

Laboratorium 3 – Hibernate

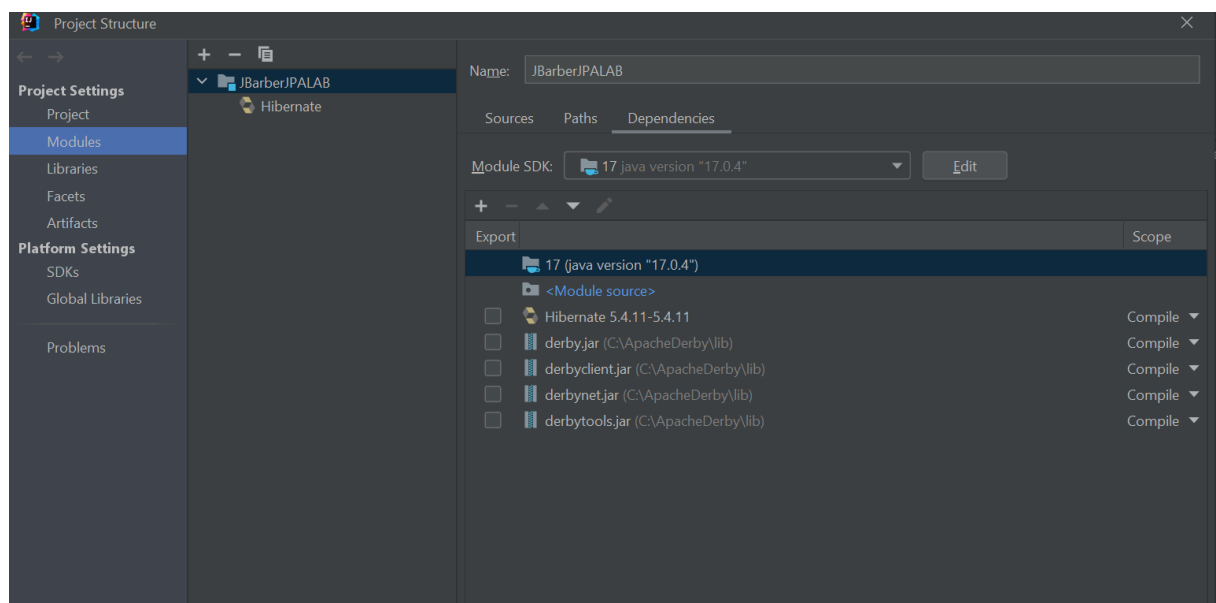
Jakub Barber

Po pobraniu ze wskazanego adresu odpowiedniej wersji Apache Derby dla mojej wersji Javy (17), uruchomiłem skrypt startNetworkServer w celu przetestowania działania serwera Derby.

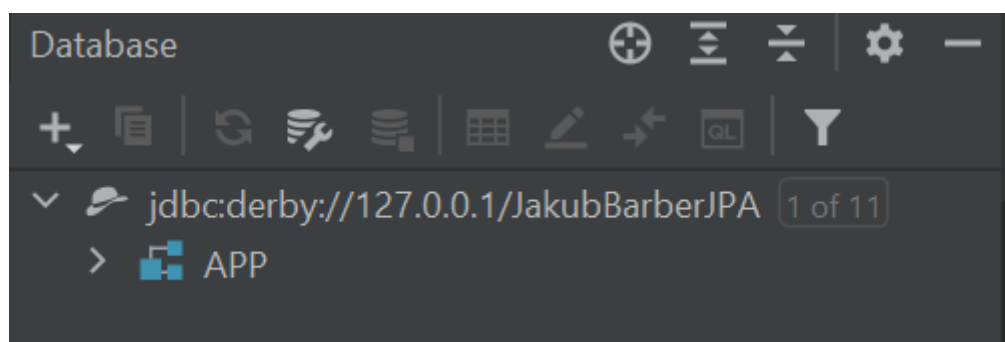
```
C:\ApacheDerby\bin>startNetworkServer
Wed Apr 26 13:06:03 CEST 2023 : Serwer sieciowy Apache Derby - 10.16.1.1 - (1901046) uruchomiony i gotowy do zaakceptowania połączeń na porcie 1527 w {3}
```

