

# Hibernate

Bartosz Szar

## 1.a - 1.i - Konfiguracja:

Pobrałem i rozpakowałem DBMS Derby oraz ustawiłem JAVA\_HOME na Javę 13 w celu wyeliminowania błędu pojawiającego się przy próbie uruchomienia serwera.

```
PS C:\Users\Lenovo\Desktop\db-derby-10.15.2.0-bin\bin> ./startNetworkServer
Thu Apr 30 14:11:03 CEST 2020 : Security manager installed using the Basic server security policy.
Thu Apr 30 14:11:04 CEST 2020 : Serwer sieciowy Apache Derby - 10.15.2.0 - (1873585) uruchomiony i gotowy
do zaakceptowania połączeń na porcie 1527 w {3}
```

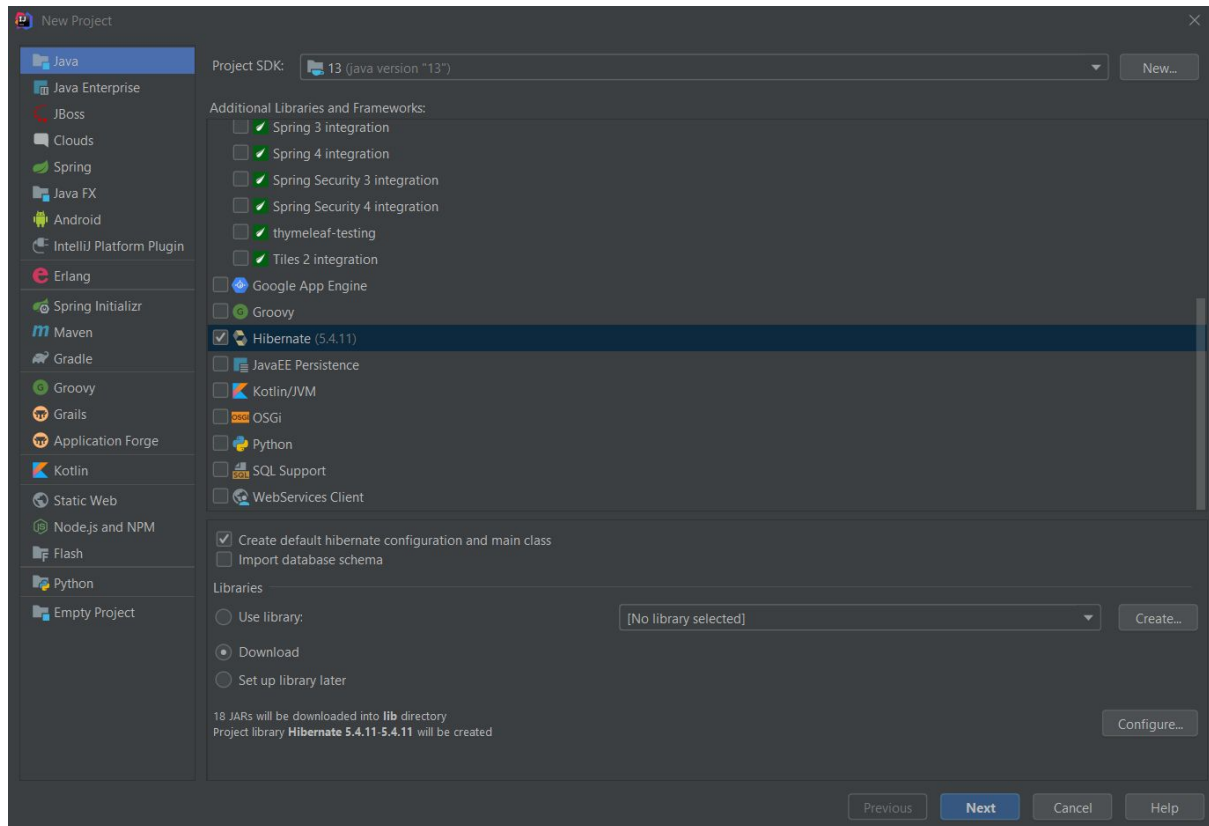
Podpiąłem się do serwera, po czym stworzyłem nową bazę danych. Poniższy zrzut ekranu sugeruje, iż na tym etapie wszystko działa w najlepszym porządku.

```
C:\WINDOWS\system32\cmd.exe
wersja ij 10.15
ij> connect 'jdbc:derby://127.0.0.1/BSzarJPA;create=true';
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       |

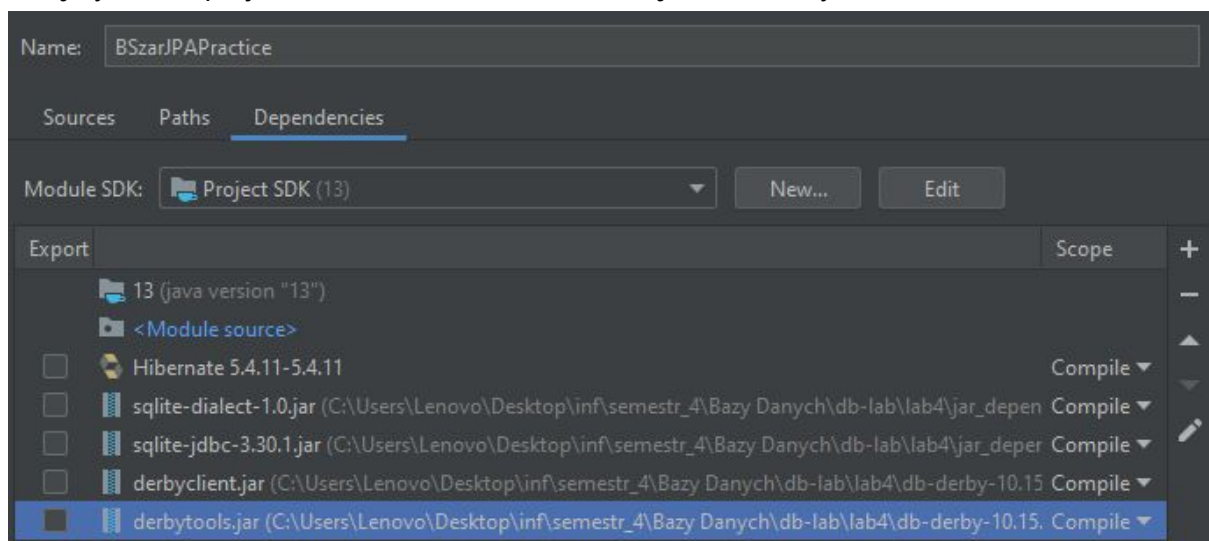
23 wierszy wybranych
ij>
```

