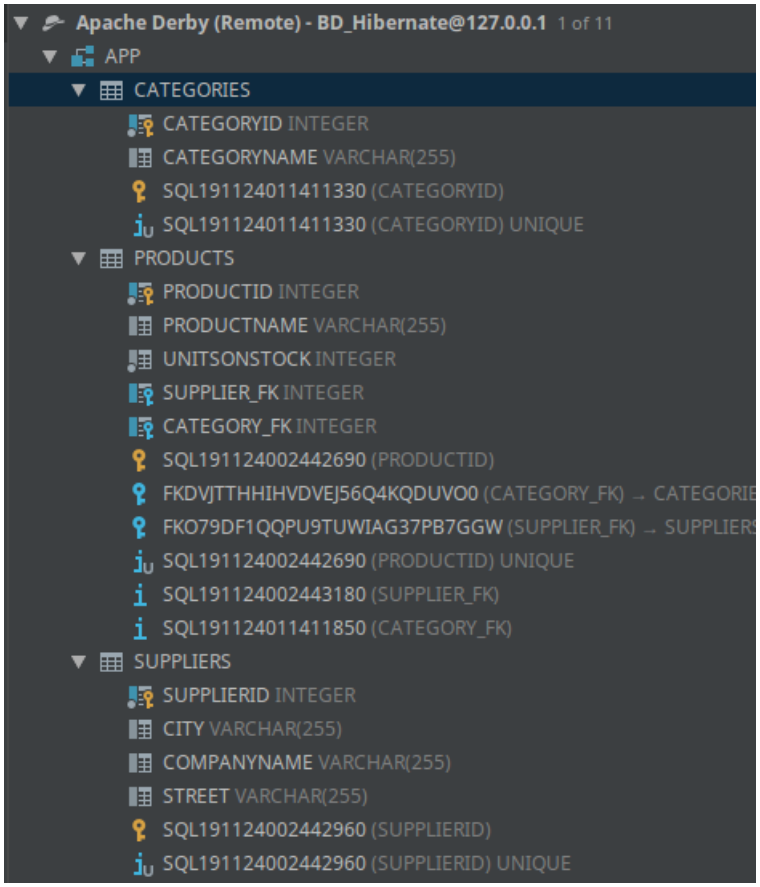
Logi Hibernate'a zaobserwowane przy tworzeniu nowej tabeli i modyfikacji istniejącej (Products) przedstawia Rysunek 28. Strukturę bazy danych przedstawia Rysunek 29, a jej schemat Rysunek 30. Log Hibernate'a dla dodawania nowych danych zawiera Rysunek 31. Po dodaniu kilku kategorii i przypi-

saniu dotychczas istniejących produktów do nich, wywołano polecenie *select* na tabeli *Products*. Wynik zamieszczono na Rysunku 32.

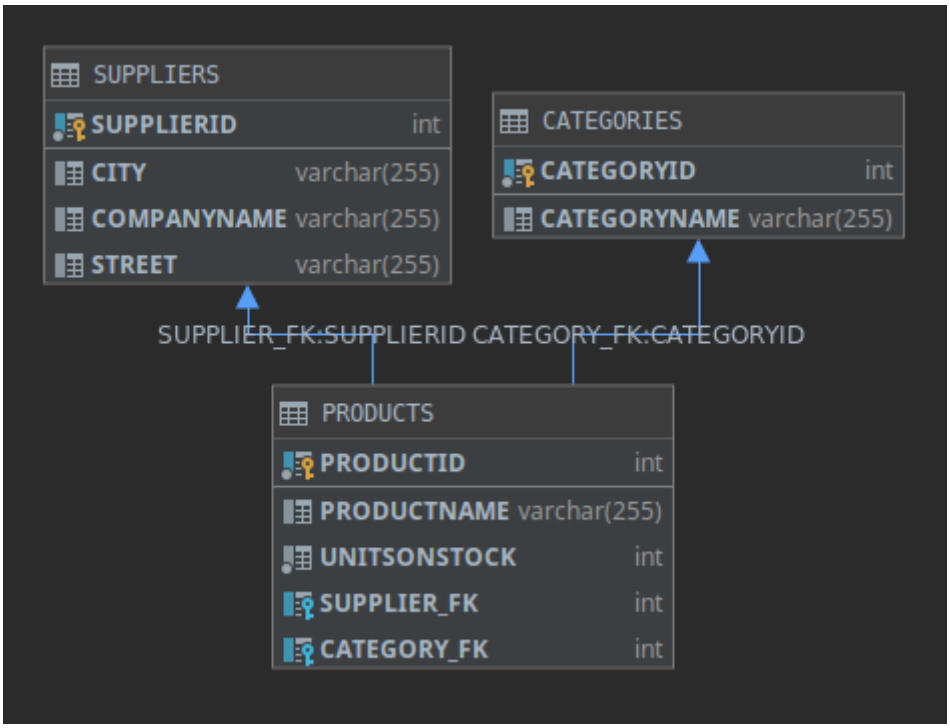
```
Hibernate:

    create table Categories (
       categoryID integer not null,
        categoryName varchar(255),
        primary key (categoryID)
    )
Hibernate:

    alter table Products
       add column CATEGORY_FK integer
Hibernate:

    alter table Products
       add constraint FKdvjtthhihvdvej56q4kqduvo0
       foreign key (CATEGORY_FK)
       references Categories
```

Rysunek 28. Log Hibernate dotyczący dodania tabeli *Categories* i modyfikacji *Products*

Wyciągnięto z poziomu maina jedną z kategorii i wypisano wszystkie produkty należące do niej, log przedstawia Rysunek 33. Następnie dla pojedynczego produktu wypisano kategorię, do której on należy – Rysunek 34.



Rysunek 29. Struktura bazy danych widziana w DataGripie



Rysunek 30. Schemat bazy danych

```
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    /* insert Category
        */ insert
        into
            Categories
            (categoryName, categoryID)
        values
            (?, ?)
Hibernate:
    /* insert Category
        */ insert
        into
            Categories
            (categoryName, categoryID)
        values
            (?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Products
            (CATEGORY_FK, productName, SUPPLIER_FK, unitsOnStock, productID)
        values
            (?, ?, ?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Products
            (CATEGORY_FK, productName, SUPPLIER_FK, unitsOnStock, productID)
        values
            (?, ?, ?, ?, ?)
Hibernate:
    /* insert Product
        */ insert
        into
            Products
            (CATEGORY_FK, productName, SUPPLIER_FK, unitsOnStock, productID)
        values
            (?, ?, ?, ?, ?)
```

Rysunek 31. Log Hibernate dotyczący dodawania nowych kategorii i produktów

5 rows Tx: Auto

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK	CATEGORY_FK
1	1	Kawa czarna	11	3	9
2	2	Herbata zielona	29	3	9
3	6	T-shirt	5	<null>	5
4	7	Sweter	12	<null>	5
5	8	Kurtka	9	<null>	5