Po odpowiedniej konfiguracji ustawień projektu zgodnie z zaleceniami, sytuacja w IntelliJ prezentuje się następująco :

Moduły projektu:



Stan połączenia z bazą danych:



Konfiguracja Hibernate'a (plik hibernate.cfg.xml):

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

Klasa Main:

```
package org.example;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

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[] args) throws Exception {
        final Session session = getSession();
        try {
        } finally {
            session.close();
        }
    }
}
```

Wynik odpalenia program w klasie Main:

```
"C:\Program Files\Java\jdk-17.0.4.1\bin\java.exe" ...
kwi 26, 2023 1:16:04 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate Core {5.4.11.Final}
kwi 26, 2023 1:16:04 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
INFO: HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
kwi 26, 2023 1:16:05 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
kwi 26, 2023 1:16:05 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: using driver [org.apache.derby.jdbc.ClientDriver] at URL [jdbc:derby://127.0.0.1/JakubBarberJPA;create=true]
kwi 26, 2023 1:16:05 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {}
kwi 26, 2023 1:16:05 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
kwi 26, 2023 1:16:05 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
kwi 26, 2023 1:16:06 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.DerbyTenSevenDialect
kwi 26, 2023 1:16:07 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
```

Wynik jest zgodny z oczekiwaniami, jedyna różnica w stosunku do Upelowego przewodnika to wystąpienie jednego warninga, ale program nie wyrzuca póki co żadnych wyjątków.

Praca z modelem:

ZADANIE 1

Stworzyłem nową klasę i uzupełniłem odpowiednimi adnotacjami Hibernateowymi w celu odpowiedniego zmapowania modelu na bazę danych

```
package org.example;

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

@Entity
public class Product {
    public Product() {}
    @Id
    @GeneratedValue( strategy = GenerationType.AUTO)
    private int ProductID;

    private String ProductName;
    private int UnitsOnStock;
```

```
}
```

Po uruchomieniu Maina otrzymałem błąd Entity Class Product not found, w celu rozwiązania problemu przy tworzeniu obiektu configuration dodałem linię:

```
configuration.addAnnotatedClass(Product.class);
```

Po ponownym uruchomieniu otrzymałem oczekiwany rezultat:

```
Hibernate:

    drop sequence hibernate_sequence restrict
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:

    create table Product (
      ProductID integer not null,
      ProductName varchar(255),
      UnitsOnStock integer not null,
      primary key (ProductID)
    )
kwi 26, 2023 1:40:09 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
```

