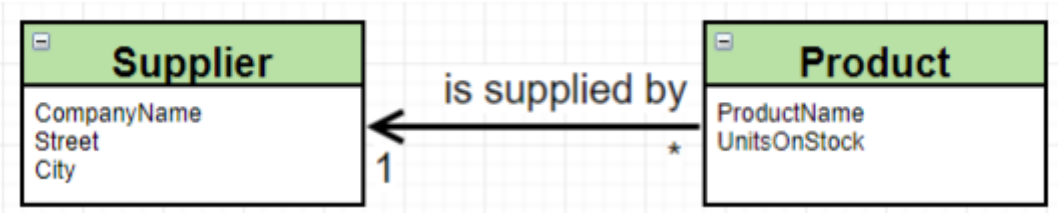
Laboratorium 3: Hibernate/JPA

Autor: Krzysztof Solecki

1. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej:



Klasa Product:

```
package org.example;
import javax.persistence.*;

2 usages
@Entity
public class Product {
    @Id
    @GeneratedValue(
    strategy = GenerationType.AUTO)
    private int productID;
    1 usage
    private String ProductName;
    1 usage
    private int UnitsOnStock;

    no usages
    @ManyToOne
    private Supplier supplier;

    public Product(){}

    1 usage
    public Product(String ProductName, int UnitsOnStock){
        this.ProductName = ProductName;
        this.UnitsOnStock = UnitsOnStock;
    }
    no usages
    public int getProductID(){ return productID;}
}
```

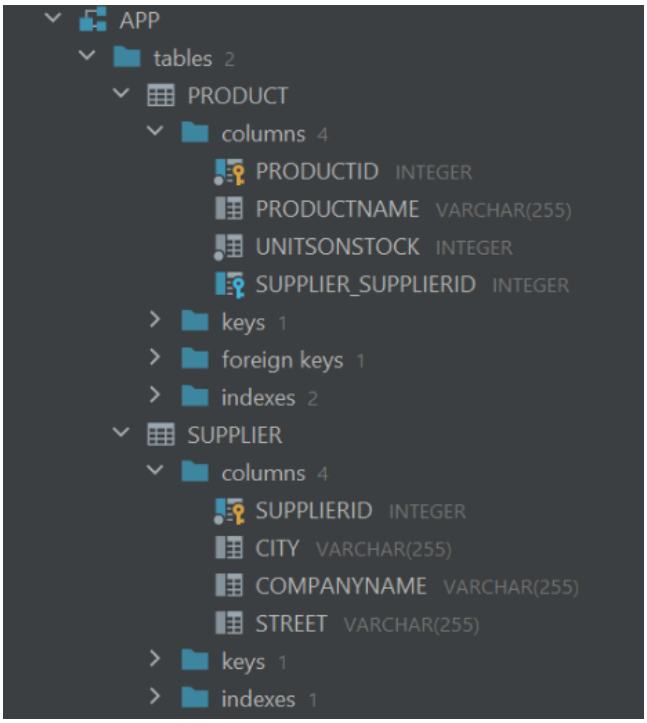
Klasa Supplier:

```java
package org.example;

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

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

    public Supplier(){}

    no usages
    public Supplier(String companyName, String street, String city){
        this.CompanyName = companyName;
        this.Street = street;
        this.City = city;
    }
}
```

Plik konfiguracyjny:

```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/KSoleckiJPA;create=true</property>
    <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <mapping class="org.example.Product" />
    <mapping class="org.example.Supplier" />
    <!-- <property name="connection.username"/> -->
    <!-- <property name="connection.password"/> -->

    <!-- DB schema will be updated if needed -->
    <!-- <property name="hibernate.hbm2ddl.auto">update</property> -->
  </session-factory>
</hibernate-configuration>
```

Schemat bazy:



a) Stwórz nowego dostawcę:

```
    public static void main(final String[] args) throws Exception {
        final Session session = getSession();
        Supplier supplier = new Supplier( companyName: "FirstCompany",  street: "Krakowska 64",  city: "Krakow");
        try {
            Transaction tx = session.beginTransaction();
            session.save(supplier);
            tx.commit();
        } finally {
            session.close();
        }
    }
}
```

1 row Tx: Auto DDL CSV

WHERE ORDER BY

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	2	Krakow	FirstCompany	Krakowska 64

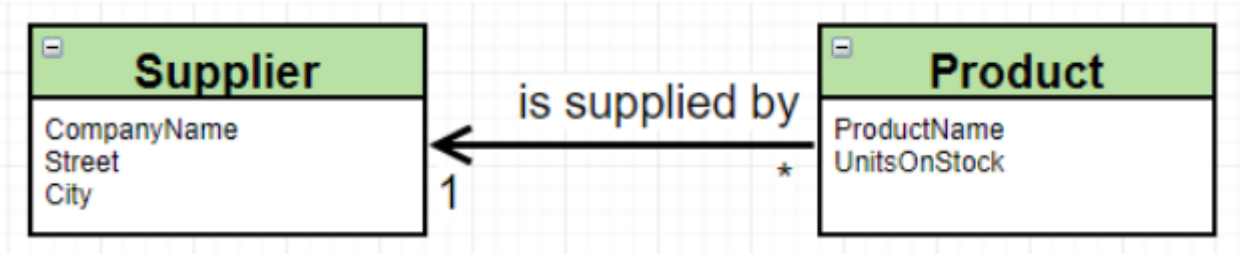
b) Znajdź poprzednio wprowadzony produkt i ustaw jego dostawcę na właśnie dodanego:

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

    try {
        Transaction tx = session.beginTransaction();
        Product foundProduct = session.get(Product.class,  serializable: 1);
        Supplier supplier = session.get(Supplier.class,  serializable: 2);
        foundProduct.setSupplier(supplier);
        tx.commit();
    } finally {
        session.close();
    }
}
```

WHERE ORDER BY

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	1	Krzeslo	111	2

2. Odwróć relację zgodnie z poniższym schematem:



a) Zamodeluj powyższe w dwóch wariantach “z” i “bez” tabeli łącznikowej
b) Stwórz kilka produktów
c) Dodaj je do produktów dostarczanych przez nowo stworzonego dostawcę

Z tabelą łącznikową:

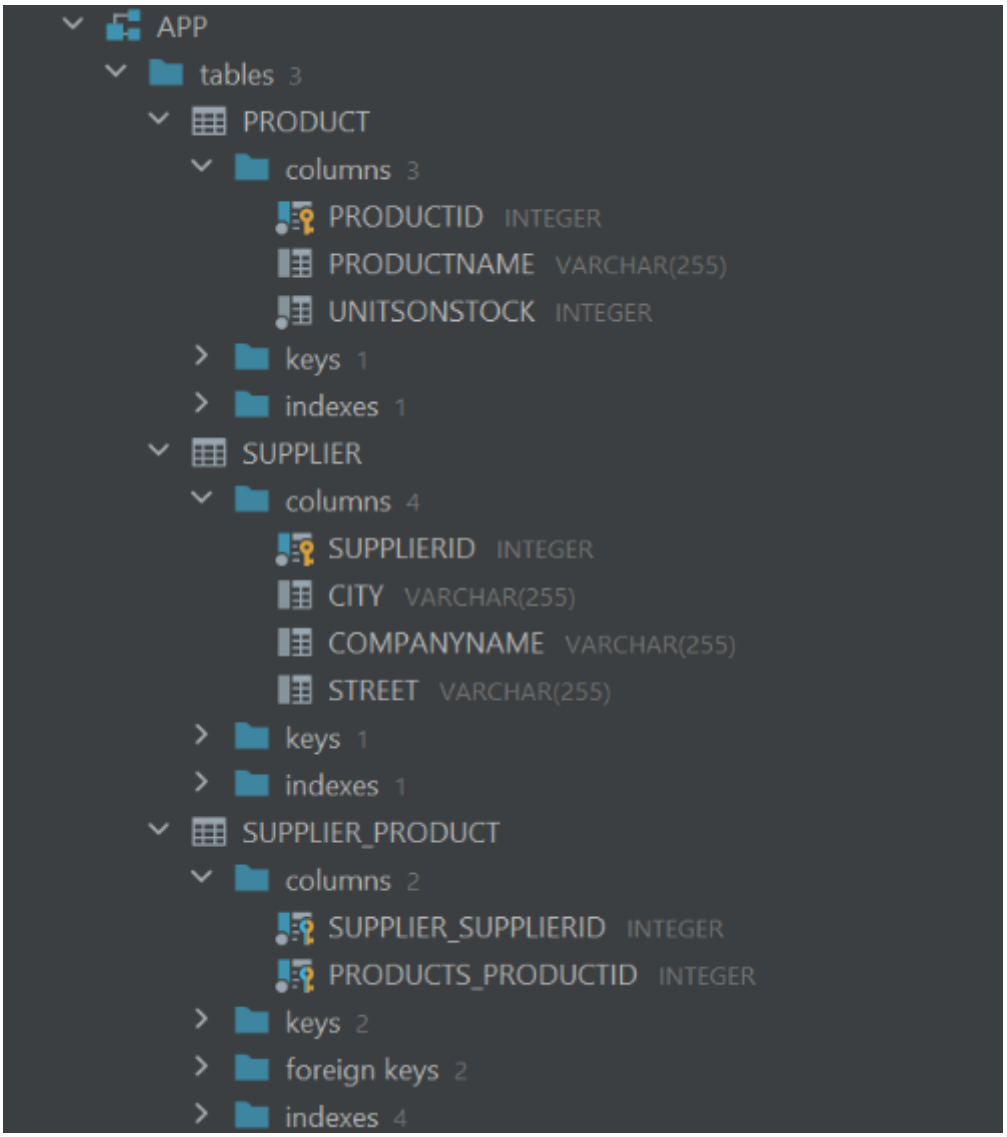
Klasa Product:

```
1   package org.example;
2   import javax.persistence.*;
3
    4 usages
4   @Entity
5   public class Product {
6       @Id
7       @GeneratedValue(
8       strategy = GenerationType.AUTO)
9       private int productID;
        1 usage
10      private String ProductName;
        1 usage
11      private int UnitsOnStock;
12
13      public Product(){}
14
        no usages
15      public Product(String ProductName, int UnitsOnStock){
16          this.ProductName = ProductName;
17          this.UnitsOnStock = UnitsOnStock;
18      }
        no usages
19      public int getProductID(){ return productID;}
20  }
21
```

Klasa Supplier:

```
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    1 usage
    private String CompanyName;
    1 usage
    private String Street;
    1 usage
    private String City;

    1 usage
    @OneToMany
    private Set<Product> products;

    public Supplier(){}

    no usages
    public Supplier(String companyName, String street, String city){
        this.CompanyName = companyName;
        this.Street = street;
        this.City = city;
        this.products = new HashSet<>();
    }
}
```

Struktura bazy:



Wprowadzenie danych:

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

    try {
        Transaction tx = session.beginTransaction();
        Product product1 = new Product( ProductName: "Dlugopis",  UnitsOnStock: 5);
        Product product2 = new Product( ProductName: "Linijka",  UnitsOnStock: 10);
        Product product3 = new Product( ProductName: "Kalkulator",  UnitsOnStock: 3);
        session.save(product1);
        session.save(product2);
        session.save(product3);
        Supplier supplier = new Supplier( companyName: "FirstCompany",  street: "Krakowska 15",  city: "Krakow");
        session.save(supplier);
        supplier.addProductToSet(product1);
        supplier.addProductToSet(product2);
        supplier.addProductToSet(product3);
        tx.commit();
    } finally {
        session.close();
    }
}
}
```

WHERE ORDER BY

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Dlugopis	5
2	2	Linijka	10
3	3	Kalkulator	3

WHERE ORDER BY

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Krakow	FirstCompany	Krakowsa 15

WHERE ORDER BY

	SUPPLIER_SUPPLIERID	PRODUCTS_PRODUCTID
1	4	1
2	4	2
3	4	3

Bez tabeli łącznikowej:

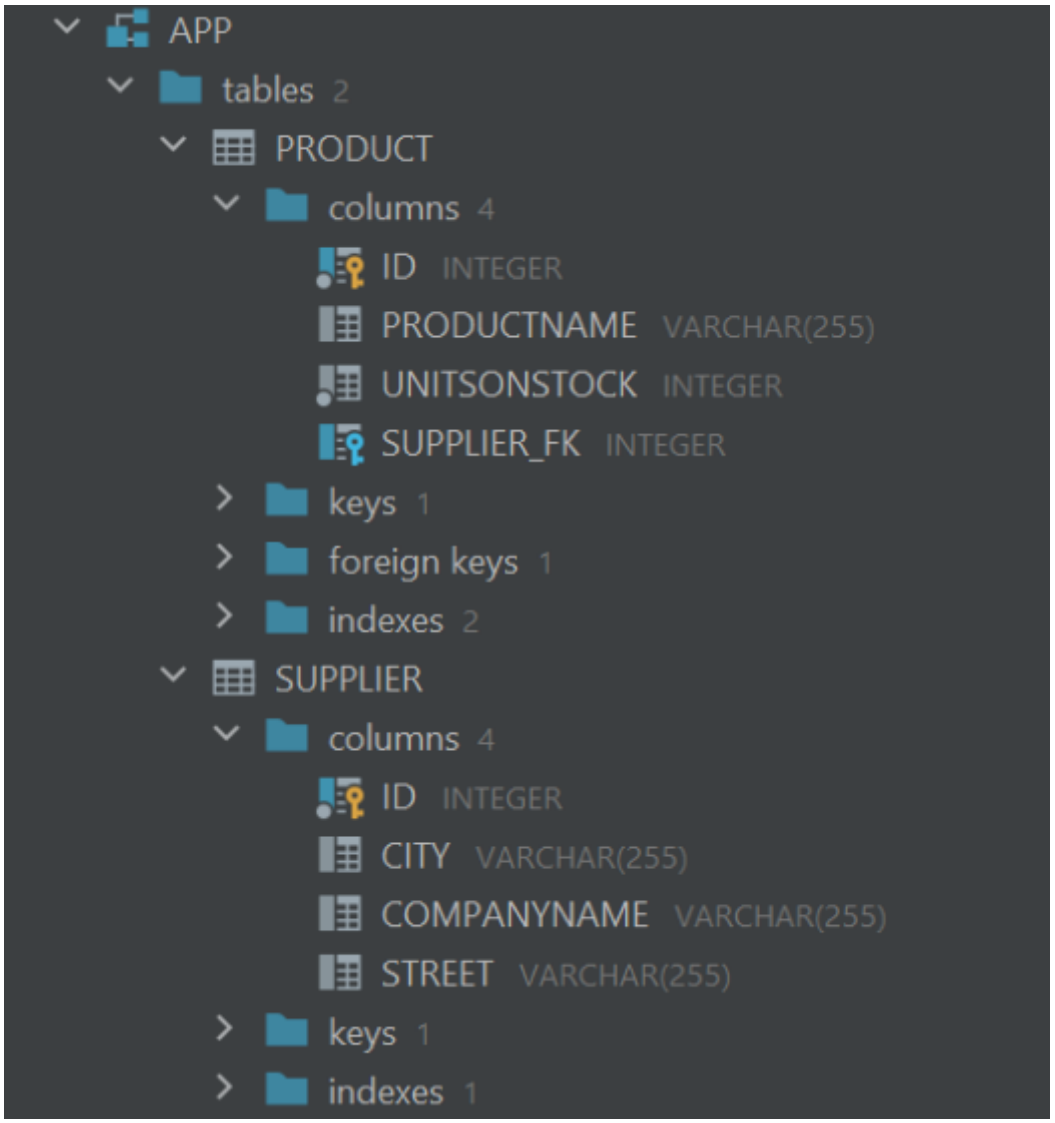
Klasa Product:

```
10 usages
@Entity
public class Product {
    @Id
    @GeneratedValue(
    strategy = GenerationType.AUTO)
    private int id;
    1 usage
    private String ProductName;
    1 usage
    private int UnitsOnStock;
    no usages
    @Column(name="SUPPLIER_FK")
    private Integer supplier_fk;

    public Product(){}

    3 usages
    public Product(String ProductName, int UnitsOnStock){
        this.ProductName = ProductName;
        this.UnitsOnStock = UnitsOnStock;
    }
```

Klasa Supplier:

```
4 usages
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    1 usage
    private String CompanyName;
    1 usage
    private String Street;
    1 usage
    private String City;

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

    public Supplier(){}

    1 usage
    public Supplier(String companyName, String street, String city){
        this.CompanyName = companyName;
        this.Street = street;
        this.City = city;
        this.products = new HashSet<>();
    }
```

Struktura bazy:



Wprowadzanie danych:

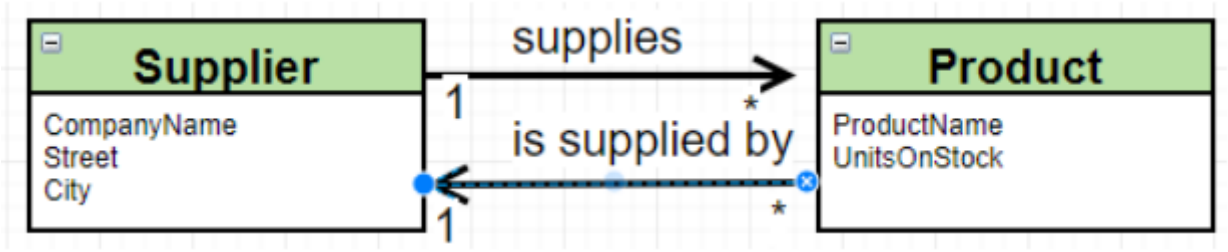
```java
public static void main(final String[] args) throws Exception {
    final Session session = getSession();

    try {
        Transaction tx = session.beginTransaction();
        Product product1 = new Product( ProductName: "Dlugopis",  UnitsOnStock: 5);
        Product product2 = new Product( ProductName: "Linijka",  UnitsOnStock: 10);
        Product product3 = new Product( ProductName: "Kalkulator",  UnitsOnStock: 3);
        session.save(product1);
        session.save(product2);
        session.save(product3);
        Supplier supplier = new Supplier( companyName: "FirstCompany",  street: "Krakowska 15",  city: "Krakow");
        session.save(supplier);
        supplier.addProductToSet(product1);
        supplier.addProductToSet(product2);
        supplier.addProductToSet(product3);
        tx.commit();
    } finally {
        session.close();
    }
}
}
```

WHERE ORDER BY

	ID	CITY	COMPANYNAME	STREET
1	4	Krakow	FirstCompany	Krakowsa 15

	ID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_FK
1	1	Dlugopis	5	4
2	2	Linijka	10	4
3	3	Kalkulator	3	4

3. Zamodeluj relację dwustronną jak poniżej:



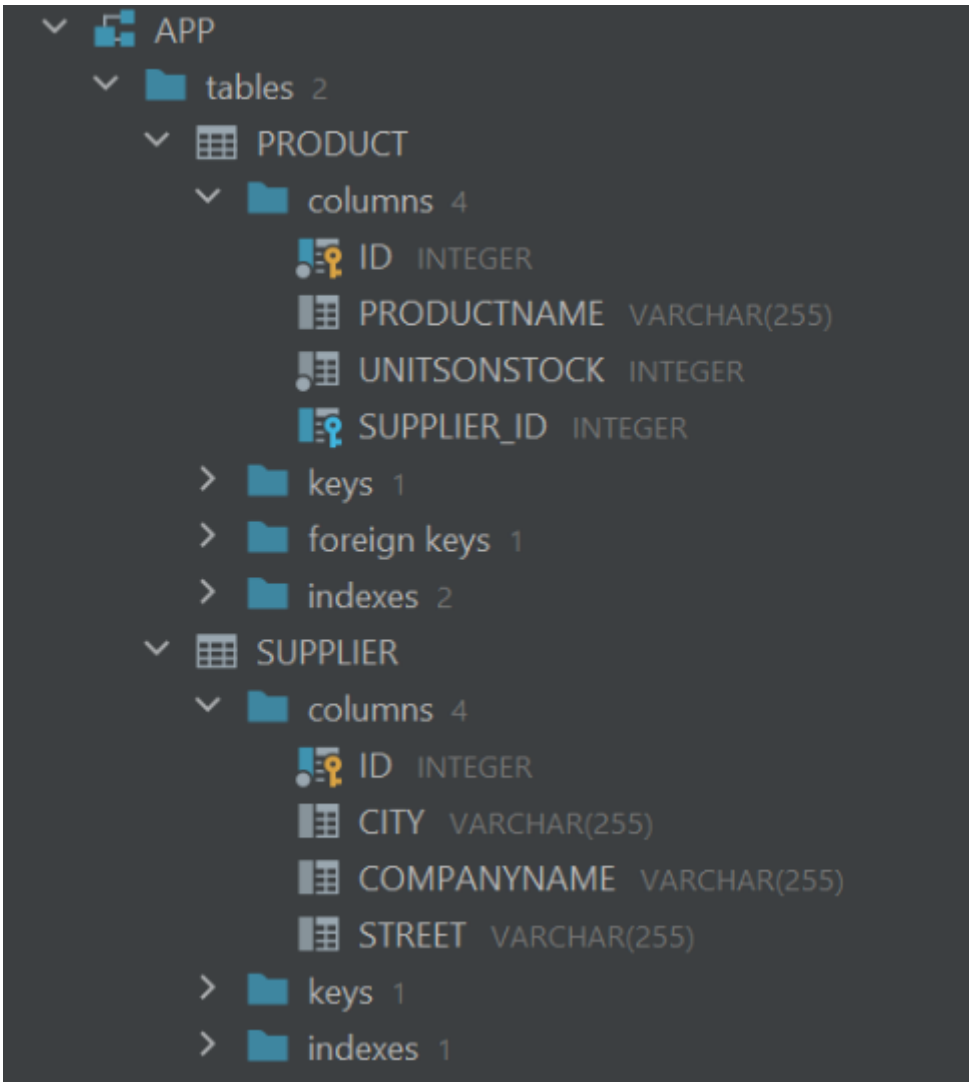
Klasa Product:

```
10 usages
@Entity
public class Product {
    @Id
    @GeneratedValue(
    strategy = GenerationType.AUTO)
    private int id;
    1 usage
    private String ProductName;
    1 usage
    private int UnitsOnStock;
    no usages
    @ManyToOne
    private Supplier supplier;

    public Product(){}

    3 usages
    public Product(String ProductName, int UnitsOnStock){
        this.ProductName = ProductName;
        this.UnitsOnStock = UnitsOnStock;
    }
```

Klasa Supplier:

```
5 usages
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    1 usage
    private String CompanyName;
    1 usage
    private String Street;
    1 usage
    private String City;

    2 usages
    @OneToMany(mappedBy = "supplier")
    private Set<Product> products = new HashSet<>();

    public Supplier(){}

    1 usage
    public Supplier(String companyName, String street, String city){
        this.CompanyName = companyName;
        this.Street = street;
        this.City = city;
        this.products = new HashSet<>();
    }
```

Struktura bazy:



a) Stwórz kilka produktów
b) Dodaj je do produktów dostarczanych przez nowo stworzonego dostawcę (dwustronność relacji)

Wprowadzenie danych:

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

    try {
        Transaction tx = session.beginTransaction();
        Product product1 = new Product( ProductName: "Dlugopis",  UnitsOnStock: 5);
        Product product2 = new Product( ProductName: "Linijka",  UnitsOnStock: 10);
        Product product3 = new Product( ProductName: "Pioro",  UnitsOnStock: 20);
        session.save(product1);
        session.save(product2);
        session.save(product3);
        Supplier supplier = new Supplier( companyName: "FirstCompany",  street: "Krakowska 15",  city: "Krakow");
        session.save(supplier);
        product1.setSupplier(supplier);
        product2.setSupplier(supplier);
        product3.setSupplier(supplier);
        tx.commit();
    } finally {
        session.close();
    }
}
```

WHERE ORDER BY

	ID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_ID
1	1	Dlugopis	10	4
2	2	Linijka	5	4
3	3	Pioro	20	4

WHERE ORDER BY

	ID	CITY	COMPANYNAME	STREET
1	4	Krakow	FirstCompany	Krakowska 15

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

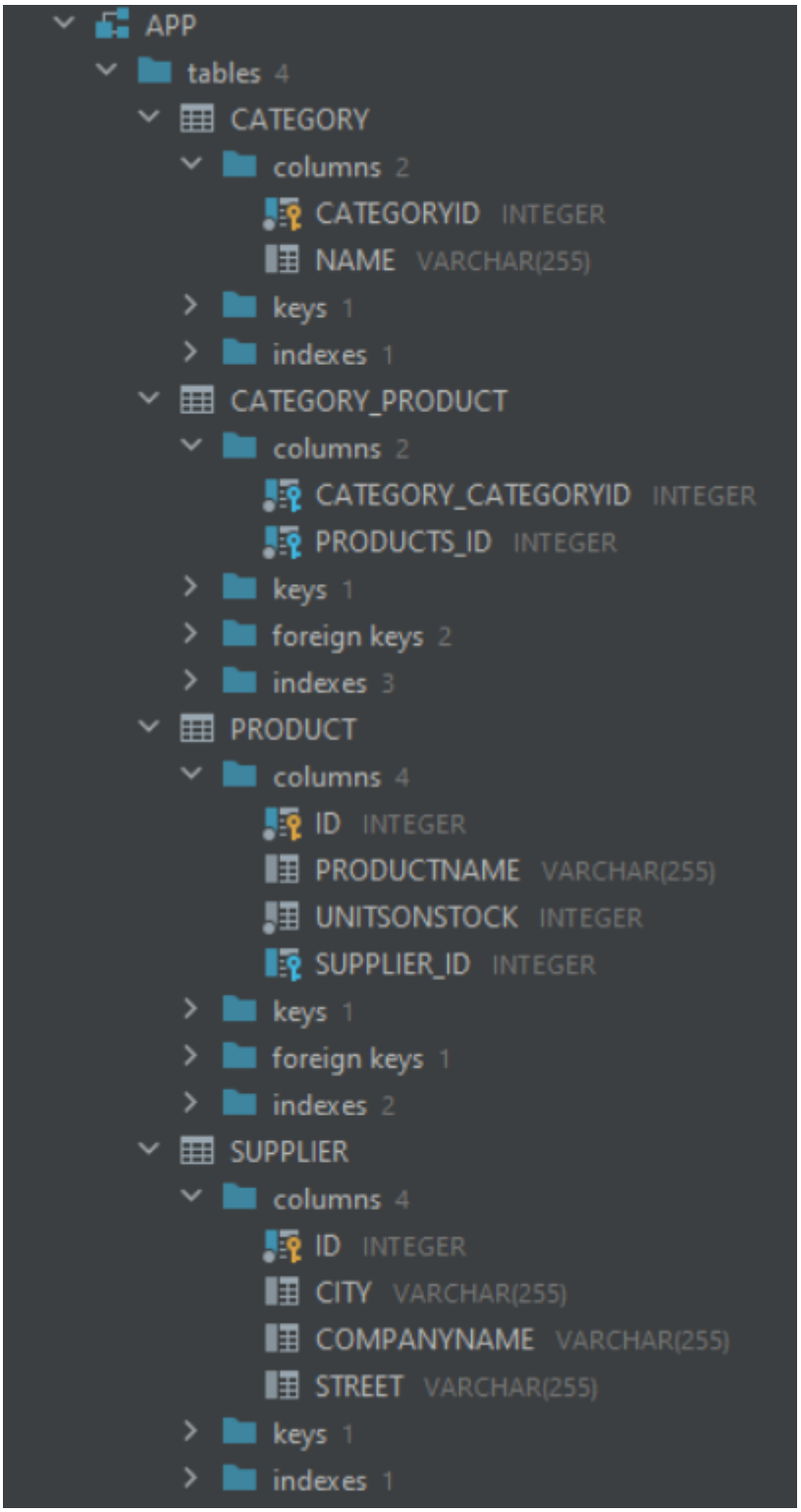
Klasa Category:

```
no usages
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CategoryID;
    1 usage
    private String Name;
    1 usage
    @OneToMany
    private List<Product> Products;

    no usages
    public Category(String name){
        this.Name = name;
        this.Products = new ArrayList<>();
    }

}
```

Plik konfiguracyjny:

```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/KSoleckiJPA;create=true</property>
    <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <mapping class="org.example.Product"/>
    <mapping class="org.example.Supplier"/>
    <mapping class="org.example.Category"/>
    <!-- <property name="connection.username"/> -->
    <!-- <property name="connection.password"/> -->

    <!-- DB schema will be updated if needed -->
    <!-- <property name="hibernate.hbm2ddl.auto">update</property> -->
  </session-factory>
</hibernate-configuration>
```

Struktura bazy:



Zmodyfikuj produkty dodając wskazanie na kategorie do której należy

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Category category = new Category( name: "schoolTools");
    try {
        Transaction tx = session.beginTransaction();
        session.save(category);
        Product product1 = session.get(Product.class,  serializable: 1);
        Product product2 = session.get(Product.class,  serializable: 2);
        Product product3 = session.get(Product.class,  serializable: 3);

        category.addProductToList(product1);
        category.addProductToList(product2);
        category.addProductToList(product3);

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

WHERE ORDER BY

	CATEGORYID	NAME
1	7	schoolTools

WHERE ORDER BY

	CATEGORY_CATEGORYID	PRODUCTS_ID
1	7	1
2	7	2
3	7	3

b. Stworz kilka produktow i kilka kategorii

c. Dodaj kilka produktów do wybranej kategorii

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Category category1 = new Category( name: "Phones");
    Product product1 = new Product( ProductName: "IPhone",  UnitsOnStock: 10);
    Product product2 = new Product( ProductName: "Samsung",  UnitsOnStock: 10);
    Category category2 = new Category( name: "Cars");
    Product product3 = new Product( ProductName: "BMW",  UnitsOnStock: 2);
    Product product4 = new Product( ProductName: "OPEL",  UnitsOnStock: 5);
    Category category3 = new Category( name: "TVs");
    Product product5 = new Product( ProductName: "LG",  UnitsOnStock: 1);
    Product product6 = new Product( ProductName: "DELL",  UnitsOnStock: 2);
    try {
        Transaction tx = session.beginTransaction();
        session.save(category1);
        session.save(product1);
        session.save(product2);
        category1.addProductToList(product1);
        category1.addProductToList(product2);

        session.save(category2);
        session.save(product3);
        session.save(product4);
        category1.addProductToList(product3);
        category1.addProductToList(product4);

        session.save(category3);
        session.save(product5);
        session.save(product6);
        category1.addProductToList(product5);
        category1.addProductToList(product5);

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

WHERE ORDER BY

	ID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_ID
1	1	Dlugopis	10	4
2	2	Linijka	5	4
3	3	Pioro	20	4
4	9	IPhone	10	<null>
5	10	Samsung	10	<null>
6	12	BMW	2	<null>
7	13	OPEL	5	<null>
8	15	LG	1	<null>
9	16	DELL	2	<null>

WHERE ORDER BY

	CATEGORYID	NAME
1	7	schoolTools
2	8	Phones
3	11	Cars
4	14	TVs

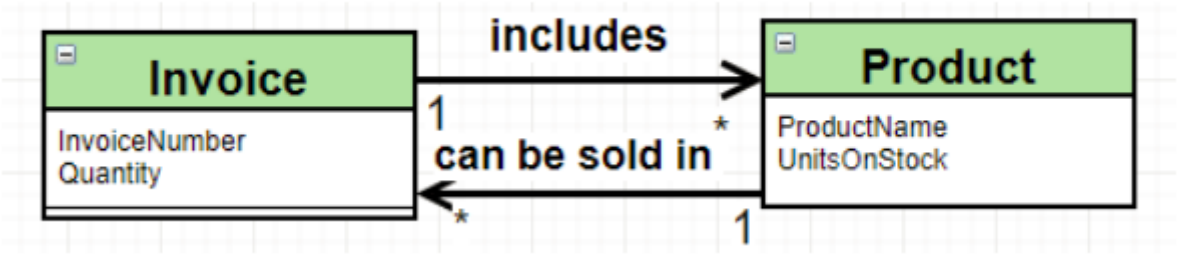
WHERE ORDER BY

	CATEGORY_CATEGORYID	PRODUCTS_ID
1	7	1
2	7	2
3	7	3
4	8	9
5	8	10
6	11	12
7	11	13
8	14	15
9	14	16

Wydobądź produkty z wybranej kategorii oraz kategorię do której należy wybrany produkt:

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

    Query query = session.createQuery( s: "From Category where CategoryID = 8");
    Category category = (Category) query.getResultList().get(0);
    for(Product product: category.getProducts()){
        System.out.println(product.getProductName());
    }

    tx.commit();
```