Rysunek 32. Wynik polecenia *select* na tabeli *Products*

```
Hibernate:
    select
        category0_.categoryID as category1_0_0_,
        category0_.categoryName as category2_0_0_
    from
        Categories category0_
    where
        category0_.categoryID=?
Extracted category: 5: Ubrania
Hibernate:
    select
        products0_.CATEGORY_FK as CATEGORY4_3_0_,
        products0_.productID as productI1_3_0_,
        products0_.productID as productI1_3_1_,
        products0_.CATEGORY_FK as CATEGORY4_3_1_,
        products0_.productName as productN2_3_1_,
        products0_.SUPPLIER_FK as SUPPLIER5_3_1_,
        products0_.unitsOnStock as unitsOnS3_3_1_,
        supplier1_.supplierID as supplier1_4_2_,
        supplier1_.city as city2_4_2_,
        supplier1_.companyName as companyN3_4_2_,
        supplier1_.street as street4_4_2_
    from
        Products products0_
    left outer join
        Suppliers supplier1_
            on products0_.SUPPLIER_FK=supplier1_.supplierID
    where
        products0_.CATEGORY_FK=?
    7: Sweter, unitsOnStock: 12
    8: Kurtka, unitsOnStock: 9
    6: T-shirt, unitsOnStock: 5

Process finished with exit code 0
```

Rysunek 33. Log Hibernate dotyczący wyciągnięcia kategorii z bazy i wypisania należących do niej produktów

```
Hibernate:
    select
        product0_.productID as productI1_3_0_,
        product0_.CATEGORY_FK as CATEGORY4_3_0_,
        product0_.productName as productN2_3_0_,
        product0_.SUPPLIER_FK as SUPPLIER5_3_0_,
        product0_.unitsOnStock as unitsOnS3_3_0_,
        category1_.categoryID as category1_0_1_,
        category1_.categoryName as category2_0_1_,
        supplier2_.supplierID as supplier1_4_2_,
        supplier2_.city as city2_4_2_,
        supplier2_.companyName as companyN3_4_2_,
        supplier2_.street as street4_4_2_
    from
        Products product0_
    left outer join
        Categories category1_
            on product0_.CATEGORY_FK=category1_.categoryID
    left outer join
        Suppliers supplier2_
            on product0_.SUPPLIER_FK=supplier2_.supplierID
    where
        product0_.productID=?
Extracted product: 2: Herbata zielona, unitsOnStock: 29
Product category: 9: Napoje

Process finished with exit code 0
```

Rysunek 34. Log Hibernate dotyczący wyciągnięcia produktu z bazy i wypisania jego kategorii

7. Relacja wiele-do-wielu

Utworzono nową klasę *Invoice*:

```
@Entity
@Table(name = "Invoices")
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int invoiceNumber;
    private int quantity;
    @ManyToMany
    private Set<Product> products;

    public Invoice() {
    }

    public Invoice(int quantity) {
        this.quantity = quantity;
        this.products = new HashSet<>();
    }

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

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

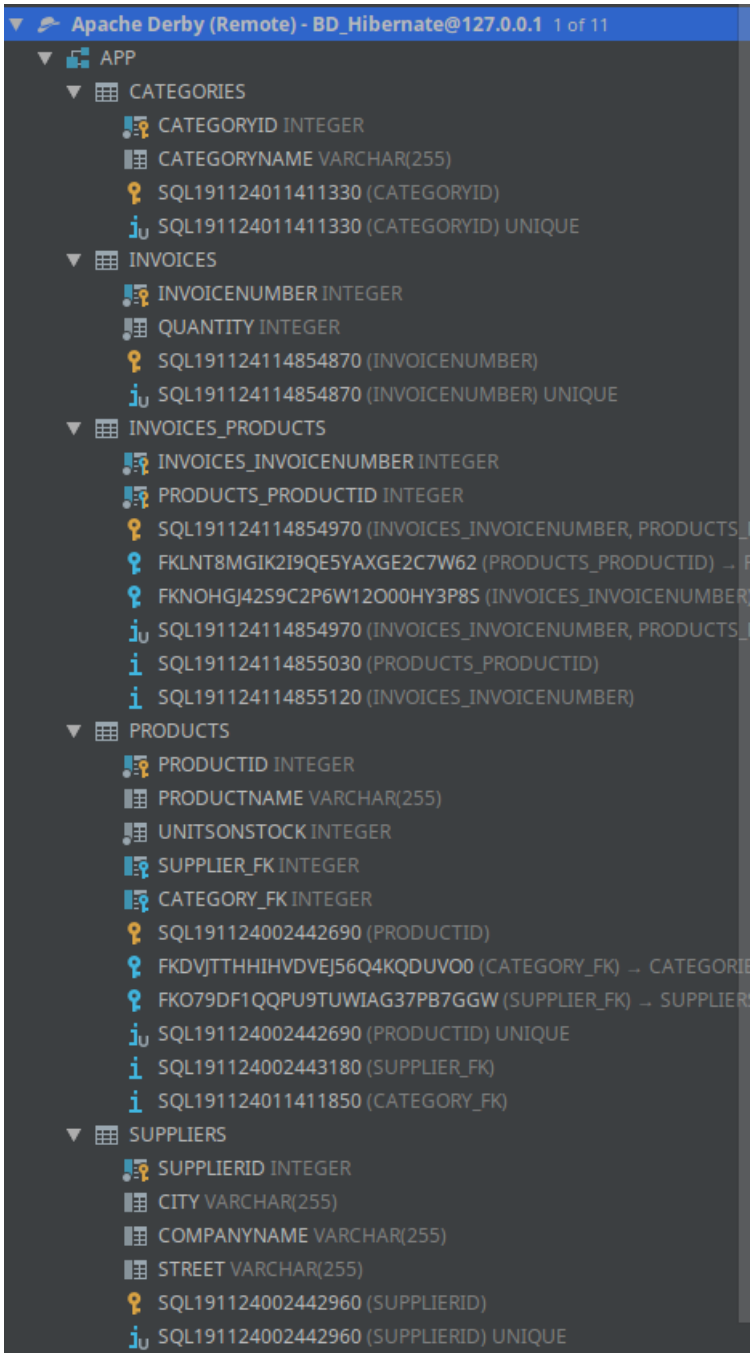
Do klasy *Product* dodano zbiór zamówień i metody obsługujące zamówienia:

```
@ManyToMany(mappedBy = "products")
private Set<Invoice> invoices;