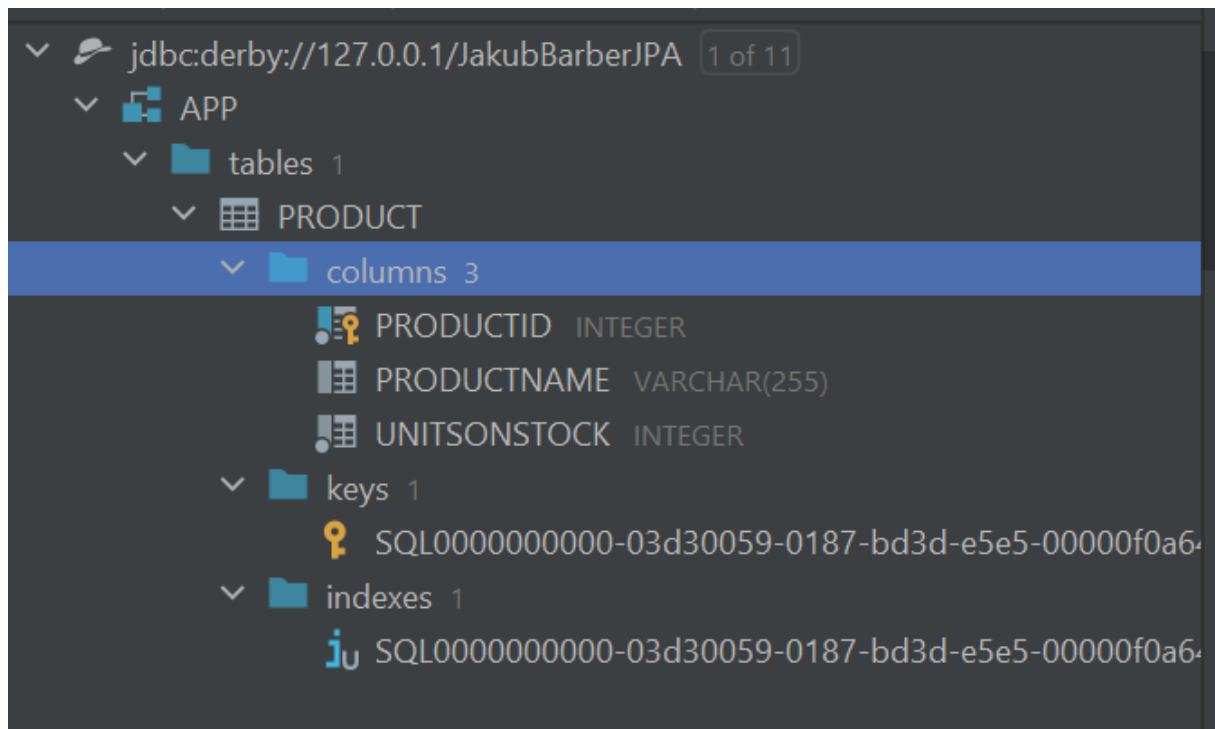
```
Hibernate:

values
    next value for hibernate_sequence
Hibernate:

    /* insert org.example.Product
    */ insert
    into
        Product
        (ProductName, UnitsOnStock, ProductID)
    values
        (?, ?, ?)

Process finished with exit code 0
```

Wynik w zakładce Database:



ZADANIE 2

W celu zrealizowania relacji ManyToOne stworzyłem nową klasę Product i dodałem w niej odpowiednią adnotację pola @ManyToOne

```
package org.example;

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

@Entity
public class Product {
    public Product() {}

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

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

    private String ProductName;
    private int UnitsOnStock;

    @ManyToOne
    private Supplier supplier ;
}
```

Następnie stworzyłem klasę Supplier

```
package org.example;

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

@Entity
public class Supplier {
    public Supplier() {
    }

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

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

    private String companyName;
    private String Street;
    private String City;
}
```

Po wykonaniu tych czynności w klasie Main stworzyłem nowy produkt i
ustaliłem jego suppliersa na nowo utworzonego suppliersa:

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Product product = new Product("Krzyszto", 111);
    Supplier supplier = new Supplier("Chemwit", "Golkowicka", "Krakow");
    // Product product = session.get(Product.class, 0);

    try {
        Transaction tx = session.beginTransaction();
        // product = session.get(Product.class, 1);
        product.setSupplier(supplier);
        session.save(supplier);
        session.save(product);
        tx.commit();
    } finally {
        session.close();
    }
}
```

Po wykonaniu program nie wyrzucił wyjątków więc sprawdzam wynik w bazie danych podłączając się do bazy Apache Derby przy użyciu narzędzia DataGrip.

```
1 ✓ SELECT * FROM PRODUCT;  
2 ✓ SELECT * FROM SUPPLIER;
```

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	2	Krzesło	111	1

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Krakow	Chemwit	Gólkowicka

Po zmianie w konfiguracji hibernate'a z create-drop na update dla pewności dodaje jeszcze jeden produkt i linkuję z tym samym dostawcą:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	2	Krzesło	111	1
2	3	Stół	112	1

ZADANIE 3

W celu realizacji tym razem relacji OneToMany modyfikujemy odpowiednio nasze dotychczasowe klasy:

W klasie Supplier dodaję następujący kod odpowiedzialny za handling relacji OneToMany:

```
@OneToMany  
private Set<Product> productsGroup;  
  
public Set<Product> getProductsGroup() {  
    return productsGroup;  
}  
  
public void addToProductsGroup(Product product) {  
    if (this.productsGroup==null) {  
        this.productsGroup = new HashSet<Product>();  
    }  
    this.productsGroup.add(product);  
}
```

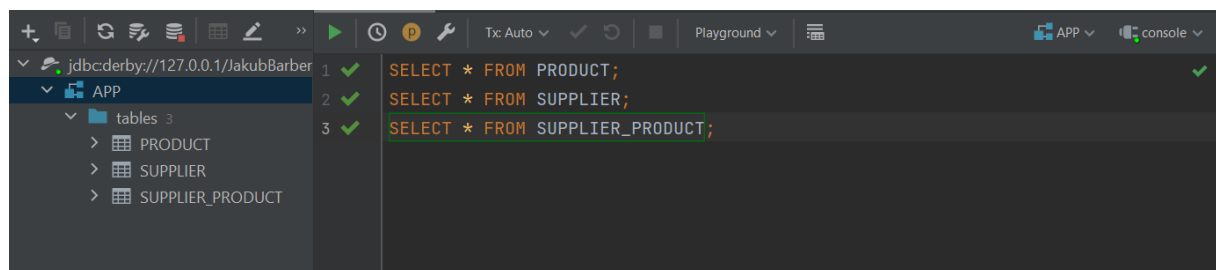
W klasie Product naturalnie zakomentowałem adnotację z poprzedniego zadania.

Po wykonaniu kodu klasy Main:

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Product product1 = new Product("Stolik",112);
    Product product2 = new Product("Talerz",11);
    Product product3 = new Product("Widelec",22);
    Supplier supplier = new Supplier("Chemwit","Golkowicka","Krakow");
    // Product product = session.get(Product.class,0);
    // Supplier supplier = session.get(Supplier.class,1);

    try {
        Transaction tx = session.beginTransaction();
        // product = session.get(Product.class,1);
        // product.setSupplier(supplier);
        // session.save(supplier);
        session.save(product1);
        session.save(product2);
        session.save(product3);
        supplier.addToProductsGroup(product1);
        supplier.addToProductsGroup(product2);
        supplier.addToProductsGroup(product3);
        session.save(supplier);
        tx.commit();
    } finally {
        session.close();
    }
}
```

Sprawdzamy wynik w Datagripie:



Widzimy, że w bazie utworzyła się nowa tabela mapująca relacje dwóch tabeli Product i Supplier, zobaczymy wyniki Selectów:

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	1	Stolik	112
2	2	Talerz	11
3	3	Widelec	22

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Krakow	Chemwit	Gólkowicka

	SUPPLIER_SUPPLIERID	PRODUCTSGROUP_PRODUCTID
1	4	1
2	4	2
3	4	3

Wynik jest zgodny z oczekiwaniami.

Zgodnie z treścią polecenia, zrealizujemy teraz ten przykład bez tworzenia dodatkowej tablicy słownikowej, co przyszczędzi pewnym stopniem niepotrzebnej redundancji danych:

W klasie Supplier dodajemy adnotacje @JoinColumn

Jako, że hibernate rzucał błędami prawdopodobnie w związku z próbą updateu/creatu nowej schemy dla bazy danych (co ciekawe przy ustawieniu w configu create oraz create-drop), postanowiłem wykonać DROP TABLE na tabeli słownikowej i uruchomić ten sam kod w klasie Main dla zmodyfikowanej klasy Supplier.

```
@OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
@JoinColumn(name="supplier_id")
private Set<Product> productsGroup;

public Set<Product> getProductsGroup() {
    return productsGroup;
}
```

Wynik w bazie danych:

1	✓	SELECT * FROM PRODUCT;
2	✓	SELECT * FROM SUPPLIER;

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_ID
1	1	Stolik	112	4
2	2	Talerz	11	4
3	3	Widelec	22	4

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Krakow	Chemwit	Gołkowicka

Jak widać rozwiązanie obejmuje teraz dwie tabele, tym razem przy pomocy adnotacji @JoinColumn dodajemy klucz obcy do tabeli Products wskazujący na rekordy z Supplier.

ZADANIE 4

W celu zrealizowania relacji dwustronnej łączymy dwa powyższe podejścia. W tym celu dodajemy na nowo adnotację @ManyToOne w klasie Product i dodajemy dodatkową metodę w klasie Product, która jednocześnie ustawia referencję w dwie strony przy tej relacji.