1.i - 1.m - Stwórz nowy projekt, dołącz potrzebne zależności, stwórz klasę produktu:

W środowisku IntelliJ stworzyłem nowy projekt.



Dołączyłem do projektu konieczne zależności związane z Derby.



Zmodyfikowałem odpowiednie własności w pliku konfiguracyjnym hibernate.

```
<?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="show_sql">true</property>
        <property name="format_sql">true</property>
        <!-- <property name="dialect">org.hibernate.dialect.SQLiteDialect</property>-->
        <!-- <property name="connection.url">jdbc:derby://127.0.0.1/BSzarJPA</property>-->
        <!-- <property name="connection.driver_class">org.sqlite.JDBC</property>-->
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="connection.url">jdbc:derby://127.0.0.1/BSzarJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"/>
        <mapping class="Supplier"/>
        <!-- DB schema will be updated if needed -->
        <!-- <property name="hibernate.hbm2ddl.auto">update</property> -->
    </session-factory>
</hibernate-configuration>
```

Stworzyłem klasę Produktu oraz uzupełniłem w niej elementy potrzebne do zmapowania klasy produktu. Nadpisałem również metodę toString(), by w bardziej obrazowy sposób wydobywać produkty z bazy.

@Entity

```
public class Product {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
    private int ProductID;
```

```
    private String ProductName;
```

```
    private int UnitsInStock;
```

```
    public Product() {
```

```
}
```

```
    public Product(String ProductName, int UnitsInStock) {
```

```
        this.ProductName = ProductName;
```

```
        this.UnitsInStock = UnitsInStock;
```

```
}
```

```
@Override
```

```
    public String toString() {
```

```
        return "Product(ProductID: " + ProductID + ", ProductName: " +  
        ProductName + ", UnitsInStock: " + UnitsInStock + ")";
```

```
}
```

### 3.a - 3b - Dodaj do bazy przykładowy produkt.

W funkcji main() umieściłem kod odpowiadający za umieszczenie danego produktu w bazie.

```
Product product = new Product("Ucho od śledzia", 1);
final Session session = getSession();
Transaction tx = session.beginTransaction();
session.save(product);
tx.commit();
```

Transakcja przebiegła pomyślnie, co potwierdzają logi wywołań hibernate.

```
Hibernate:
  select
    product0_.ProductID as producti1_0_,
    product0_.ProductName as productn2_0_,
    product0_.UnitsInStock as unitsins3_0_
  from
    Product product0_
  Product(ProductID: 1, ProductName: Ucho od śledzia, UnitsInStock: 1)

Process finished with exit code 0
```

W IntelliJ dodatkowo podłączyłem się do bazy by móc wyświetlić tabelę produktów.

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK
1	1	Ucho od śledzia	1

#### 4 - Zmodyfikuj model wprowadzając pojęcie dostawcy.

Stworzyłem klasę Supplier.

```
@Entity
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;
    }

    @Override
    public String toString() {
        return "Supplier(SupplierID: " + SupplierID + ",
            CompanyName: " + CompanyName + ", Street: " + Street +
            "City: " + City + ")";
    }
}
```

Zmodyfikowałem klasę Product dodając atrybut supplier oraz dodając metodę setSupplier().

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsInStock;
    @ManyToOne
    private Supplier supplier;

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

W pliku konfiguracyjnym hibernate zapewniłem mapowanie klasy Supplier.

```
<mapping class="Product"/>
<mapping class="Supplier"/>
```

4.a - 3.b - Upřednio dodanemu produktowi przypisz nowo stworzonego dostawcę:

W klasie main stworzyłem transakcję przypisującą nowo stworzonego dostawcę do znajdującego się w bazie produktu.

```
final Session session = getSession();
Transaction tx = session.beginTransaction();
Product product = session.find(Product.class,1);
Supplier supplier = new Supplier("Kwinto", "Szeroka", "Lwów");
product.setSupplier(supplier);
session.save(product);
session.save(supplier);
tx.commit();
```

Hibernate:

select

product0\_.ProductID as producti1\_0\_,  
product0\_.ProductName as productn2\_0\_,  
product0\_.UnitsInStock as unitsins3\_0\_,  
product0\_.supplier\_SupplierID as supplier4\_0\_

from

Product product0\_

Product(ProductID: 1, ProductName: Ucho od śledzia, UnitsInStock: 1)

executing: from Supplier

Hibernate:

select

supplier0\_.SupplierID as supplier1\_1\_,  
supplier0\_.City as city2\_1\_,  
supplier0\_.CompanyName as companyn3\_1\_,  
supplier0\_.Street as street4\_1\_

from

Supplier supplier0\_

Supplier(SupplierID: 1, CompanyName: Kwinto, Street: SzerokaCity: Lwów)

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIER_SUPPLIERID
1	1	Ucho od śledzia	1	1

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Lwów	Kwinto	Szeroka



5.a - Odwróć relację między tabelą Product i Supplier, zamodeluj schemat w dwóch wersjach - z i bez tabeli łącznikowej:

wersja z tabelą łącznikową:

Z klasy Product usunąłem atrybut dostawcy i metodę setSupplier().