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

public void decreaseUnitsOnStock(int quantity) {
  this.unitsOnStock -= quantity;
}
```

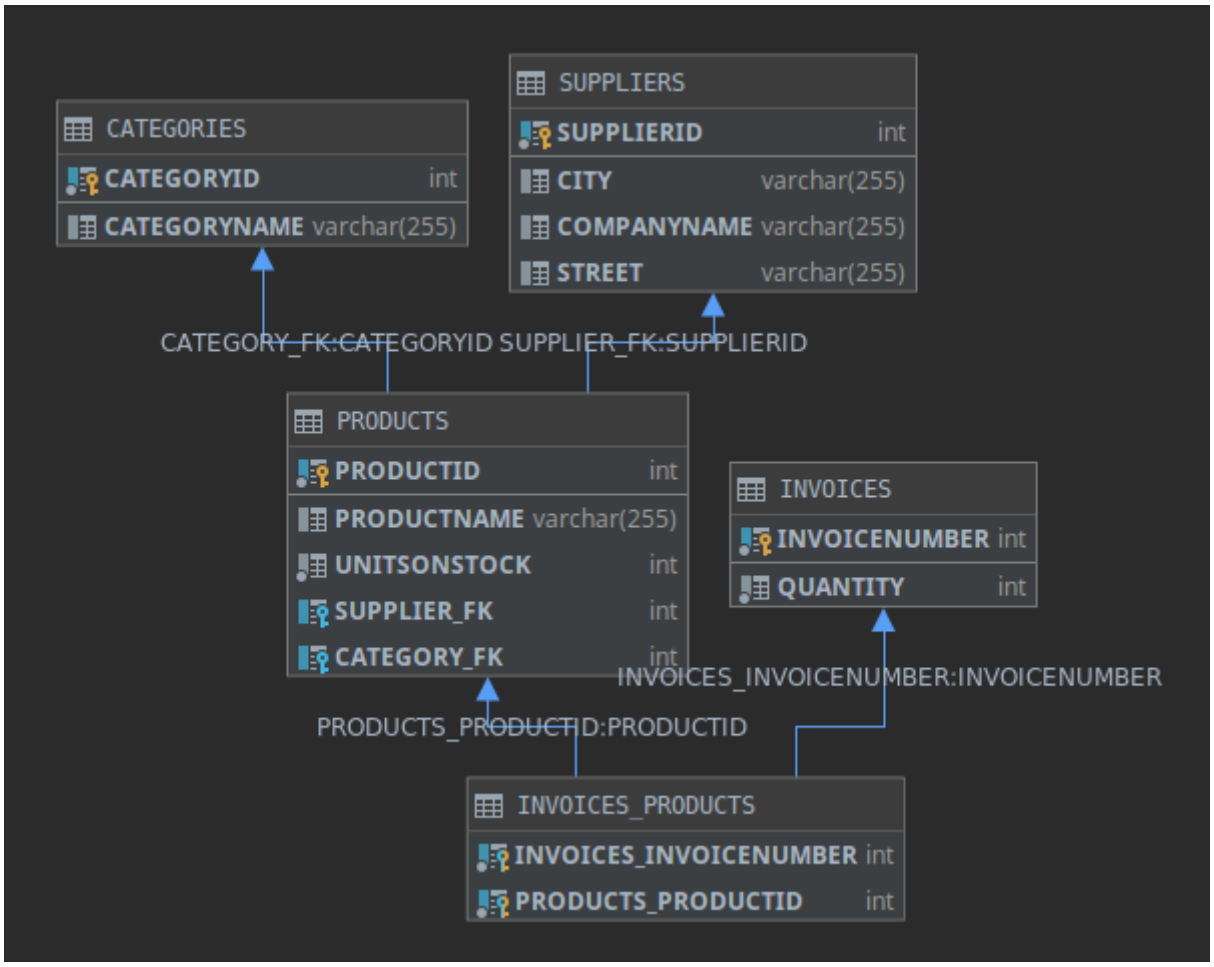
Po wywołaniu programu, zaobserwowano tworzenie nowej tabeli wraz z tabelą łącznikową – log zawiera Rysunek 35, strukturę bazy Rysunek 36 a schemat bazy Rysunek 37. Dodano kilka zamówień, nowych produktów i powiązano je ze sobą. Log dla wypisania produktów z konkretnego zamówienia przedstawia Rysunek 38, a dla wypisania zamówień dla konkretnego produktu Rysunek 39.

```
Hibernate:

    create table Invoices (
       invoiceNumber integer not null,
        quantity integer not null,
        primary key (invoiceNumber)
    )
Hibernate:

    create table Invoices_Products (
       invoices_invoiceNumber integer not null,
        products_productID integer not null,
        primary key (invoices_invoiceNumber, products_productID)
    )
Hibernate:

    alter table Invoices_Products
       add constraint FKlnt8mgik2i9qe5yaxge2c7w62
       foreign key (products_productID)
       references Products
Hibernate:

    alter table Invoices_Products
       add constraint FKnohgj42s9c2p6w12o00hy3p8s
       foreign key (invoices_invoiceNumber)
       references Invoices
```

Rysunek 35. Log Hibernate dotyczący dodania tabeli *Invoices* i tabeli łącznikowej



Rysunek 36. Struktura bazy danych widziana w DataGripie



Rysunek 37. Schemat bazy danych

```
Hibernate:
    select
        invoice0_.invoiceNumber as invoiceN1_1_0_,
        invoice0_.quantity as quantity2_1_0_
    from
        Invoices invoice0_
    where
        invoice0_.invoiceNumber=?
Extracted invoice: Invoice 110, total quantity: 15Hibernate:
    select
        products0_.invoices_invoiceNumber as invoices1_2_0_,
        products0_.products_productID as products2_2_0_,
        product1_.productID as productI1_3_1_,
        product1_.CATEGORY_FK as CATEGORY4_3_1_,
        product1_.productName as productN2_3_1_,
        product1_.SUPPLIER_FK as SUPPLIER5_3_1_,
        product1_.unitsOnStock as unitsOnS3_3_1_,
        category2_.categoryID as category1_0_2_,
        category2_.categoryName as category2_0_2_,
        supplier3_.supplierID as supplier1_4_3_,
        supplier3_.city as city2_4_3_,
        supplier3_.companyName as companyN3_4_3_,
        supplier3_.street as street4_4_3_
    from
        Invoices_Products products0_
    inner join
        Products product1_
            on products0_.products_productID=product1_.productID
    left outer join
        Categories category2_
            on product1_.CATEGORY_FK=category2_.categoryID
    left outer join
        Suppliers supplier3_
            on product1_.SUPPLIER_FK=supplier3_.supplierID
    where
        products0_.invoices_invoiceNumber=?
    106: Cebula, unitsOnStock: 10
    107: Szczypiorek, unitsOnStock: 10
    108: Pomidor, unitsOnStock: 30

Process finished with exit code 0
```

Rysunek 38. Log Hibernate dotyczący wyciągnięcia zamówienia z bazy i wypisania należących do niego produktów

```
Hibernate:
    select
        product0_.productID as productI1_3_0_,
        product0_.CATEGORY_FK as CATEGORY4_3_0_,
        product0_.productName as productN2_3_0_,
        product0_.SUPPLIER_FK as SUPPLIER5_3_0_,
        product0_.unitsOnStock as unitsOnS3_3_0_,
        category1_.categoryID as category1_0_1_,
        category1_.categoryName as category2_0_1_,
        supplier2_.supplierID as supplier1_4_2_,
        supplier2_.city as city2_4_2_,
        supplier2_.companyName as companyN3_4_2_,
        supplier2_.street as street4_4_2_
    from
        Products product0_
    left outer join
        Categories category1_
            on product0_.CATEGORY_FK=category1_.categoryID
    left outer join
        Suppliers supplier2_
            on product0_.SUPPLIER_FK=supplier2_.supplierID
    where
        product0_.productID=?
Extracted product: 106: Cebula, unitsOnStock: 8
Hibernate:
    select
        invoices0_.products_productID as products2_2_0_,
        invoices0_.invoices_invoiceNumber as invoices1_2_0_,
        invoice1_.invoiceNumber as invoiceN1_1_1_,
        invoice1_.quantity as quantity2_1_1_
    from
        Invoices_Products invoices0_
    inner join
        Invoices invoice1_
            on invoices0_.invoices_invoiceNumber=invoice1_.invoiceNumber
    where
        invoices0_.products_productID=?
    Invoice 109, total quantity: 7
    Invoice 110, total quantity: 15

Process finished with exit code 0
```

Rysunek 39. Log Hibernate dotyczący wyciągnięcia produktu z bazy i wypisania zamówień na niego

8. JPA

W celu przejścia na korzystanie z JPA, w katalogu źródłowym projektu utworzono katalog META--INF, a w nim plik konfiguracyjny *persistence.xml*:

```
<?xml version='1.0' encoding='utf-8'?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
    <persistence-unit name="DBHibernateConfig"
                      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/BD_Hibernate"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```

Zdefiniowano nową klasę główną *MainJPA*, w metodzie *main* utworzono dwie nowe kategorie, przypisano do istniejących produktów bez kategorii i wypisano produkty jednej z kategorii (Rysunek 40) i kategorię jednego produktu (Rysunek 41), tak jak w punkcie VI.