Klasa Product:

```
@ManyToOne
private Supplier supplier ;

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

Modyfikacja kodu w klasie Main :

```
product1.setSupplier(supplier);
product2.setSupplier(supplier);
product3.setSupplier(supplier);
```

Wyniki w bazie danych:

```

1 ✓ SELECT * FROM PRODUCT;
2 ✓ SELECT * FROM SUPPLIER;
3 ✓ SELECT * FROM SUPPLIER_PRODUCT;

```

	SUPPLIER_SUPPLIERID	PRODUCTSGROUP_PRODUCTID
1	4	1
2	4	2
3	4	3

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Krakow	Chemwit	Golkowicka

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	1	Stolik	112	4
2	2	Talerz	11	4
3	3	Widelec	22	4

ZADANIE 5

Dodajemy nową klasę Category, w której przy pomocy adnotacji @OneToMany będziemy mapować relacje z tabelą Product. Dla czytelności dodałem adnotację @JoinColumn, żeby nie tworzyć dodatkowej tabeli.

```

package org.example;

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

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

    public Category() {
    }

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

```

```

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

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

```

Po wykonaniu kodu w klasie Main:

```

public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Product product1 = new Product("Stolik",112);
    Product product2 = new Product("Talerz",11);
    Product product3 = new Product("Widelec",22);
    Product product4 = new Product("lalka barbie",3);
    Product product5 = new Product("buzz astral",8);
    Supplier supplier1 = new Supplier("Chemwit","Golkowicka","Krakow");
    Supplier supplier2 = new Supplier("Zabawki","Zabawowa","Krakow");
    Category category1 = new Category("Dom");
    Category category2 = new Category("Zabawki");
    // Product product = session.get(Product.class,0);
    // Supplier supplier = session.get(Supplier.class,1);

    try {
        Transaction tx = session.beginTransaction();
        // product = session.get(Product.class,1);
        // product.setSupplier(supplier);
        // session.save(supplier);
        session.save(product1);
        session.save(product2);
        session.save(product3);
        category1.addProduct(product1);
        category1.addProduct(product2);
        category1.addProduct(product3);
        category2.addProduct(product4);
        category2.addProduct(product5);
        // supplier.addToProductsGroup(product1);
        // supplier.addToProductsGroup(product2);
        // supplier.addToProductsGroup(product3);
        product1.setSupplier(supplier1);
        product2.setSupplier(supplier1);
        product3.setSupplier(supplier1);
        product4.setSupplier(supplier2);
        product5.setSupplier(supplier2);
        session.save(supplier1);
        session.save(supplier2);
        session.save(category1);
        session.save(category2);
        tx.commit();
    } finally {
        session.close();
    }
}

```

```
}  
}
```

Otrzymujemy następujące rezultaty analizując wyniki w DataGripie:

Naturalnie została utworzona nowa tabela Category, i pole klucza obcego wskazujące na tabelę Category zostało dodane do tabeli Product

```
SELECT * FROM PRODUCT;  
SELECT * FROM SUPPLIER;  
SELECT * FROM SUPPLIER_PRODUCT;  
SELECT * FROM CATEGORY;
```

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID	CATEGORY_ID
1	1	Stolik	112	4	8
2	2	Talerz	11	4	8
3	3	Widielec	22	4	8
4	6	buzz astral	8	5	9
5	7	lalka barbie	3	5	9

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Krakow	Chemwit	Golkowicka
2	5	Krakow	Zabawki	Zabawowa

	SUPPLIER_SUPPLIERID	PRODUCTSGROUP_PRODUCTID
1	4	1
2	4	2
3	4	3
4	5	6
5	5	7

	CATEGORYID	NAME
1	8	Dom
2	9	Zabawki

W celu przetestowania pobierania danych z bazy bez ich usuwania przy każdej nowej transakcji, zmieniłem ustawienie w configu na update oraz wykonałem w klasie Main pobranie kategorii i jej produktów, wyniki poniżej (niestety hibernate w połowie bloku wykonuje zapytanie SQL więc wynik jest oddzielony tym zapytaniem).