W klasie Supplier dodałem jako atrybut zbiór produktów, oraz stworzyłem metodę addProduct() dodającą nowy produkt do zbioru produktów danego dostawcy.

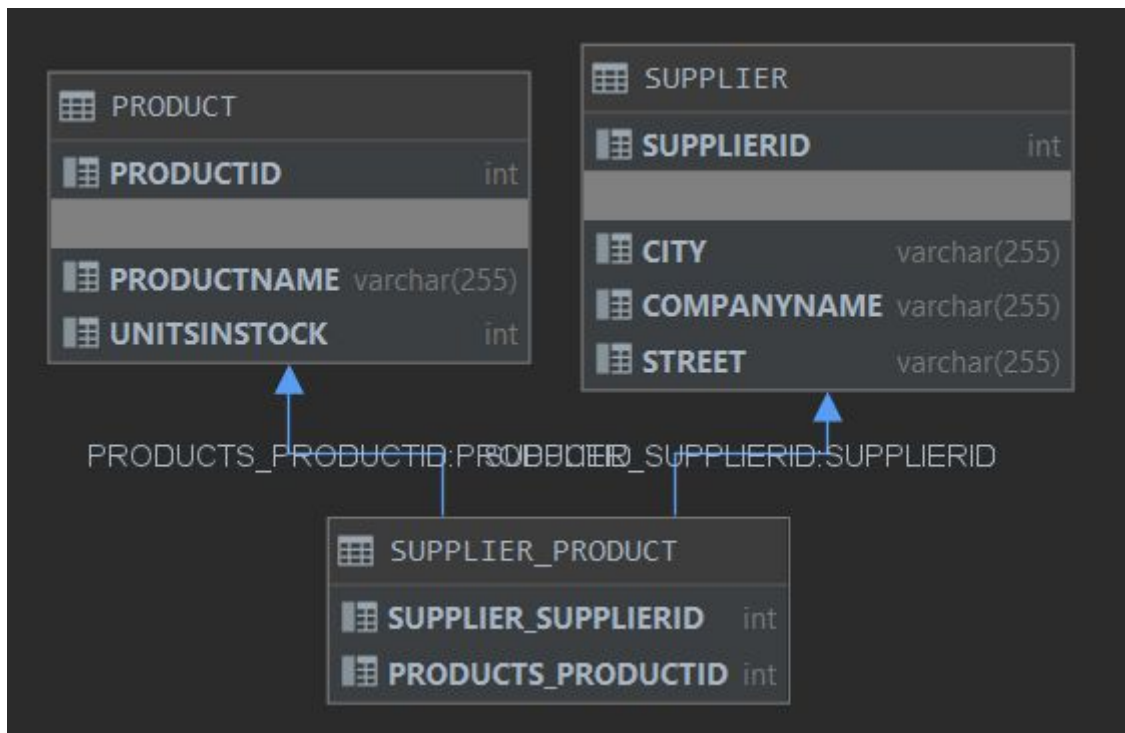
```
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    private Set<Product> products = new LinkedHashSet<>();

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

W klasie main stworzyłem transakcję przypisującą 2 nowo stworzone produkty do dostawcy.

```
final Session session = getSession();
Transaction tx = session.beginTransaction();
Product product = session.find(Product.class, 1);
Supplier supplier = new Supplier("Kramer", "Wąska", "Zurych");
Product product1 = new Product("Czekolada", 2);
Product product2 = new Product("Flakon", 1);
supplier.addProduct(product1);
supplier.addProduct(product2);
session.save(product1);
session.save(product2);
session.save(supplier);
tx.commit();
```

W celu aktualizacji struktury tabel usunąłem je, po czym uruchomiłem funkcję main. Schemat powstałych na nowo tabel przedstawia również tabelę pośredniczącą SUPPLIER\_PRODUCT.

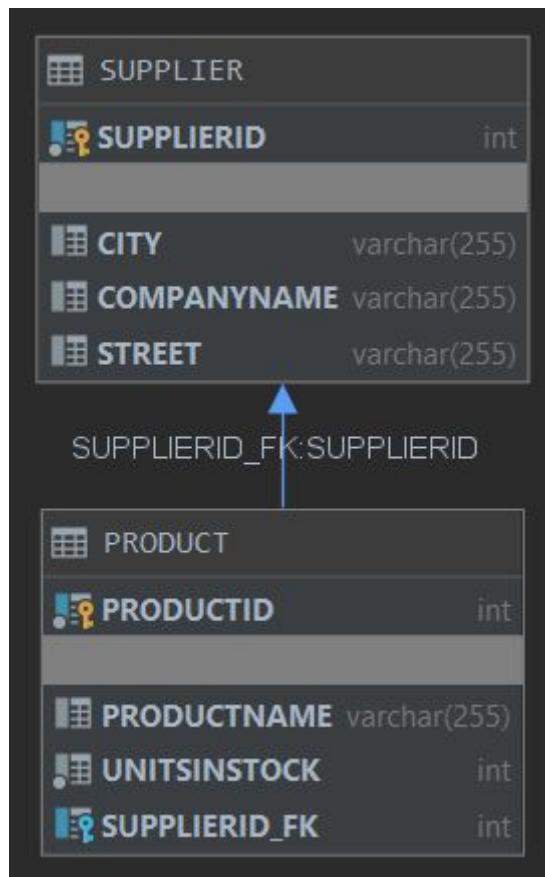


wersja bez tabeli łącznikowej:

W klasie Supplier przy atrybucie reprezentującym zbiór produktów dodałem adnotację @JoinColumn.

```
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    @JoinColumn(name = "SupplierID_FK")
    private Set<Product> products = new LinkedHashSet<>();
}
```





## 6 - Zamodeluj relację dwustronną.

W klasie Product ponownie umieściłem atrybut odpowiadający dostawcy dodatkowo dodając adnotację `@JoinColumn`.

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

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





Klasa Supplier nie wymagała modyfikacji po ostatnim zadaniu.





## 6.a - 6.c - Dodaj kilka nowo stworzonych produktów do nowego dostawcy:

W klasie main stworzyłem transakcję przypisującą 4 nowo stworzone produkty do dostawcy.