Kod klasy:

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

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

        Category foodCategory = new Category("Żywność");
        em.persist(foodCategory);

        Product food1 = em.find(Product.class, 106);
        food1.setCategory(foodCategory);
        Product food2 = em.find(Product.class, 107);
        food2.setCategory(foodCategory);
        Product food3 = em.find(Product.class, 108);
        food3.setCategory(foodCategory);

        Category washCategory = new Category("Środki czystości");
        em.persist(washCategory);

        Product washingProduct1 = em.find(Product.class, 104);
        washingProduct1.setCategory(washCategory);
        Product washingProduct2 = em.find(Product.class, 105);
        washingProduct2.setCategory(washCategory);

        Category category = em.find(Category.class, 5);
```

```
32          System.out.println("Extracted category: " + category);
33
34          for(Product nextProduct : category.getProducts()) {
35              System.out.printf("\t%s\n", nextProduct);
36          }
37
38          Product product = em.find(Product.class, 2);
39          System.out.println("Extracted product: " + product);
40          System.out.println("Product category: " + product.getCategory());
41
42          etx.commit();
43          em.close();
44      }
45  }
```

```
Hibernate:
    select
        category0_.categoryID as category1_0_0_,
        category0_.categoryName as category2_0_0_
    from
        Categories category0_
    where
        category0_.categoryID=?
Extracted category: 5: Ubrania
Hibernate:
    select
        products0_.CATEGORY_FK as CATEGORY4_3_0_,
        products0_.productID as productI1_3_0_,
        products0_.productID as productI1_3_1_,
        products0_.CATEGORY_FK as CATEGORY4_3_1_,
        products0_.productName as productN2_3_1_,
        products0_.SUPPLIER_FK as SUPPLIER5_3_1_,
        products0_.unitsOnStock as unitsOnS3_3_1_,
        supplier1_.supplierID as supplier1_4_2_,
        supplier1_.city as city2_4_2_,
        supplier1_.companyName as companyN3_4_2_,
        supplier1_.street as street4_4_2_
    from
        Products products0_
    left outer join
        Suppliers supplier1_
            on products0_.SUPPLIER_FK=supplier1_.supplierID
    where
        products0_.CATEGORY_FK=?
    8: Kurtka, unitsOnStock: 9
    7: Sweter, unitsOnStock: 12
    6: T-shirt, unitsOnStock: 5

Process finished with exit code 0
```

Rysunek 40. Log Hibernate dotyczący wyciągnięcia kategorii z bazy i wypisania należących do niej produktów

```
Hibernate:
    select
        product0_.productID as productI1_3_0_,
        product0_.CATEGORY_FK as CATEGORY4_3_0_,
        product0_.productName as productN2_3_0_,
        product0_.SUPPLIER_FK as SUPPLIER5_3_0_,
        product0_.unitsOnStock as unitsOnS3_3_0_,
        category1_.categoryID as category1_0_1_,
        category1_.categoryName as category2_0_1_,
        supplier2_.supplierID as supplier1_4_2_,
        supplier2_.city as city2_4_2_,
        supplier2_.companyName as companyN3_4_2_,
        supplier2_.street as street4_4_2_
    from
        Products product0_
    left outer join
        Categories category1_
            on product0_.CATEGORY_FK=category1_.categoryID
    left outer join
        Suppliers supplier2_
            on product0_.SUPPLIER_FK=supplier2_.supplierID
    where
        product0_.productID=?
Extracted product: 2: Herbata zielona, unitsOnStock: 29
Product category: 9: Napoje

Process finished with exit code 0
```

Rysunek 41. Log Hibernate dotyczący wyciągnięcia produktu z bazy i wypisania jego kategorii

9. Kaskady

W klasie *Invoices* zmieniono adnotację przed zbiorem produktów na:

```
@ManyToMany(cascade = {CascadeType.PERSIST})
private Set<Product> products;
```

analogicznie w klasie *Product*:

```
@ManyToMany(mappedBy = "products", cascade = CascadeType.PERSIST)
private Set<Invoice> invoices;
```

Po wywołaniu funkcji *main*:

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

  Invoice newInvoice = new Invoice(0);
  Product product1 = new Product("Paluszki solone", 3);
  Product product2 = new Product("Orzeszki ziemne", 5);

  newInvoice.addProduct(product1, 1);
  newInvoice.addProduct(product2, 3);

  em.persist(newInvoice);

  Product newProduct = new Product("Piłka do siatkówki", 20);

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

  newProduct.addInvoice(invoice1, 2);
  newProduct.addInvoice(invoice2, 5);

  em.persist(newProduct);

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

Zaobserwowano dodawanie wszystkich utworzonych elementów. Wynik wywołania *select* na tabeli *Products* przedstawia Rysunek 42, na tabeli łącznikowej Rysunek 43. Faktury utworzone w drugiej części programu (dodane do zbioru wewnątrz produktu) nie są mapowane do tabeli łącznikowej, ze względu na to, że właścicielem tej tabeli jest klasa *Invoice* a nie *Products*.

13 rows | Tx: Auto

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK	CATEGORY_FK
1	1	Kawa czarna	11	3	9
2	2	Herbata zielona	29	3	9
3	6	T-shirt	5	<null>	5
4	7	Sweter	12	<null>	5
5	8	Kurtka	9	<null>	5
6	104	Szczoteczka do zębów	1	<null>	112
7	105	Pasta do zębów	7	<null>	112
8	106	Cebula	8	<null>	111
9	107	Szczypiorek	10	<null>	111
10	108	Pomidor	30	<null>	111
11	133	Orzeszki ziemne	2	<null>	<null>
12	134	Paluszki solone	2	<null>	<null>
13	135	Piłka do siatkówki	13	<null>	<null>

Rysunek 42. Wynik polecenia *select* na tabeli *Products*

8 rows | Tx: Auto

	INVOICES_INVOICENUMBER	PRODUCTS_PRODUCTID
1	109	104
2	109	105
3	109	106
4	110	106
5	110	107
6	110	108
7	132	133
8	132	134

Rysunek 43. Wynik polecenia *select* na tabeli łącznikowej

10. Embedded class

Utworzono nową klasę *Address*:

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

    public Address() {
    }

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

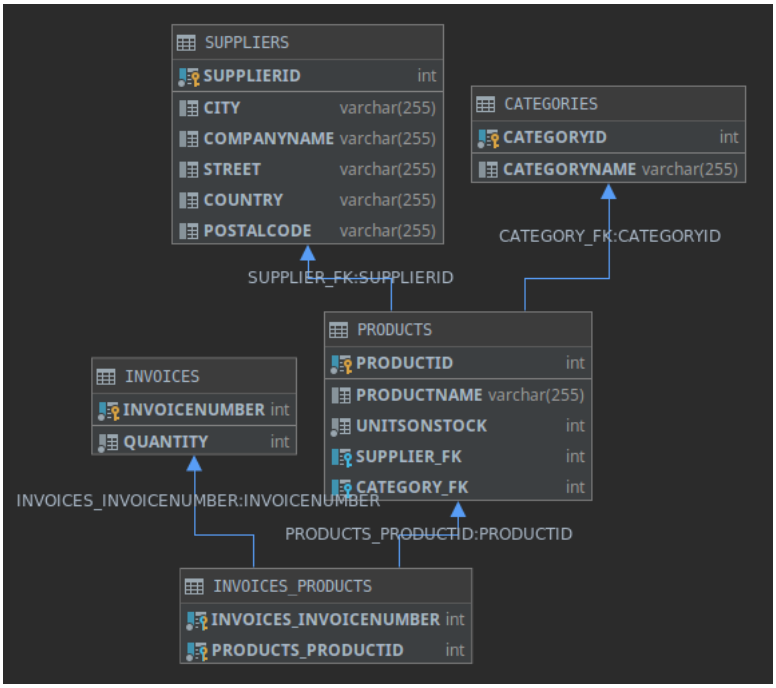
W klasie *Suppliers* dodano nowy atrybut w zamian za poprzednie odpowiadające adresowi:

```
@Entity
@Table(name = "Suppliers")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    private String companyName;
    @Embedded
    private Address address;
    @OneToMany(mappedBy = "supplier")
    private Set<Product> suppliedProducts;

    public Supplier() {
    }

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

        this.suppliedProducts = new HashSet<>();
    }

    public void addProductToList(Product addedProduct) {
        this.suppliedProducts.add(addedProduct);
        addedProduct.setSupplier(this);
    }

    public Set<Product> getSuppliedProducts() {
        return suppliedProducts;
    }
}
```

Po uruchomieniu programu zaobserwowano dodawanie nowych kolumn do tabeli *Suppliers*. Log Hibernate'a przedstawia Rysunek 44 a schemat bazy danych Rysunek 45.

```
Hibernate:

    alter table Suppliers
       add column country varchar(255)
Hibernate:

    alter table Suppliers
       add column postalCode varchar(255)
```

Rysunek 44. Log Hibernate dotyczący dodawania nowych kolumn



Rysunek 45. Schemat bazy danych

Zmodyfikowano klasę *Supplier* tak, aby była mapowana do dwóch tabel:

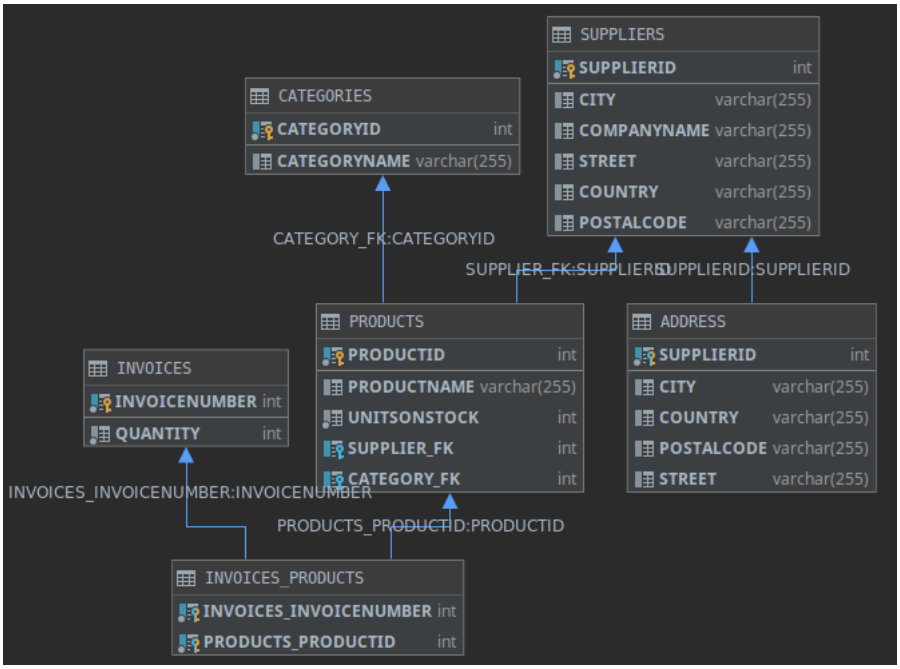
```
@Entity
@Table(name = "Suppliers")
@SecondaryTable(name = "Address")
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    private String companyName;
    @Column(table = "Address")
    private String street;
    @Column(table = "Address")
    private String postalCode;
    @Column(table = "Address")
    private String city;
    @Column(table = "Address")
    private String country;
    @OneToMany(mappedBy = "supplier")
    private Set<Product> suppliedProducts;

    public Supplier() {
    }

    public Supplier(String companyName, String street, String postalCode, String city, String country){
        this.companyName = companyName;
        this.street = street;
```

```
26          this.postalCode = postalCode;
27          this.city = city;
28          this.country = country;
29
30          this.suppliedProducts = new HashSet<>();
31      }
32
33      public void addProductToList(Product addedProduct) {
34          this.suppliedProducts.add(addedProduct);
35          addedProduct.setSupplier(this);
36      }
37
38      public Set<Product> getSuppliedProducts() {
39          return suppliedProducts;
40      }
41  }
```

Log Hibernate'a załączono na Rysunku 46, a schemat bazy danych na Rysunku 47.

```
Hibernate:

    create table Address (
       city varchar(255),
        country varchar(255),
        postalCode varchar(255),
        street varchar(255),
        supplierID integer not null,
        primary key (supplierID)
    )
Hibernate:

    alter table Address
       add constraint FK8nms75ofomr1o9v8fpsmscxol
       foreign key (supplierID)
       references Suppliers
```

Rysunek 46. Log Hibernate dotyczący mapowania klasy *Supplier* do dwóch różnych tabel



Rysunek 47. Schemat bazy danych

11. Dziedziczenie

11.1. Jedna tabela na całą hierarchię

Utworzono abstrakcyjną klasę *Company*:

```
@Entity
@Table(name = "Companies")
@SecondaryTable(name = "Address")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int companyID;
    private String companyName;
    @Column(table = "Address")
    private String street;
    @Column(table = "Address")
    private String postalCode;
    @Column(table = "Address")
    private String city;
    @Column(table = "Address")
    private String country;

    public Company() {
    }

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

    @Override
    public String toString() {
        return "Company name: " + companyName
                + ", address: "
                + street + " " + postalCode
                + " " + city + ", " + country;
    }
}
```

Zmodyfikowana klasa *Supplier* ma postać:

```
@Entity
@DiscriminatorValue(value = "S")
public class Supplier extends Company {
    private String bankAccountNumber;
    @OneToMany(mappedBy = "supplier")
    private Set<Product> suppliedProducts;

    public Supplier() {
        super();
    }

    public Supplier(String companyName, String street, String postalCode, String city, String country,
      String bankAccountNumber) {
        super(companyName, street, postalCode, city, country);

        this.bankAccountNumber = bankAccountNumber;
```

```
        this.suppliedProducts = new HashSet<>();
    }

    public void addProductToList(Product addedProduct) {
        this.suppliedProducts.add(addedProduct);
        addedProduct.setSupplier(this);
    }

    public Set<Product> getSuppliedProducts() {
        return suppliedProducts;
    }

    @Override
    public String toString() {
        return super.toString()
                + ", bank account number: " + bankAccountNumber;
    }
}
```

a nowa klasa *Customer*:

```
@Entity
@DiscriminatorValue(value = "C")
public class Customer extends Company {
    private double discount;

    public Customer() {
        super();
    }

    public Customer(String companyName, String street, String postalCode, String city, String country,
      double discount) {
        super(companyName, street, postalCode, city, country);

        this.discount = discount;
    }

    public double getDiscount() {
        return discount;
    }

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

    @Override
    public String toString() {
        return super.toString()
                + ", discount: " + discount;
    }
}
```

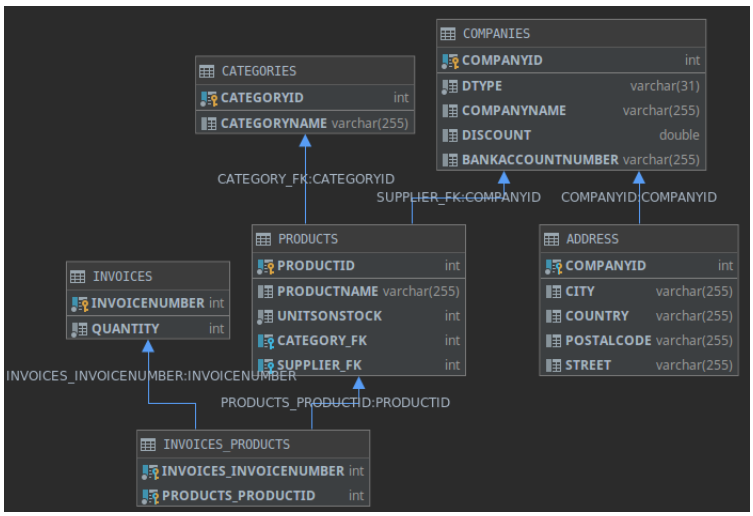
Log dla operacji tworzenia tabel zawiera Rysunek 48, schemat bazy danych Rysunek 49, a wynik polecenia *select* na tabeli *Companies* Rysunek 50.

Wypisano wszystkich klientów korzystając z kodu (wynik przedstawia Rysunek 51):

```
List<Customer> customers = em.createQuery("from Customer").getResultList();
for(Customer c : customers) {
  System.out.println(c);
}
```

```
Hibernate:

    create table Address (
       city varchar(255),
        country varchar(255),
        postalCode varchar(255),
        street varchar(255),
        companyID integer not null,
        primary key (companyID)
    )
Hibernate:

    create table Categories (
       categoryID integer not null,
        categoryName varchar(255),
        primary key (categoryID)
    )
Hibernate:

    create table Companies (
       DTYPE varchar(31) not null,
        companyID integer not null,
        companyName varchar(255),
        discount double,
        bankAccountNumber varchar(255),
        primary key (companyID)
    )
```

Rysunek 48. Log Hibernate dotyczący tworzenia nowych tabel w hierarchii dziedziczenia



Rysunek 49. Schemat bazy danych

4 rows Tx: Auto

	DTYPE	COMPANYID	COMPANYNAME	DISCOUNT	BANKACCOUNTNUMBER
1	C	1	Janusz sp. z.o.o.	30.3	<null>
2	C	2	Andrzej	12.3	<null>
3	S	3	Dostawca pizzy	<null>	12 2345 5678
4	S	4	Dostawca perfum	<null>	33 2245 6654

Rysunek 50. Wynik polecenia *select* na tabeli *Companies*

```
Hibernate:
    select
        customer0_.companyID as companyI2_2_,
        customer0_.companyName as companyN3_2_,
        customer0_1_.city as city1_0_,
        customer0_1_.country as country2_0_,
        customer0_1_.postalCode as postalCo3_0_,
        customer0_1_.street as street4_0_,
        customer0_.discount as discount4_2_
    from
        Companies customer0_
    left outer join
        Address customer0_1_
            on customer0_.companyID=customer0_1_.companyID
    where
        customer0_.DTYPE='C'
Company name: Janusz sp. z.o.o., address: al. Mickiewicza 20a 30-000 Kraków, Polska, discount: 30.3
Company name: Andrzej, address: Telimeny 25 36-098 Kraków, Polska, discount: 12.3

Process finished with exit code 0
```

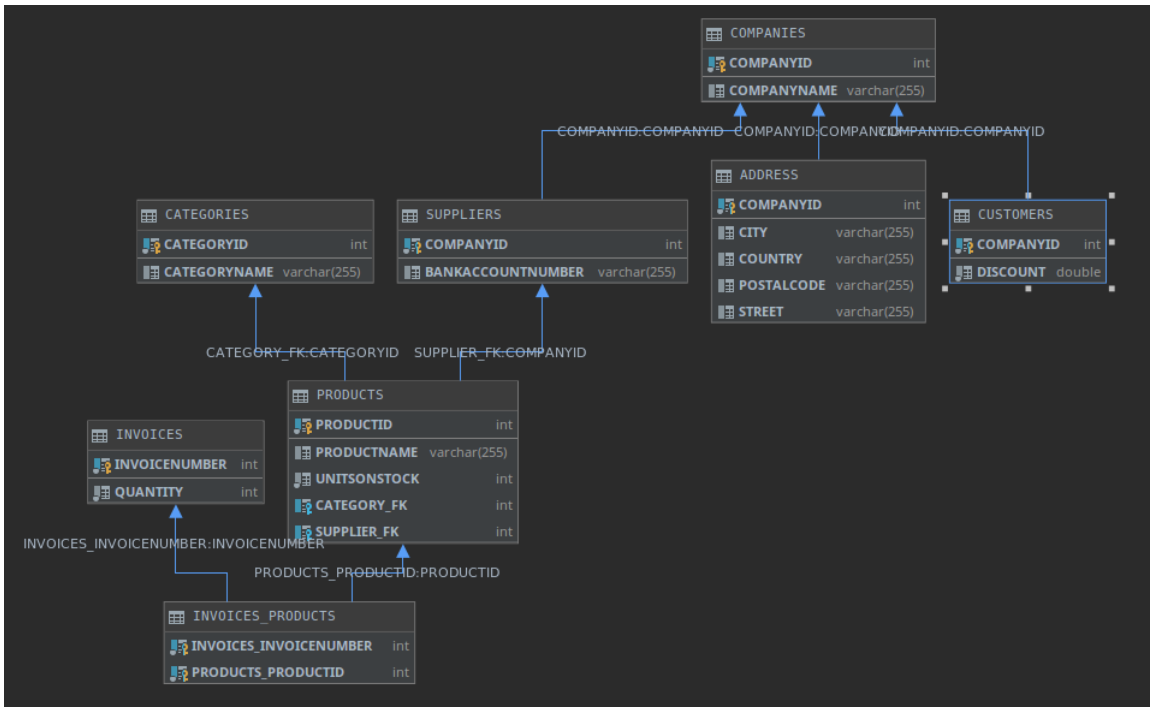
Rysunek 51. Wynik wypisania wszystkich klientów

11.2. Tabele łączone

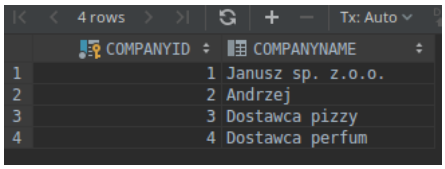
Zmieniono adnotację stojącą przed deklaracją klasy *Company* na @Inheritance(strategy = InheritanceType.JOINED), usunięto adnotacje @DiscriminatorValue i dodano adnotacje @Table do klas *Supplier* i *Customer* określające nazwy tabel. Dalej wykonano analogiczne kroki jak w poprzednim podpunkcie. Log dla operacji tworzenia tabel zawiera Rysunek 52, schemat bazy danych Rysunek 53, wynik polecenia *select* na tabeli *Companies* Rysunek 54, a wynik wypisania wszystkich klientów Rysunek 55.