```
Product product = session.get(Product.class,1);
Category category = session.get(Category.class,8);
```

```
List<Product> products = category.getProducts();
System.out.println("Nazwa kategorii :"+category.getName());
for (Product prod : products){
    System.out.println(prod.getProductName());
}
System.out.println("Category of "+product.getProductName());
```

ZADANIE 6

W tym poleceniu tworzymy nową klasę Invoice oraz tworzymy relację ManyToMany pomiędzy Invoice, a Product.

```
package org.example;

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

@Entity
public class Invoice {

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

    private int Quantity;

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

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

    public Invoice() {
    }

    @ManyToMany
```

```
private Set<Product> products= new HashSet<>();  
}
```

oraz dodajemy nowe pole w klasie Product:

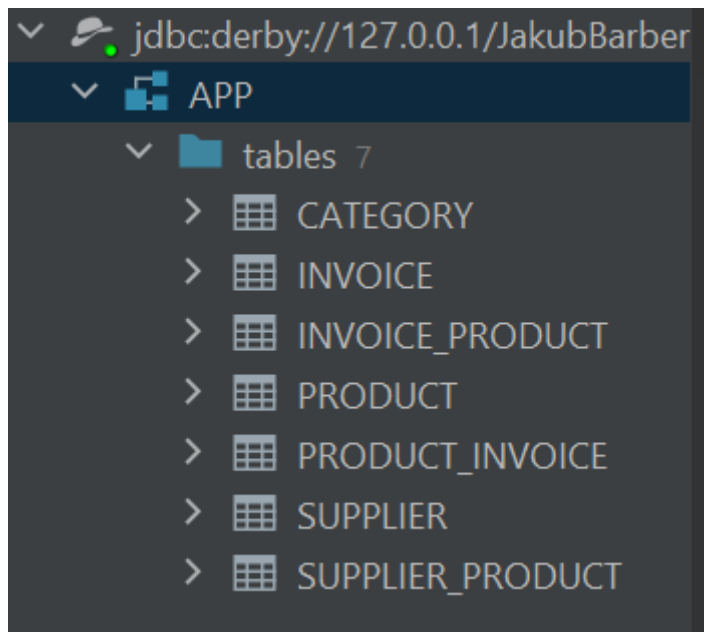
```
@ManyToMany  
private Set<Invoice> invoices= new HashSet<>();
```

W kodzie klasy Main rejestrujemy nową Entity klasę oraz odpalamy kod w klasie Main wzbogacony o poniższe linie (reszta jak we wcześniejszym przykładzie):

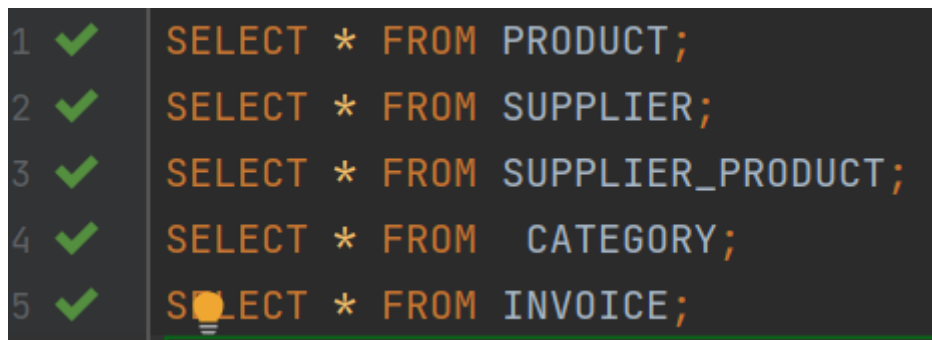
```
Invoice invoice1 = new Invoice(3);  
Invoice invoice2 = new Invoice(2);  
product1.setSupplier(supplier1);  
product1.addToInvoice(invoice1);  
  
product2.setSupplier(supplier1);  
product2.addToInvoice(invoice1);  
  
product3.setSupplier(supplier1);  
product3.addToInvoice(invoice1);  
  
product4.setSupplier(supplier2);  
product4.addToInvoice(invoice2);  
  
product5.setSupplier(supplier2);  
product5.addToInvoice(invoice2);  
  