```
final Session session = getSession();
Transaction tx = session.beginTransaction();
Product product = session.find(Product.class, 1);
Supplier supplier = new Supplier("Brzeczyszczykiewicz",
                                "Leśna", "Lublin");

Product product1 = new Product("jabłko", 20);
Product product2 = new Product("gruszka", 16);
Product product3 = new Product("śliwka", 40);
Product product4 = new Product("morela", 36);
supplier.addProduct(product1);
supplier.addProduct(product2);
supplier.addProduct(product3);
supplier.addProduct(product4);
product1.setSupplier(supplier);
product2.setSupplier(supplier);
product3.setSupplier(supplier);
product4.setSupplier(supplier);
session.save(product1);
session.save(product2);
session.save(product3);
session.save(product4);
session.save(supplier);
tx.commit();
```

	 SUPPLIERID	 CITY	 COMPANYNAME	 STREET
1	15	Lublin	Brzeczyszczykiewicz	Leśna

	 PRODUCTID	 PRODUCTNAME	 UNITSINSTOCK	 SUPPLIERID_FK
1	11	jabłko	20	15
2	12	gruszka	16	15
3	13	śliwka	40	15
4	14	morela	36	15

7.a - Dodaj klasę Category oraz zmodyfikuj produkty dodając wskazanie na kategorię do której należy:

Stworzyłem klasę Category oraz zmodyfikowałem plik konfiguracyjny hibernate dodając mapping tej klasy.

```
@Entity
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CategoryID;
    private String Name;
    @OneToMany
    private List<Product> products = new ArrayList<>();

    public Category(){

    }

    public Category(String Name){
        this.Name = Name;
    }

    public List<Product> getProducts(){
        return this.products;
    }
}
```

W klasie Product dodałem atrybut wskazujący na kategorię do której należy produkt.

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsInStock;
    @ManyToOne
    @JoinColumn(name = "SupplierID_FK")
    private Supplier supplier;
    @ManyToOne
    @JoinColumn(name = "CategoryID_FK")
    private Category category;

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

7.b - 7.e - Stwórz kilka produktów, dodaj je do danej kategorii a następnie wydobądź produkty z kategorii oraz kategorię z produktu:

W klasie main stworzyłem transakcję tworzącą 4 produkty z 2 kategorii.

```
final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Supplier grzesio = new Supplier("Brzeczyszczykiewicz",
                                    "Leśna", "Lublin");

    Product apple = new Product("jabłko", 20);
    Product pear = new Product("gruszka", 16);
    Product tomato = new Product("pomidor", 40);
    Product pepper = new Product("papryka", 36);
    Category fruits = new Category("owoce");
    Category vegetables = new Category("warzywa");
    grzesio.addProduct(apple);
    grzesio.addProduct(pear);
    grzesio.addProduct(tomato);
    grzesio.addProduct(pepper);
    apple.setSupplier(grzesio);
    pear.setSupplier(grzesio);
    tomato.setSupplier(grzesio);
    pepper.setSupplier(grzesio);
    apple.setCategory(fruits);
    pear.setCategory(fruits);
    tomato.setCategory(vegetables);
```

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORYID_FK	SUPPLIERID_FK
1	16	jabłko	20	21	20
2	17	gruszka	16	21	20
3	18	pomidor	40	22	20
4	19	papryka	36	22	20

	CATEGORYID	NAME
1	21	owoce
2	22	warzywa

Napisałem transakcję wydobywającą produkty danej kategorii.

```
final Session session = getSession();
Transaction tx = session.beginTransaction();
Category category = session.find(Category.class, 21);
List<Product> products = category.getProducts();
for (Product product : products){
    System.out.println(product);
}
tx.commit();
```

where

products0\_.CategoryID\_FK=?

Product(ProductID: 16, ProductName: jabłko, UnitsInStock: 20)

Product(ProductID: 17, ProductName: gruszka, UnitsInStock: 16)

Oraz transakcję wydobywającą kategorię danego produktu.

```
final Session session = getSession();
Transaction tx = session.beginTransaction();
Product product = session.find(Product.class, 18);
System.out.println(product.getCategory());
tx.commit();
```

where

product0\_.ProductID=?

Category(CategoryID: 22, CategoryName: warzywa)

## 8 - Zamodeluj relację wiele-do-wielu między tabelą Invoice i Product.

Stworzyłem klasę Invoice.

```
@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int InvoiceNumber;
    private int Quantity;
    @ManyToMany
    private Set<Product> products = new LinkedHashSet<>();

    public Invoice() {

    }

    public Invoice(int Quantity){
        this.Quantity = Quantity;
    }

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

Zmodyfikowałem klasę Product dodając atrybut invoices oraz metody addInvoice() i getInvoices().