```
Hibernate:

    create table Address (
       city varchar(255),
        country varchar(255),
        postalCode varchar(255),
        street varchar(255),
        companyID integer not null,
        primary key (companyID)
    )
Hibernate:

    create table Categories (
       categoryID integer not null,
        categoryName varchar(255),
        primary key (categoryID)
    )
Hibernate:

    create table Companies (
       companyID integer not null,
        companyName varchar(255),
        primary key (companyID)
    )
Hibernate:

    create table Customers (
       discount double not null,
        companyID integer not null,
        primary key (companyID)
    )
```

Rysunek 52. Log Hibernate dotyczący tworzenia nowych tabel w hierarchii dziedziczenia



Rysunek 53. Schemat bazy danych



4 rows Tx: Auto

	COMPANYID	COMPANYNAME
1	1	Janusz sp. z.o.o.
2	2	Andrzej
3	3	Dostawca pizzy
4	4	Dostawca perfum

Rysunek 54. Wynik polecenia *select* na tabeli *Companies*

```
Hibernate:
    select
        customer0_.companyID as companyI1_2_,
        customer0_1_.companyName as companyN2_2_,
        customer0_2_.city as city1_0_,
        customer0_2_.country as country2_0_,
        customer0_2_.postalCode as postalCo3_0_,
        customer0_2_.street as street4_0_,
        customer0_.discount as discount1_3_
    from
        Customers customer0_
    inner join
        Companies customer0_1_
            on customer0_.companyID=customer0_1_.companyID
    left outer join
        Address customer0_2_
            on customer0_.companyID=customer0_2_.companyID
Company name: Janusz sp. z.o.o., address: al. Mickiewicza 20a 30-000 Kraków, Polska, discount: 30.3
Company name: Andrzej, address: Telimeny 25 36-098 Kraków, Polska, discount: 12.3

Process finished with exit code 0
```

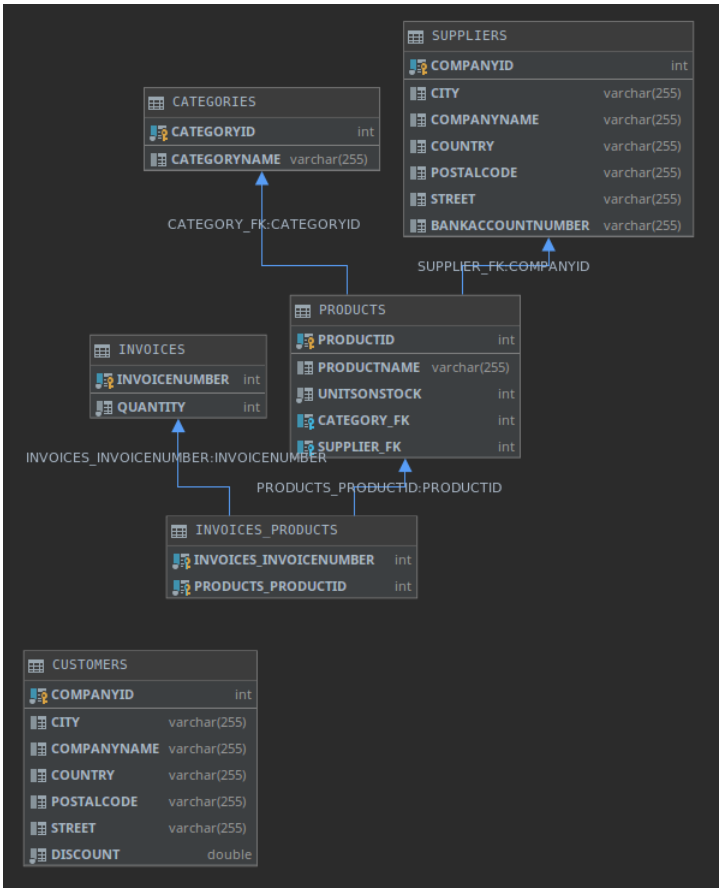
Rysunek 55. Wynik wypisania wszystkich klientów

11.3. Jedna tabela na klasę

Zmieniono adnotację stojącą przed deklaracją klasy *Company* na @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS). Usunięto z tej klasy adnotacje służące do tworzenia osobnej tabeli na adresy. Dalej wykonano analogiczne kroki jak w poprzednim podpunkcie. Log dla operacji tworzenia tabel zawiera Rysunek 56, schemat bazy danych Rysunek 57, wynik polecenia *select* na tabeli *Companies* Rysunek 58, a wynik wypisania wszystkich klientów Rysunek 59.

```
Hibernate:

    create table Categories (
       categoryID integer not null,
        categoryName varchar(255),
        primary key (categoryID)
    )
Hibernate:

    create table Customers (
       companyID integer not null,
        city varchar(255),
        companyName varchar(255),
        country varchar(255),
        postalCode varchar(255),
        street varchar(255),
        discount double not null,
        primary key (companyID)
    )
Hibernate:

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

Rysunek 56. Log Hibernate dotyczący tworzenia nowych tabel w hierarchii dziedziczenia



Rysunek 57. Schemat bazy danych

2 rows Tx: Auto

	COMPANYID	CITY	COMPANYNAME	COUNTRY	POSTALCODE	STREET	BANKACCOUNTNUMBER
1	3	Warszawa	Dostawca pizzy	Polska	00-304	ul. Wileńska 4	12 2345 5678
2	4	Gdańsk	Dostawca perfum	Polska	23-567	ul. Długa 12	33 2245 6654

Rysunek 58. Wynik polecenia *select* na tabeli *Companies*

```
Hibernate:
    select
        customer0_.companyID as companyI1_1_,
        customer0_.city as city2_1_,
        customer0_.companyName as companyN3_1_,
        customer0_.country as country4_1_,
        customer0_.postalCode as postalCo5_1_,
        customer0_.street as street6_1_,
        customer0_.discount as discount1_2_
    from
        Customers customer0_
Company name: Janusz sp. z.o.o., address: al. Mickiewicza 20a 30-000 Kraków, Polska, discount: 30.3
Company name: Andrzej, address: Telimeny 25 36-098 Kraków, Polska, discount: 12.3

Process finished with exit code 0
```