session.save(supplier1);  
session.save(supplier2);  
session.save(category1);  
session.save(category2);  
session.save(invoice1);  
session.save(invoice2);
```

Badamy rezultat w DataGripie:

Widzimy, że poza oczywistym faktem utworzenia tabeli Invoice, została również utworzona tabela reprezentująca mapowanie relacji ManyToMany pomiędzy tabelami Product i Invoice.







Zobaczmy wyniki zapytań SELECT :





	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID	CATEGORY_ID
1	1	Stolik	112	4	8
2	2	Talerz	11	4	8
3	3	Widelec	22	4	8
4	6	buzz astral	8	5	9
5	7	lalka barbie	3	5	9

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Krakow	Chemwit	Golkowicka
2	5	Krakow	Zabawki	Zabawowa

	 SUPPLIER_SUPPLIERID ▾	 PRODUCTSGROUP_PRODUCTID ▾
1	4	1
2	4	2
3	4	3
4	5	6
5	5	7

	 CATEGORYID ▾	 NAME ▾
1	8	Dom
2	9	Zabawki

	 INVOICENUMBER ▾	 QUANTITY ▾
1	10	3
2	11	2

Wyniki są zgodne z oczekiwaniami. Wykonajmy jeszcze zapytanie o już istniejące w bazie danych w klasie Main:

Numer faktury :10

Hibernate:

```
select
    products0_.Invoice_InvoiceNuber as invoice_1_2_0_,
    products0_.products_ProductID as products2_2_0_,
    product1_.ProductID as producti1_3_1_,
    product1_.ProductName as productn2_3_1_,
    product1_.UnitsOnStock as unitsons3_3_1_,
    product1_.supplier_SupplierID as supplier4_3_1_,
    supplier2_.SupplierID as supplier1_5_2_,
    supplier2_.City as city2_5_2_,
    supplier2_.CompanyName as companyn3_5_2_,
    supplier2_.Street as street4_5_2_
from
    Invoice_Product products0_
inner join
    Product product1_
        on products0_.products_ProductID=product1_.ProductID
left outer join
    Supplier supplier2_
        on product1_.supplier_SupplierID=supplier2_.SupplierID
```

where

products0_.Invoice_InvoiceNuber=?

Talerz

Widolec

Stolik

Process finished with exit code 0

Faktycznie zapytania generują wynik zgodny z oczekiwaniami.

ZADANIE 7

W tym poleceniu tworzymy nową klasę Main (u mnie Main2) i powtarzamy funkcjonalność z poprzedniego zadania z wykorzystaniem mechanizmów JPA.

W tym celu stworzyłem nowy plik konfiguracyjny, który umieściłem w podfolderze META_INF folderu resources mojego projektu w intellij.

Plik konfiguracyjny persistence.xml :

```
<?xml version='1.0' encoding='utf-8'?>
<persistence version="2.0" 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">
<persistence-unit name="myDatabaseConfig"
    transaction-type="RESOURCE_LOCAL">
    <properties>
        <property name="hibernate.connection.driver_class"
value="org.apache.derby.jdbc.ClientDriver"/>
        <!-- create=true-->
        <property name="hibernate.connection.url"
value="jdbc:derby://127.0.0.1/JakubBarberJPA"/>
        <property name="hibernate.dialect"
value="org.hibernate.dialect.DerbyTenSevenDialect"/>
        <property name="hibernate.show_sql" value="true"/>
        <property name="hibernate.format_sql" value="format_sql"/>
        <property name="hibernate.use_sql_comments" value="true"/>
        <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
</persistence-unit>
</persistence>
```

Nowa klasa Main2:

```
package org.example;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

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

import java.util.List;
import java.util.Set;

public class Main2 {

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

```

    etx.begin();
    Invoice invoice1 = em.find(Invoice.class, 10);
    Product product = new Product("odkurzacz", 7);
    Supplier supplier = new Supplier("SprzataniePL", "czysta", "Gdansk");
    Invoice invoice = new Invoice(4);
    Category category = new Category("Sprzatanie");
    product.setSupplier(supplier);
    category.addProduct(product);
    product.addToInvoice(invoice);
    em.persist(product);
    em.persist(supplier);
    em.persist(category);
    em.persist(invoice);
    Product product1 = em.find(Product.class, 2);
    System.out.println(product1.getProductName());

    Set<Product> products = invoice1.getProducts();
    System.out.println("Numer faktury :"+invoice.getInvoiceNuber());
    for (Product prod : products){
        System.out.println(prod.getProductName());
    }

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

```

Po wykonaniu program badamy wyniki w Datagripie. Widzimy, że odpowiednie rekordy zostały dodane do bazy, podobnie jak w poprzednim przypadku, a filtrowanie, tym razem przy użyciu metody find na instancji EntityManager.