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ProductID;
    private String ProductName;
    private int UnitsInStock;
    @ManyToMany(mappedBy = "products")
    private Set<Invoice> invoices = new LinkedHashSet<>();

    public void addInvoice(Invoice invoice) {
        this.invoices.add(invoice);
    }

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



8.a - 8.d - Stwórz kilka produktów, sprzedaj je w kilku transakcjach, a następnie pokaż produkty sprzedane w ramach wybranej faktury oraz faktury w ramach których sprzedany był wybrany produkt:

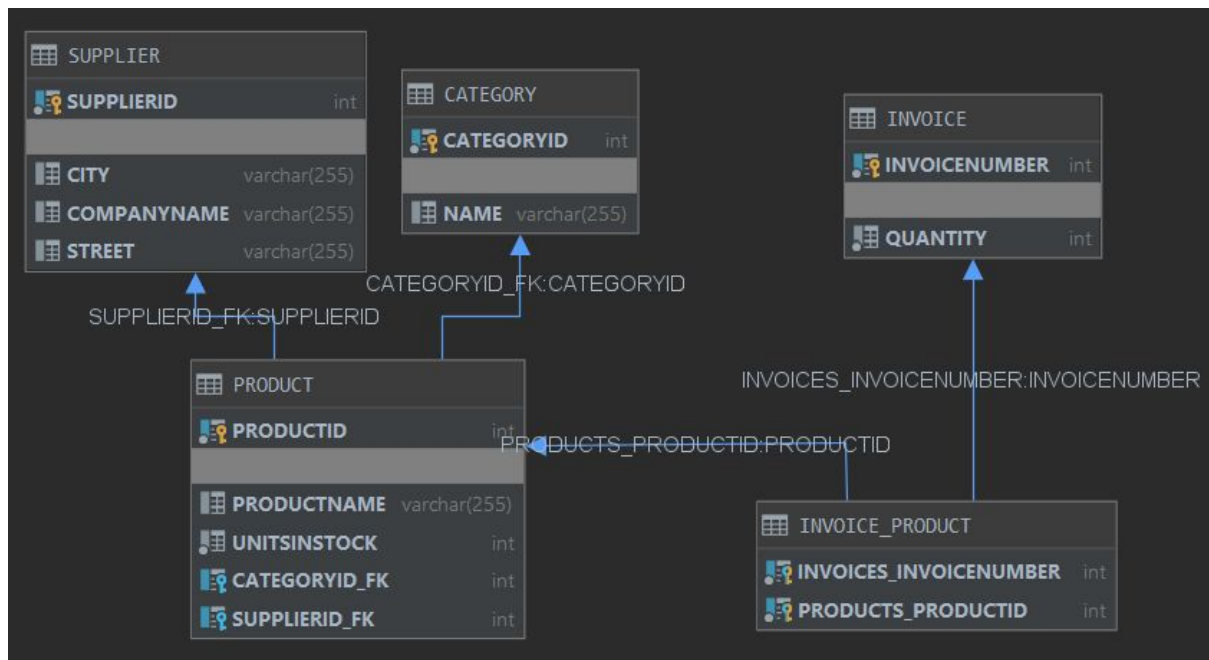
W klasie main stworzyłem transakcję tworzącą 4 produkty z 2 kategorii a następnie "sprzedałem" je w ramach 2 transakcji.

```
final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Supplier supplier1 = new Supplier("companyname1",
        "street1", "city1");
    Product apple = new Product("jabłko", 20);
    Product pear = new Product("gruszka", 16);
    Product tomato = new Product("pomidor", 40);
    Product pepper = new Product("papryka", 36);
    Category fruits = new Category("owoce");
    Category vegetables = new Category("warzywa");
    supplier1.addProduct(apple);
    supplier1.addProduct(pear);
    supplier1.addProduct(tomato);
    supplier1.addProduct(pepper);

    apple.setSupplier(supplier1);
    pear.setSupplier(supplier1);
    tomato.setSupplier(supplier1);
    pepper.setSupplier(supplier1);
    apple.setCategory(fruits);
    pear.setCategory(fruits);
    tomato.setCategory(vegetables);
    pepper.setCategory(vegetables);

    Invoice invoice1 = new Invoice(1);
    Invoice invoice2 = new Invoice(2);

    invoice1.addProduct(apple);
    apple.addInvoice(invoice1);
    invoice1.addProduct(tomato);
    tomato.addInvoice(invoice1);
    invoice2.addProduct(pear);
    pear.addInvoice(invoice2);
    invoice2.addProduct(pepper);
    pepper.addInvoice(invoice2);
```



	INVOICES_INVOICENUMBER	PRODUCTS_PRODUCTID
1	8	1
2	8	4
3	9	2
4	9	3

Produkty sprzedane w ramach wybranej transakcji:

```

Transaction tx = session.beginTransaction();
    Invoice invoice = session.find(Invoice.class, 8);
    invoice.getProducts().forEach(System.out::println);
    tx.commit();
  
```

```

where
    products0_.invoices_InvoiceNumber=?
Product(ProductID: 4, ProductName: pomidor, UnitsInStock: 40)
Product(ProductID: 1, ProductName: jabłko, UnitsInStock: 20)|
  
```

Faktura w ramach której sprzedany został dany produkt:

```

Transaction tx = session.beginTransaction();
    Product product = session.find(Product.class, 1);
    product.getInvoices().forEach(System.out::println);
    tx.commit();
  
```

```

where
    products0_.invoices_InvoiceNumber=?
Invoice(InvoiceNumber: 8, Quantity: 1Products[Product(ProductID: 4, ProductName: pomidor, UnitsInStock: 40), Product(ProductID: 1, ProductName: jabłko, UnitsInStock: 20)])
  
```

9.a - 9.b - Stwórz nowego main w którym zrobisz to samo co w punkcie 6, ale z wykorzystaniem JPA:

Stworzyłem plik konfiguracyjny persistence.xml.

```
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="myDatabaseConfig" 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/B5SzacJPA"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.DerbyTenSevenDialect"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.hbm2ddl.auto" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Utworzyłem nową klasę main w której wprowadziłem wszystkie potrzebne zmiany.