5. Zamodeluj relację wiele-do-wielu jak poniżej:



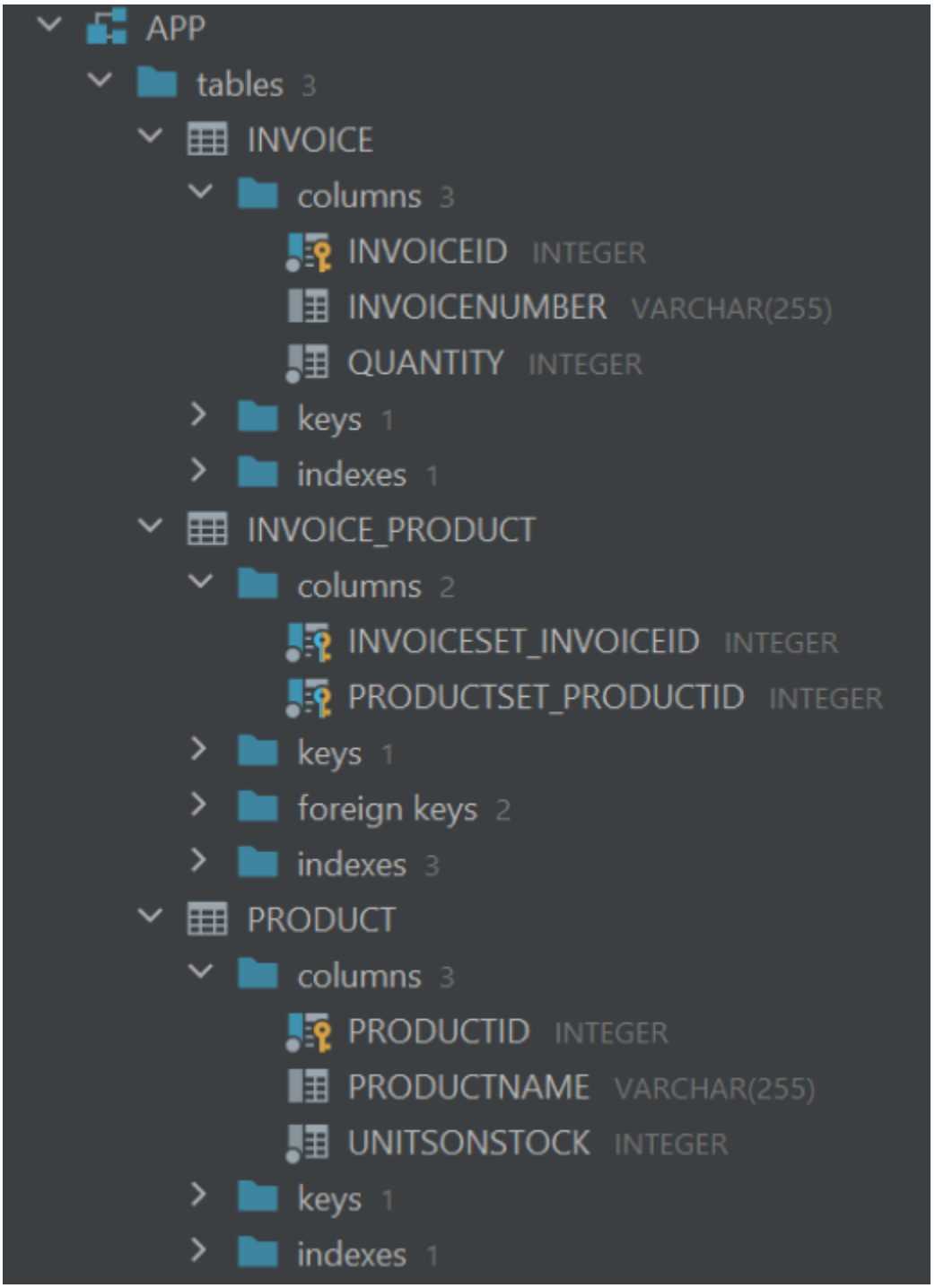
Klasa Product:

```
9 usages
@Entity
public class Product {
    @Id
    @GeneratedValue(
    strategy = GenerationType.AUTO)
    private int ProductID;
    1 usage
    private String ProductName;
    1 usage
    private int UnitsOnStock;
    no usages
    @ManyToMany(mappedBy = "productSet")
    private Set<Invoice> invoiceSet;

    public Product(){}

    no usages
    public Product(String ProductName, int UnitsOnStock){
        this.ProductName = ProductName;
        this.UnitsOnStock = UnitsOnStock;
    }
```

Klasa Invoice:

```
1 usage
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int InvoiceID;
    1 usage
    private String InvoiceNumber;
    1 usage
    private int Quantity;
    no usages
    @ManyToMany
    private Set<Product> productSet;

    no usages
    public Invoice(String invoiceNumber, int quantity){
        this.InvoiceNumber = invoiceNumber;
        this.Quantity = quantity;
    }

    no usages
    public Invoice(){};
}
```

Struktura bazy:



Stwórz kilka produktów I “sprzedaj” je na kilku transakcjach:

```
    final Session session = getSession();
    Product product1 = new Product( productName: "Dlugopis", unitsOnStock: 5);
    Product product2 = new Product( productName: "Olowek", unitsOnStock: 5);
    Product product3 = new Product( productName: "Pioro", unitsOnStock: 5);
    Product product4 = new Product( productName: "Linijka", unitsOnStock: 5);

    Invoice invoice1 = new Invoice( invoiceNumber: "123123123", quantity: 5);
    Invoice invoice2 = new Invoice( invoiceNumber: "321321321", quantity: 2);
    try {
        Transaction tx = session.beginTransaction();
        session.save(product1);
        session.save(product2);
        session.save(product3);
        session.save(product4);

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

        invoice1.sellProduct(product1, quantity: 3);
        invoice1.sellProduct(product2, quantity: 2);

        invoice2.sellProduct(product3, quantity: 1);
        invoice2.sellProduct(product4, quantity: 1);

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

WHERE ORDER BY

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Dlugopis	2
2	2	Olowek	3
3	3	Pioro	4
4	4	Linijka	4

WHERE ORDER BY

	INVOICEID	INVOICENUMBER	QUANTITY
1	5	123123123	5
2	6	321321321	2

WHERE ORDER BY

	INVOICESET_INVOICEID	PRODUCTSET_PRODUCTID
1	5	1
2	5	2
3	6	3
4	6	4

Pokaż produkty sprzedane w ramach wybranej faktury/transakcji:

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

        Query query = session.createQuery( s: "from Invoice where InvoiceID = 5");
        Invoice invoice = (Invoice) query.getResultList().get(0);

        for(Product product : invoice.getProductSet()){
            System.out.println(product.getProductName());
        }

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

Pokaż faktury w ramach których był sprzedany wybrany produkt:

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

        Query query = session.createQuery( s: "from Product where ProductID = 1");
        Product product = (Product) query.getResultList().get(0);

        for(Invoice invoice: product.getInvoiceSet()){
            System.out.println(invoice.getInvoiceNumber());
        }

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

6. JPA

a. Stwórz nowego main’a w którym zrobisz to samo co w poprzednim ale z wykorzystaniem JPA:

a. Stwórz kilka produktów I “sprzedaj” je na kilku transakcjach

```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();
         Product product1 = new Product( productName: "Dlugopis",  unitsOnStock: 5);
         Product product2 = new Product( productName: "Olowek",  unitsOnStock: 5);
         Product product3 = new Product( productName: "Pioro",  unitsOnStock: 5);
         Product product4 = new Product( productName: "Linijka",  unitsOnStock: 5);

         Invoice invoice1 = new Invoice( invoiceNumber: "123123123",  quantity: 5);
         Invoice invoice2 = new Invoice( invoiceNumber: "321321312",  quantity: 2);

         //save products
         em.persist(product1);
         em.persist(product2);
         em.persist(product3);
         em.persist(product4);

         //save invoices
         em.persist(invoice1);
         em.persist(invoice2);

         //sellproducts
         invoice1.sellProduct(product1, quantity: 3);
         invoice1.sellProduct(product2, quantity: 2);

         invoice2.sellProduct(product3, quantity: 1);
         invoice2.sellProduct(product4, quantity: 1);

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

;

b. Pokaż produkty sprzedane w ramach wybranej faktury/transakcji:

```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceU
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();
         Query query = em.createQuery( s: "from Invoice where InvoiceID = 5");
         Invoice invoice = (Invoice) query.getResultList().get(0);

         for(Product product: invoice.getProductSet()){
             System.out.println(product.getProductName());
         }
        etx.commit();
        em.close();
    }
```

Pokaż faktury w ramach których był sprzedany wybrany produkt:

```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUn
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();
         Query query = em.createQuery( s: "from Product where ProductID = 1");
         Product product = (Product) query.getResultList().get(0);

         for(Invoice invoice: product.getInvoiceSet()){
             System.out.println(invoice.getInvoiceNumber());
         }
        etx.commit();
        em.close();
    }
```

7. Kaskady:

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

Nowy produkt przy nowej fakturze:

Klasa Invoice:

```
@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int InvoiceID;
    private String InvoiceNumber;
    private int Quantity;
    @ManyToMany(cascade = CascadeType.PERSIST)
    private Set<Product> productSet;

    public Invoice(String invoiceNumber, int quantity){
        this.InvoiceNumber = invoiceNumber;
        this.Quantity = quantity;
        this.productSet = new HashSet<>();
    }

    public Invoice() {}
```

```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitNa
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        Invoice invoice = new Invoice( invoiceNumber: "000000000", quantity: 1);
        Product product = new Product( productName: "NowyProdukt", unitsOnStock: 10);
        etx.begin();
         em.persist(invoice);
         invoice.sellProduct(product, quantity: 1);
        etx.commit();
        em.close();
    }
}
```

	INVOICEID	INVOICENUMBER	QUANTITY
1	5	123123123	5
2	6	321321312	2
3	7	000000000	1

WHERE ORDER BY

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Dlugopis	2
2	2	Olowek	3
3	3	Pioro	4
4	4	Linijka	4
5	8	NowyProdukt	9

WHERE ORDER BY

	INVOICESET_INVOICEID	PRODUCTSET_PRODUCTID
1	5	1
2	5	2
3	6	3
4	6	4
5	7	8

Nowa faktura przy nowym produkcie:

```java
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsOnStock;
    @ManyToMany(mappedBy = "productSet",cascade = CascadeType.PERSIST)
    private Set<Invoice> invoiceSet;

    public Product() {}

    public Product(String productName, int unitsOnStock) {
        this.ProductName = productName;
        this.UnitsOnStock = unitsOnStock;
        this.invoiceSet = new HashSet<>();
    }
```

```java
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        Invoice invoice = new Invoice( invoiceNumber: "111111111", quantity: 1);
        Product product = new Product( productName: "NowiutkiProdukcik", unitsOnStock: 10);
        etx.begin();
         product.addInvoiceToSet(invoice);
         invoice.sellProduct(product, quantity: 1);
         em.persist(product);
        etx.commit();
        em.close();
    }
}
```

4 rows | Tx: Auto | DDL

WHERE | ORDER BY

	INVOICEID	INVOICENUMBER	QUANTITY
1	5	123123123	5
2	6	321321312	2
3	7	000000000	1
4	12	111111111	1

WHERE | ORDER BY

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Dlugopis	2
2	2	Olowek	3
3	3	Pioro	4
4	4	Linijka	4
5	8	NowyProdukt	9
6	11	NowiutkiProdukcik	9

WHERE | ORDER BY

	INVOICESET_INVOICEID	PRODUCTSET_PRODUCTID
1	5	1
2	5	2
3	6	3
4	6	4
5	7	8
6	12	11

8. Embedded class

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

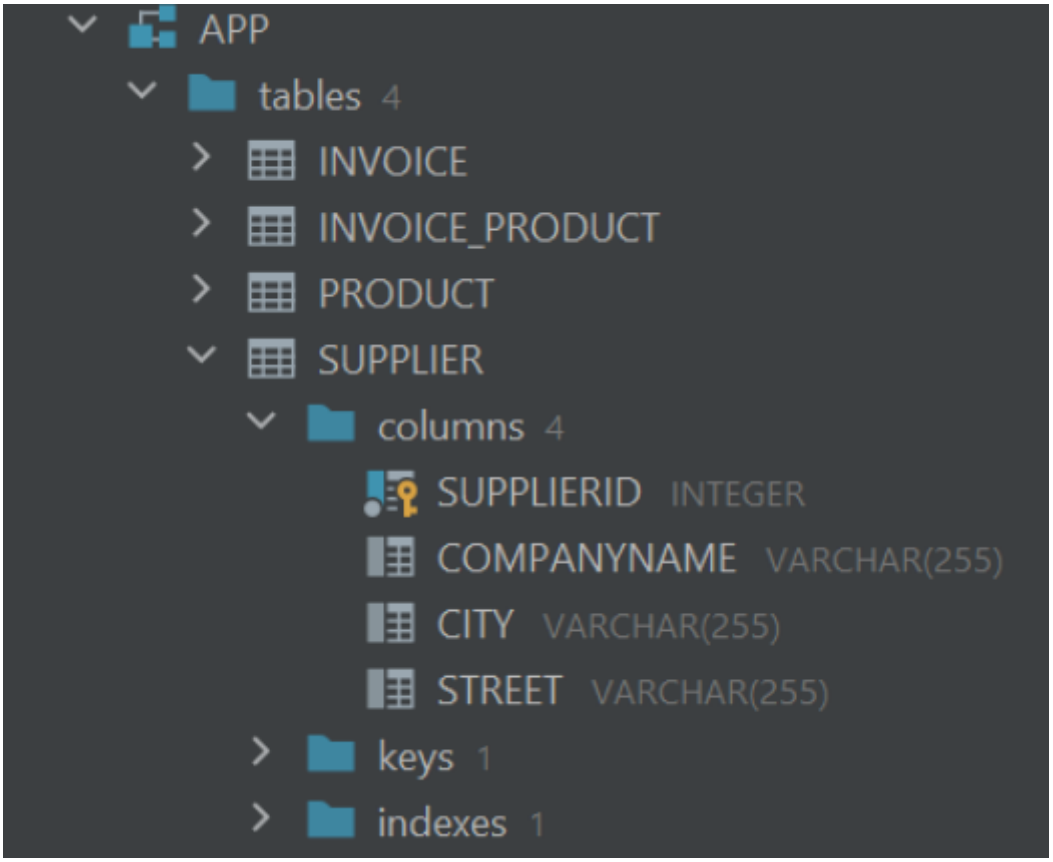
Klasa Adres:

```java
import javax.persistence.Embeddable;

@Embeddable
public class Address {
    private String Street;
    private String City;

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

    public Address() {

    }
```

Klasa Supplier:

```java
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private Address address;
    @OneToMany(mappedBy = "supplier")
    public Set<Product> productSet = new HashSet<>();

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

Struktura bazy:



Dodanie nowego dostawcy:

```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        Supplier supplier = new Supplier( companyName: "FirstCompany", street: "Krakowska 15", city: "Krakow");
        etx.begin();
         em.persist(supplier);
        etx.commit();
        em.close();
    }
}
```

WHERE ORDER BY

	SUPPLIERID	COMPANYNAME	CITY	STREET
1	1	FirstCompany	Krakow	Krakowska 15

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

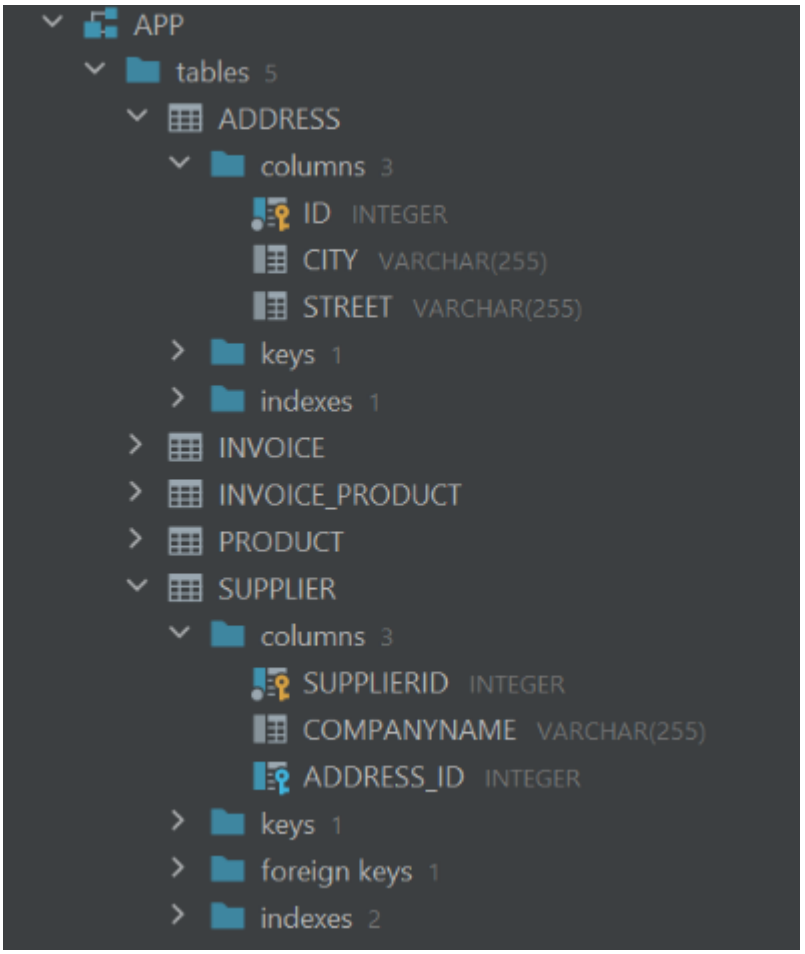
Klasa Adres:

```
import javax.persistence.*;

@Entity
public class Address {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String Street;
    private String City;

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

    public Address() {

    }
```

Klasa Supplier:

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

@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    @OneToOne(cascade = CascadeType.PERSIST)
    private Address address;
    @OneToMany(mappedBy = "supplier")
    public Set<Product> productSet = new HashSet<>();

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

Struktura bazy:



Dodanie dostawcy:

```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        Supplier supplier = new Supplier( companyName: "FirstCompany", street: "Krakowska 15", city: "Krakow");
        etx.begin();
         em.persist(supplier);
        etx.commit();
        em.close();
    }
}
```

WHERE ORDER BY

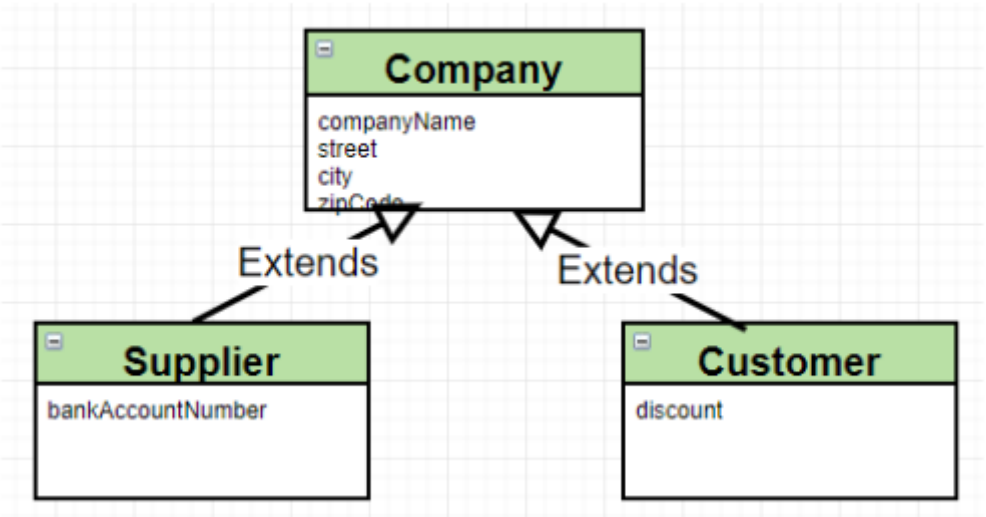
	ID	CITY	STREET
1	2	Krakow	Krakowska 15

WHERE ORDER BY

	SUPPLIERID	COMPANYNAME	ADDRESS_ID
1	1	FirstCompany	2

9. Dziedziczenie

a. Wprowadź do modelu następującą hierarchię:



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

Klasa Supplier:

```java
import javax.persistence.*;

@Entity
public class Supplier extends Company{

    private String BankAccountNumber;

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

    public String getBankAccountNumber() {
        return BankAccountNumber;
    }

    public void setBankAccountNumber(String bankAccountNumber) {
        BankAccountNumber = bankAccountNumber;
    }
}
```

Klasa Customer:

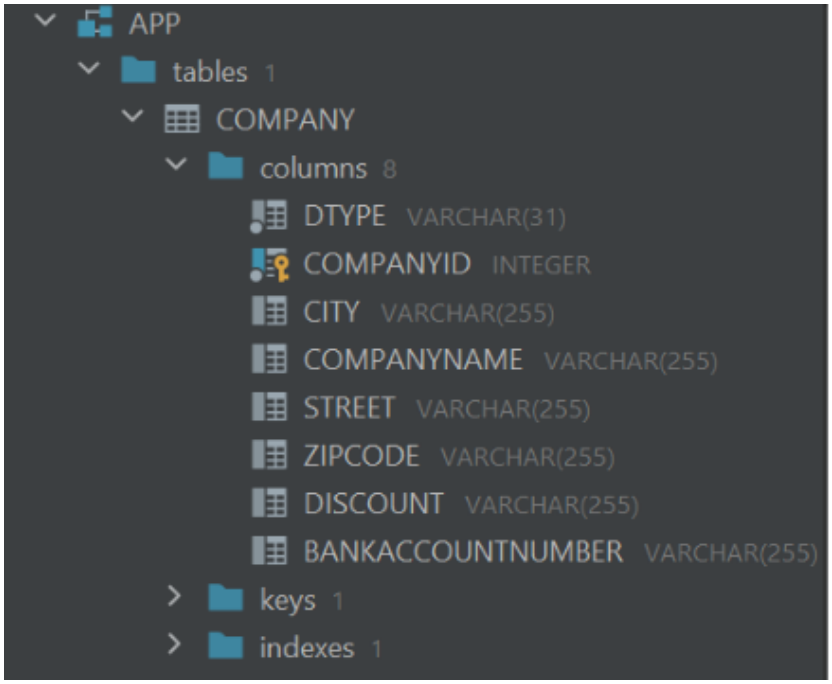
```java
public class Customer extends Company{

    private String Discount;

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

    public String getDiscount() {
        return Discount;
    }

    public void setDiscount(String discount) {
        Discount = discount;
    }
}
```

SINGLE_TABLE

Klasa Company:

```java
import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyID;
    private String CompanyName;
    private String Street;
    private String City;
    private String ZipCode;

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

    public Company() {}
```

Struktura bazy:



Dodanie firm:

```java
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        Customer customer = new Customer( companyName: "FirstCompany", street: "Krakowska 15",  city: "Krakow", zipCode: "38-200", discount: "15%");
        Supplier supplier = new Supplier( companyName: "SecondCompany", street: "Poznanska 12",  city: "Poznan", zipCode: "63-123", bankAccountNumber: "12341
        etx.begin();
         em.persist(customer);
         em.persist(supplier);
        etx.commit();
        em.close();
    }
}
```

WHERE ORDER BY

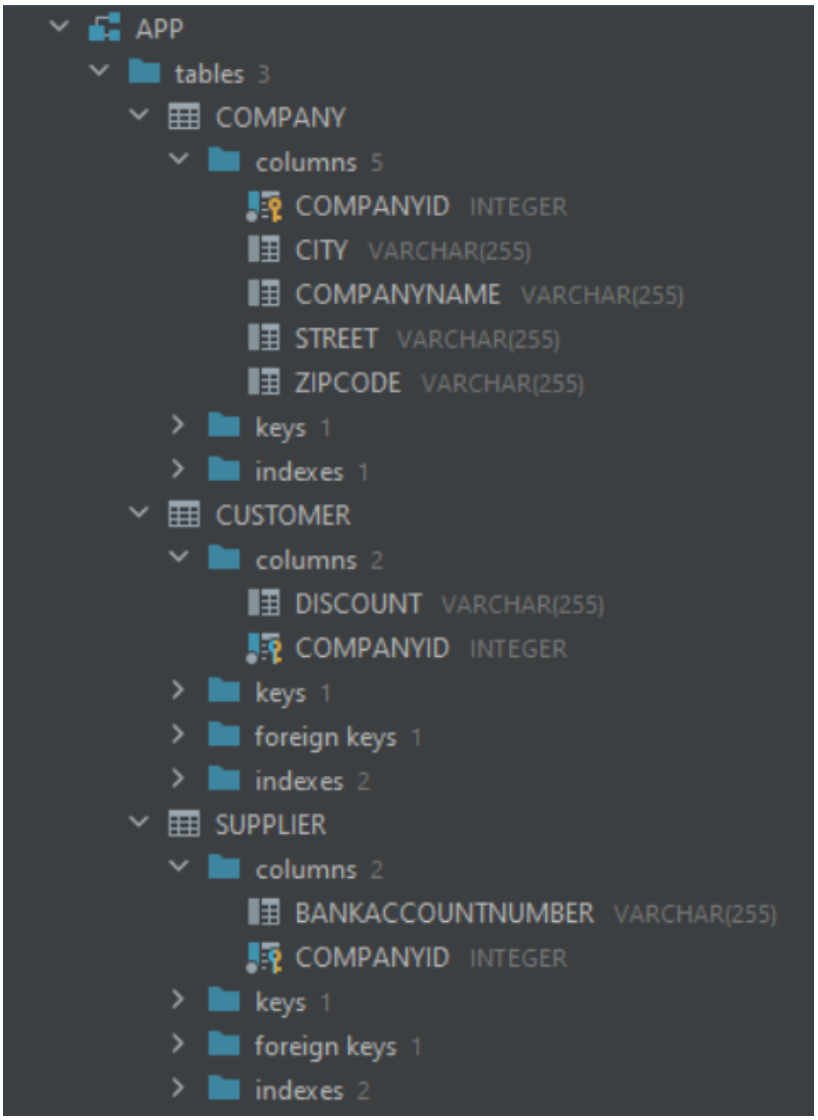
	DTYPE	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT	BANKACCOUNTNUMBER
1	Customer	1	Krakow	FirstCompany	Krakowska 15	38-200	15%	<null>
2	Supplier	2	Poznan	SecondCompany	Poznanska 12	63-123	<null>	1234123412341234

JOINED:

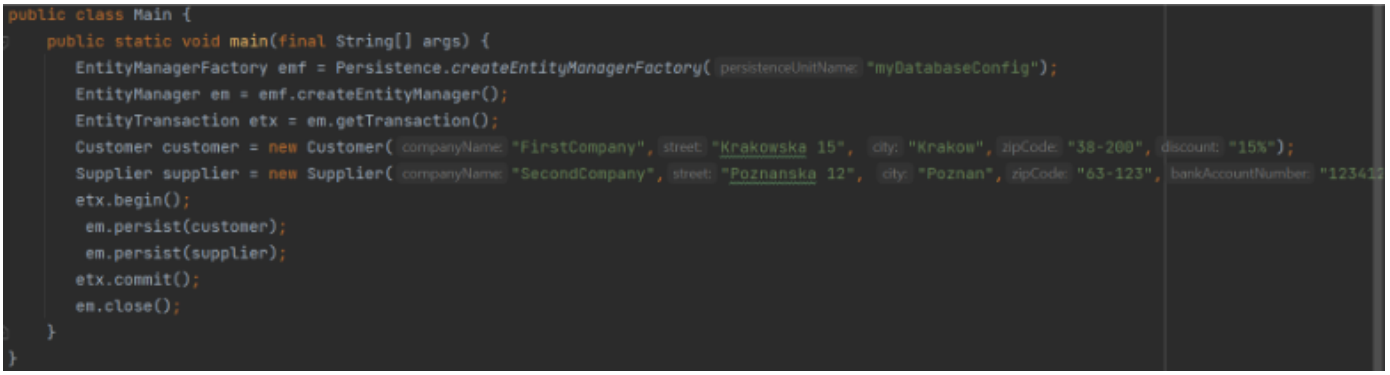
Klasa Company:

```java
import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyID;
    private String CompanyName;
    private String Street;
    private String City;
    private String ZipCode;

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

    public Company() {}
```

Struktura bazy:



Dodanie firm:



```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        Customer customer = new Customer( companyName: "FirstCompany", street: "Krakowska 15",  city: "Krakow", zipCode: "38-200", discount: "15%");
        Supplier supplier = new Supplier( companyName: "SecondCompany", street: "Poznanska 12",  city: "Poznan", zipCode: "63-123", bankAccountNumber: "12341
        etx.begin();
         em.persist(customer);
         em.persist(supplier);
        etx.commit();
        em.close();
    }
}
```

WHERE ORDER BY

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	Krakow	FirstCompany	Krakowska 15	38-200
2	2	Poznan	SecondCompany	Poznanska 12	63-123

WHERE

	DISCOUNT	COMPANYID
1	15%	1

WHERE ORDER BY

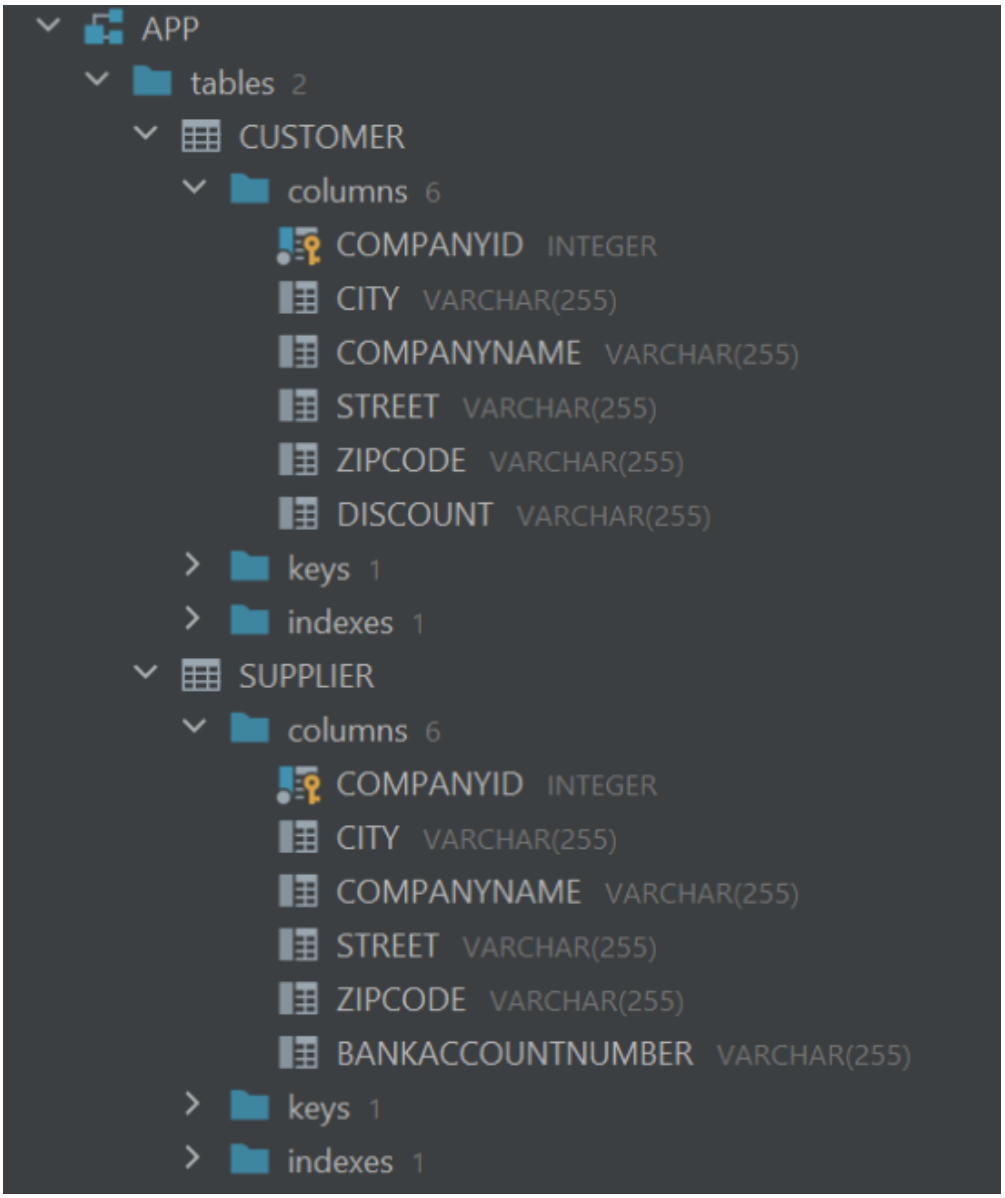
	BANKACCOUNTNUMBER	COMPANYID
1	1234123412341234	2

TABLE_PER_CLASS:

Klasa Company:

```
import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyID;
    private String CompanyName;
    private String Street;
    private String City;
    private String ZipCode;

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

    public Company() {}
```

Struktura bazy:



Dodanie firm:

```
public class Main {
    public static void main(final String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        Customer customer = new Customer( companyName: "FirstCompany", street: "Krakowska 15",  city: "Krakow", zipCode: "38-200", discount: "15%");
        Supplier supplier = new Supplier( companyName: "SecondCompany", street: "Poznanska 12",  city: "Poznan", zipCode: "63-123", bankAccountNumber: "12341
        etx.begin();
         em.persist(customer);
         em.persist(supplier);
        etx.commit();
        em.close();
    }
}
```

WHERE ORDER BY

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT
1	1	Krakow	FirstCompany	Krakowska 15	38-200	15%

WHERE ORDER BY

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	2	Poznan	SecondCompany	Poznanska 12	63-123	1234123412341234