Rysunek 59. Wynik wypisania wszystkich klientów

12. Aplikacja do zamawiania produktów

Do aplikacji dodano klasy odpowiedzialne za wyświetlanie menu konsolowego (Menu) i logowania (Logger):

```java
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.persistence.TypedQuery;
import java.util.List;
import java.util.Scanner;

public class Logger {
    private EntityManager em;
    private Scanner scanner;

    public Logger(EntityManager em, Scanner scanner) {
        this.em = em;
        this.scanner = scanner;
    }

    public Customer loginCompany() {
        System.out.println("Hello to Java Hibernate App.");
        System.out.printf("\tL - log in as an existing user\n\tR - create new account\n");
        System.out.printf("\tAny other key - exit\n");
        System.out.print("Response: ");
        String resp = scanner.nextLine().toLowerCase();

        if (resp.equals("l")) {
            return authenticateUser();
        } else if (resp.equals("r")) {
            return createNewUser();
        } else {
            return null;
        }
    }

    private Customer authenticateUser() {
        System.out.print("\tCompany name: ");
        String companyName = scanner.nextLine();

        TypedQuery<Customer> companyQuery = em.createQuery("from Customer
          as customer where customer.companyName = :compName", Customer.class);
        companyQuery.setParameter("compName", companyName);

        List<Customer> result = companyQuery.getResultList();
        if (result.size() != 1) {
            System.out.println("Authentication failed");
            System.exit(1);
        }

        System.out.println("Authentication successfull");
        return result.get(0);

    }

    private Customer createNewUser() {
        System.out.print("Enter company name: ");
        String companyName = scanner.nextLine();

        TypedQuery<Customer> companyQuery = em.createQuery("from Customer
          as company where company.companyName = :compName", Customer.class);
        companyQuery.setParameter("compName", companyName);
```

```java

        List<Customer> result = companyQuery.getResultList();
        if (result.size() != 0) {
            System.out.println("Company with this name already exists!");
            System.exit(1);
        }

        System.out.print("Company street: ");
        String street = scanner.nextLine();
        System.out.print("Company postal code: ");
        String postalCode = scanner.nextLine();
        System.out.print("Company city: ");
        String city = scanner.nextLine();
        System.out.print("Company country: ");
        String country = scanner.nextLine();

        Customer registrationResult = new Customer(companyName,
          street, postalCode, city, country, 0.0);

        EntityTransaction etx = em.getTransaction();
        etx.begin();
        em.persist(registrationResult);
        etx.commit();
        ;

        return registrationResult;
    }
}
```

```java
import java.util.HashMap;
import java.util.Scanner;
import java.util.function.Consumer;

public class Menu {
    private String entryText;
    private HashMap<Integer, MenuEntry> options;
    private Integer index;
    private boolean continueAction = true;

    public Menu(String entryText) {
        this.entryText = entryText;
        this.index = new Integer(1);
        this.options = new HashMap<>();
    }

    public void addOption(String description, Consumer<Scanner> action) {
        this.options.put(this.index, new MenuEntry(description, action));
        this.index += 1;
    }

    public void display() {
        System.out.println(this.entryText);

        Scanner inputScanner = new Scanner(System.in);
        while (this.continueAction) {
            for (int i = 1; i < index; i++) {
                System.out.printf("%d - %s\n", i, this.options.get(i).getDescription());
            }

            System.out.print("Enter option number: ");
            Integer chosenOption = Integer.parseInt(inputScanner.nextLine());

```

```
34            if (this.options.containsKey(chosenOption)) {
35                this.options.get(chosenOption).getEntryAction().accept(inputScanner);
36            } else {
37                System.out.println("Wrong action!");
38            }
39        }
40
41    }
42 }
```

Wykorzystywana jest także pomocnicza klasa MenuEntry:

```
1  import java.util.Scanner;
2  import java.util.function.Consumer;
3
4  public class MenuEntry {
5      private String description;
6      private Consumer<Scanner> entryAction;
7
8      public MenuEntry(String description, Consumer<Scanner> entryAction) {
9          this.description = description;
10         this.entryAction = entryAction;
11     }
12
13     public String getDescription() {
14         return description;
15     }
16
17     public Consumer<Scanner> getEntryAction() {
18         return entryAction;
19     }
20 }
```

W głównym programie uruchamiany jest mechanizm logowania/rejestracji nowego użytkownika, a następnie menu z opcjami: obejrzenia zamówień i złożenia nowego zamówienia:

```
1  import javax.persistence.*;
2  import java.util.LinkedList;
3  import java.util.List;
4  import java.util.Scanner;
5
6  public class MainApp {
7      public static Customer loggedUser;
8      public static EntityManager em;
9
10     public static void main(String[] args) {
11         EntityManagerFactory emf = Persistence.createEntityManagerFactory("DBHibernateConfig");
12         em = emf.createEntityManager();
13
14         Scanner inputScanner = new Scanner(System.in);
15         Logger logger = new Logger(em, inputScanner);
16         loggedUser = logger.loginCompany();
17
18         Menu menu;
19
20         menu = new Menu("Hello customer");
21         menu.addOption("show orders", MainApp::listOrders);
22         menu.addOption("make new order", MainApp::makeNewOrder);
23         menu.addOption("exit", x -> System.exit(0));
24         menu.display();
25
26         em.close();
```

```
27      }
28
29      public static void listOrders(Scanner scanner) {
30          for (Invoice currentInvoice : loggedUser.getInvoices()) {
31              System.out.println("Order number: " + currentInvoice.getInvoiceNumber());
32              System.out.printf("Quantity: " + currentInvoice.getQuantity());
33              for (Product invoiceProduct : currentInvoice.getProducts()) {
34                  System.out.printf("\t%s\n", invoiceProduct.getProductName());
35              }
36          }
37      }
38
39      public static void listProducts() {
40          TypedQuery<Product> query = em.createQuery("from Product", Product.class);
41          List<Product> products = query.getResultList();
42
43          System.out.printf("ID\t\tNAME\t\tUNITS ON STOCK\n");
44          for (Product currentProduct : products) {
45              System.out.printf("%d\t\t%s\t\t%d\n", currentProduct.getProductID(),
46                currentProduct.getProductName(), currentProduct.getUnitsOnStock());
47          }
48      }
49
50      public static void makeNewOrder(Scanner scanner) {
51          Invoice newInvoice = new Invoice(0);
52          listProducts();
53          boolean continueReading = true;
54
55          EntityTransaction etx = em.getTransaction();
56          etx.begin();
57
58          while (continueReading) {
59              System.out.print("Enter product number: ");
60              Integer productNumber = Integer.parseInt(scanner.nextLine());
61              System.out.print("Enter quantity: ");
62              Integer quantity = Integer.parseInt(scanner.nextLine());
63
64              Product orderedProduct = em.find(Product.class, productNumber);
65              if (quantity <= orderedProduct.getUnitsOnStock()) {
66                  newInvoice.addProduct(orderedProduct, quantity);
67              } else {
68                  System.out.println("Given number is bigger than units on stock number!");
69              }
70
71              System.out.println("End order? (Y/N)");
72              String response = scanner.nextLine().toLowerCase();
73
74              if (response.equals("y")) {
75                  continueReading = false;
76              }
77          }
78
79          em.persist(newInvoice);
80          loggedUser.addInvoice(newInvoice);
81
82          etx.commit();
83      }
84  }
```

Przypadek użycia dla istniejącego użytkownika przedstawia Rysunek 60, a dla rejestracji nowego Rysunek 61.

```
Hello to Java Hibernate App.
    L - log in as an existing user
    R - create new account
    Any other key - exit
Response: L
    Company name: Abcd
Authentication successfull
Hello customer
1 - show orders
2 - make new order
3 - exit
Enter option number: 1
Order number: 5
Quantity: 18    Zapałki
    Michałki
1 - show orders
2 - make new order
3 - exit
Enter option number: 2
ID      NAME        UNITS ON STOCK
2       Zapałki     9
3       Cebula      145
4       Michałki        1219
Enter product number: 3
Enter quantity: 5
End order? (Y/N)
y
1 - show orders
2 - make new order
3 - exit
Enter option number: 3

Process finished with exit code 0
```

Rysunek 60. Przypadek użycia dla zarejestrowanego użytkownika

```
Hello to Java Hibernate App.
    L - log in as an existing user
    R - create new account
    Any other key - exit
Response: R
Enter company name: Nowa firma
Company street: Opolska 123
Company postal code: 30-899
Company city: Kraków
Company country: Polska
Hello customer
1 - show orders
2 - make new order
3 - exit
Enter option number: 3

Process finished with exit code 0
```

Rysunek 61. Przypadek użycia dla rejestracji nowego użytkownika