```
public class JPAMain {

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

        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();]

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

Następnie przywróciłem schemat bazy z punktu 6, po czym dodałem do bazy kilka produktów oraz przypisałem im nowo stworzonego dostawcę.

```
Supplier supplier = new Supplier("Brzeczyszczykiewicz",
    "Leśna", "Lublin");
Product product1 = new Product("jabłko", 20);
Product product2 = new Product("gruszka", 16);
Product product3 = new Product("śliwka", 40);
Product product4 = new Product("morela", 36);
supplier.addProduct(product1);
supplier.addProduct(product2);
supplier.addProduct(product3);
supplier.addProduct(product4);
product1.setSupplier(supplier);
product2.setSupplier(supplier);
product3.setSupplier(supplier);
product4.setSupplier(supplier);

em.persist(product1);
em.persist(product2);
em.persist(product3);
em.persist(product4);
em.persist(supplier);
```

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORYID_FK	SUPPLIERID_FK
1	1	jabłko	20	<null>	5
2	2	gruszka	16	<null>	5
3	3	śliwka	40	<null>	5
4	4	morela	36	<null>	5

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	5	Lublin	Brzeczyszczykiewicz	Leśna

10.a - 10.b - Zmodyfikuj model w taki sposób, aby było możliwe kaskadowe tworzenie faktur wraz z nowymi produktami, oraz produktów wraz z nową fakturą:

Wprowadziłem modyfikacje w klasach: Product, Category, Supplier oraz Invoice. Poniżej prezentuję dodane frazy w każdej z klas.

```
public class Product {
    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "SupplierID_FK")
    private Supplier supplier;
    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "CategoryID_FK")
    private Category category;
    @ManyToMany(mappedBy = "products", cascade = CascadeType.PERSIST)
    private Set<Invoice> invoices = new LinkedHashSet<>();
}
```

```
public class Category {
    @OneToMany(cascade = CascadeType.PERSIST)
    @JoinColumn(name = "CategoryID_FK")
    private List<Product> products = new ArrayList<>();
}
```

```
public class Supplier {
    @OneToMany(mappedBy = "supplier", cascade = CascadeType.PERSIST)
    private Set<Product> products = new LinkedHashSet<>();
}
```

```
public class Invoice {
    @ManyToMany(cascade = CascadeType.PERSIST)
    private Set<Product> products = new LinkedHashSet<>();
}
```

W celu przetestowania poprawności działania kaskadowego utrwalenia `em.persist()` wywołuję tylko z nowo utworzonymi fakturami.

```
Supplier supplier = new Supplier( CompanyName: "Brzeczyszczkiewicz", Street: "Leśna", City: "Lublin");
Product product1 = new Product( ProductName: "jabłko", UnitsInStock: 20);
Product product2 = new Product( ProductName: "marchewka", UnitsInStock: 16);
Product product3 = new Product( ProductName: "śliwka", UnitsInStock: 40);
Product product4 = new Product( ProductName: "brokuł", UnitsInStock: 36);
Category category1 = new Category( Name: "owoce");
Category category2 = new Category( Name: "warzywa");

supplier.addProduct(product1);
supplier.addProduct(product2);
supplier.addProduct(product3);
supplier.addProduct(product4);
product1.setSupplier(supplier);
product2.setSupplier(supplier);
product3.setSupplier(supplier);
product4.setSupplier(supplier);
product1.setCategory(category1);
product2.setCategory(category2);
product3.setCategory(category1);
product4.setCategory(category2);

Invoice invoice1 = new Invoice( Quantity: 1);
Invoice invoice2 = new Invoice( Quantity: 2);

invoice1.addProduct(product1);
invoice1.addProduct(product2);
invoice2.addProduct(product3);
invoice2.addProduct(product4);
product1.addInvoice(invoice1);
product2.addInvoice(invoice1);
product3.addInvoice(invoice2);
product4.addInvoice(invoice2);

em.persist(invoice1);
em.persist(invoice2);
```

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORYID_FK	SUPPLIERID_FK
1	5	marchewka	16	6	4
2	7	śliwka	40	3	4
3	9	brokuł	36	6	4
4	2	jabłko	20	3	4

	INVOICES_INVOICENUMBER	PRODUCTS_PRODUCTID
1	1	2
2	1	5
3	8	7
4	8	9



11.a - 11.b - Dodaj do modelu klasę adres i “wbuduj” ją do tabeli dostawców:

Stworzyłem klasę Address.

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

    public Address() {

    }

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

W klasie Supplier zmodyfikowałem atrybuty oraz konstruktor w taki sposób, aby przy tworzeniu nowego dostawcy przekazać mu obiekt klasy Address zamiast osobnego przekazywania miasta i ulicy.

```
@Entity
public class Supplier {
    @Embedded
    private Address address;

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

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

    @Override
    public String toString() {
        return "Supplier(SupplierID: " + SupplierID + ", CompanyName: "
            + CompanyName + address.toString() + ")";
    }
}
```

Następnie przetestowałem wprowadzoną modyfikację tworząc obiekt klasy Address po czym wywołałem konstruktor nowego dostawcy z tymże adresem.

```
public class JPAMain {

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

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();

        Address address = new Address("Zawiasowa", "Szuflandia");
        Supplier supplier = new Supplier("Heniek", address);
        em.persist(supplier);

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

    }
}
```

	SUPPLIERID	COMPANYNAME	CITY	STREET
1	1	Heniek	Szuflandia	Zawiasowa

11.c - 11.d - Zmodyfikuj model w taki sposób, że dane adresowe znajdują się w klasie dostawców. Zmapuj to do dwóch osobnych tabel:

```
@Entity
@SecondaryTable(name = "ADDRESS")
public class Supplier {
    @Column(table = "ADDRESS")
    private String Street;
    @Column(table = "ADDRESS")
    private String City;

    public Supplier(String CompanyName, String street, String city){
        this.CompanyName = CompanyName;
        this.Street = street;
        this.City = city;
    }
}
```

```

public class JPAMain {

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

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();

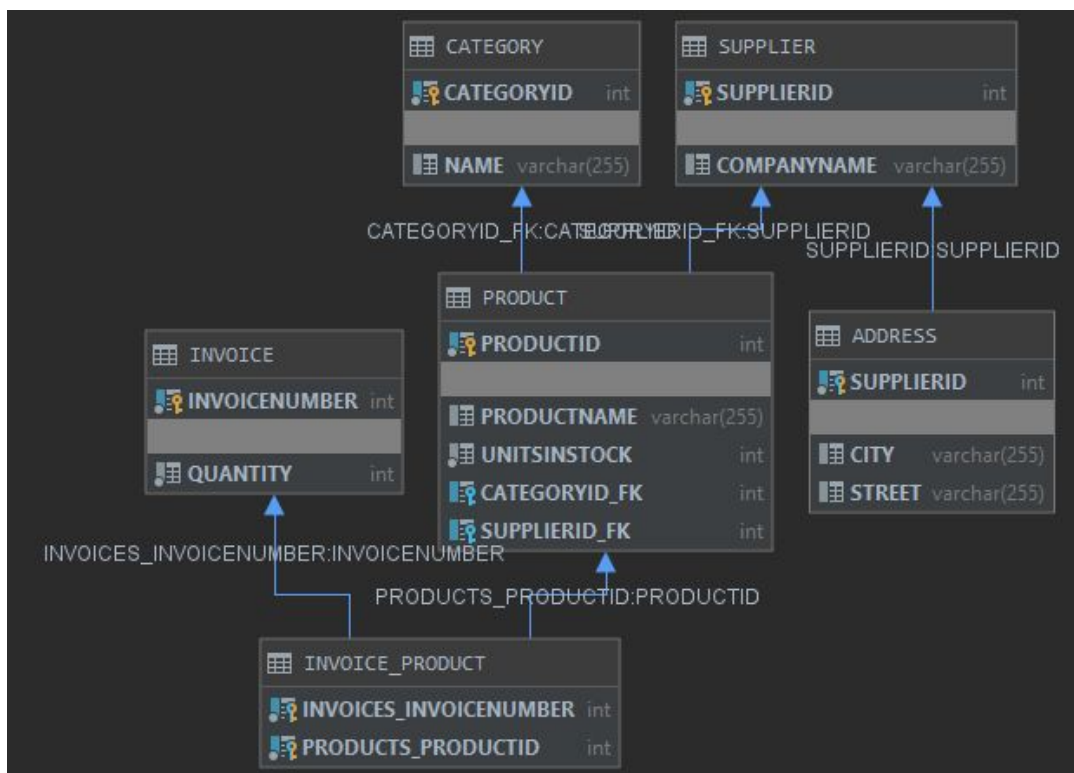
        Supplier supplier = new Supplier("Heniek",
            "Zawiasowa", "Szuflandia");
        em.persist(supplier);

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

```

SUPPLIERID	COMPANYNAME
1	Heniek

CITY	STREET	SUPPLIERID
Szuflandia	Zawiasowa	1



12.a - 12.c - Wprowadź do modelu hierarchię dziedziczenia, w której klasy Supplier i Customer dziedziczą po klasie Company, po czym dodaj i pobierz kilka firm obu rodzajów stosując po kolei trzy różne strategie mapowania dziedziczenia:

Single table:

Stworzyłem klasę Company.

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int CompanyID;

    private String CompanyName;
    private String Street;
    private String City;
    private String ZipCode;

    public Company() {

    }

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

Zmodyfikowałem klasę Supplier uwzględniając dziedziczenie po klasie Company.

```
@Entity
public class Supplier extends Company {

    private String bankAccountNumber;

    @OneToMany(mappedBy = "supplier", cascade = CascadeType.PERSIST)
    private Set<Product> products = new LinkedHashSet<>();

    public Supplier() {
        super();
    }

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

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

Stworzyłem klasę Customer, również uwzględniając dziedziczenie o klasie Company.

```
@Entity
public class Customer extends Company {

    private int Discount;

    public Customer() {
        super();
    }

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

Przetestowałem działanie bazy po modyfikacji dodając po 3 dostawców i konsumentów.

```
public class JPAMain {

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

        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("myDatabaseConfig");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();

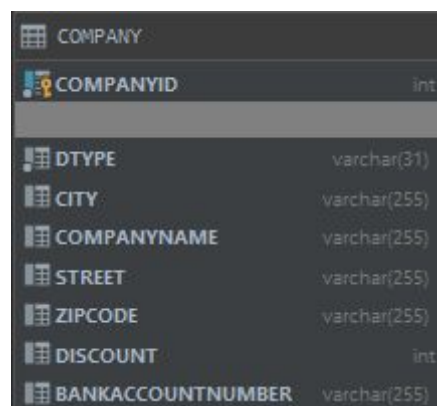
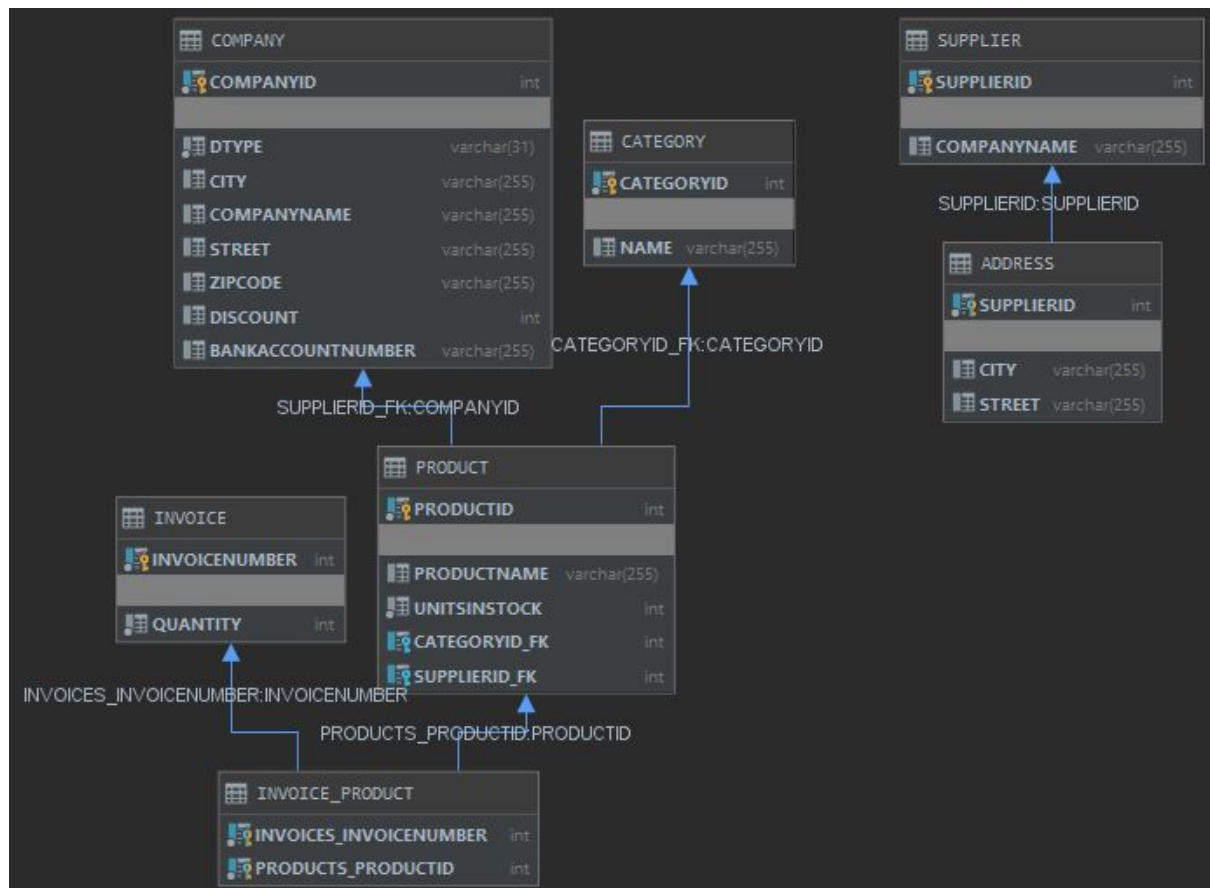
        Supplier supplier1 = new Supplier("Heniek", "Zawiasowa",
            "Szuflandia", "32-512", "12345678900000");
        Supplier supplier2 = new Supplier("KrzysiekPOL", "Chmielowa",
            "Żywiec", "22-312", "00000123456789");
        Supplier supplier3 = new Supplier("JabłkoINC", "Sadowa",
            "Sadów", "32-612", "696969123456789");
        Customer customer1 = new Customer("Marian", "Szkłana", "Denko",
            "13-312", 10);
        Customer customer2 = new Customer("PolINC", "Makrelowa",
            "Tychy", "37-313", 12);
        Customer customer3 = new Customer("StalBud", "Ceglana",
            "Pustakowo", "82-322", 69);

        em.persist(supplier1);
        em.persist(supplier2);
        em.persist(supplier3);
        em.persist(customer1);
        em.persist(customer2);
        em.persist(customer3);

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

	DTYPE	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT	BANKACCOUNTNUMBER
1	Supplier	1	Szuflandia	Heniek	Zawiasowa	32-512	<null>	12345678900000
2	Supplier	2	Żywiec	KrzysiekPOL	Chmielowa	22-312	<null>	00000123456789
3	Supplier	3	Sadów	JabłkoINC	Sadowa	32-612	<null>	696969123456789
4	Customer	4	Denko	Marian	Szkłana	13-312	10	<null>
5	Customer	5	Tychy	PolINC	Makrelowa	37-313	12	<null>
6	Customer	6	Pustakowo	StalBud	Ceglana	82-322	69	<null>



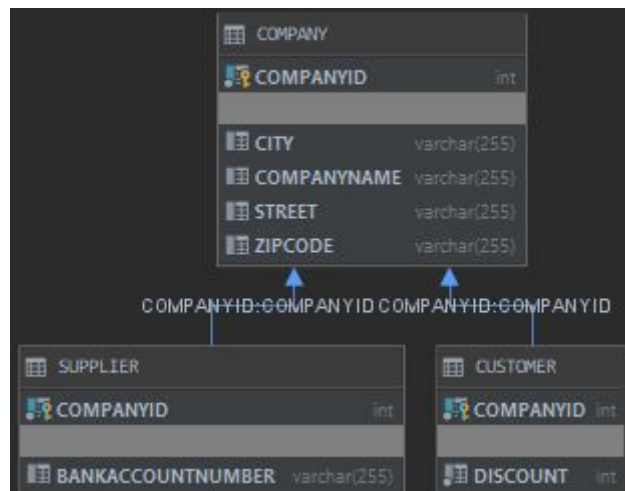


Joined:

Zmodyfikowałem strategię mapowania w klasie Company.

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Company {

}
```



	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE
1	1	Szuflandia	Heniek	Zawiasowa	32-512
2	2	Żywiec	KrzysiekPOL	Chmielowa	22-312
3	3	Sadów	JabłkoINC	Sadowa	32-612
4	4	Denko	Marian	Szklana	13-312
5	5	Tychy	PolINC	Makrelowa	37-313
6	6	Pustakowo	StalBud	Ceglana	82-322

	BANKACCOUNTNUMBER	COMPANYID
1	12345678900000	1
2	00000123456789	2
3	696969123456789	3

	DISCOUNT	COMPANYID
1	10	4
2	12	5
3	69	6

Table per class:

Ponownie zmodyfikowałem strategię mapowania w klasie Company.

```

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Company {

}
  
```

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT
1	4	Denko	Marian	Szklana	13-312	10
2	5	Tychy	PolINC	Makrelowa	37-313	12
3	6	Pustakowo	StalBud	Ceglana	82-322	69

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	1	Szuflandia	Heniek	Zawiasowa	32-512	12345678900000
2	2	Żywiec	KrzysiekPOL	Chmielowa	22-312	00000123456789
3	3	Sadów	JabłkoINC	Sadowa	32-612	696969123456789