Wynik filtrowania produktów faktury numer 10 (ucięte długie query wygenerowane przez JPA):

```

Numer faktury :15
Hibernate: select products0_.Invoice_InvoiceNub
Stolik
Talerz
Widelec

```

Wyniki Selecta w DataGripie:

```

1 ✓ SELECT * FROM PRODUCT;
2 ✓ SELECT * FROM SUPPLIER;
3 ✓ SELECT * FROM SUPPLIER_PRODUCT;
4 ✓ SELECT * FROM CATEGORY;
5 ✓ SELECT * FROM INVOICE;

```

1	1 Stolik	112	4	8
2	2 Talerz	11	4	8
3	3 Widelec	22	4	8
4	6 buzz astral	8	5	9
5	7 lalka barbie	3	5	9
6	12 odkurzacz	7	13	14

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Krakow	Chemwit	Gołkowicka
2	5	Krakow	Zabawki	Zabawowa
3	13	Gdansk	SprzataniePL	czysta

	CATEGORYID	NAME
1	8	Dom
2	9	Zabawki
3	14	Sprzatanie

	INVOICENUMBER	QUANTITY
1	10	3
2	11	2
3	15	4

Odpowiednie rekordy zostały dodane, więc zakładam, że wszystko działa jak powinno.

ZADANIE 8

W celu realizacji kaskadowego schematu dodawania produktów do faktur i faktury do produktu do bazy danych, w odpowiednich miejscach umieściłem adnotacje:

Klasa Product:

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

Klasa Invoice:

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

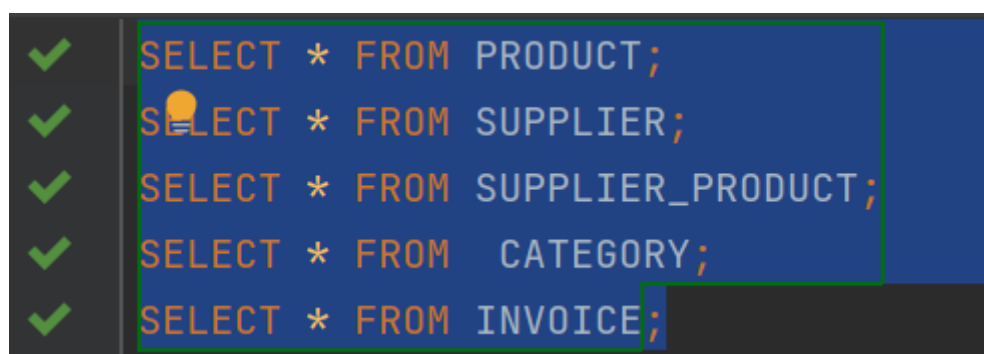
W klasie Main2 utworzyłem pare produktów, dodałem je do nowej faktury, a każdy z nich połączyłem z odpowiednim Supplierem oraz Category istniejącą już w bazie danych. Wykonany kod:

```
public static void main(final String[] args) throws Exception {
    EntityManagerFactory emf =
Persistence.createEntityManagerFactory("myDatabaseConfig");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    //
    Invoice invoice1 = em.find(Invoice.class,10);
    Category category = em.find(Category.class,14);
    Product product1 = new Product("mop",7);
    Product product2 = new Product("szczotka",10);
    Product product3 = new Product("miotła",5);
    Supplier supplier = em.find(Supplier.class,13);
    Invoice invoice = new Invoice(3);
    //
    Category category = new Category("Sprzatanie");
    product1.setSupplier(supplier);
    product2.setSupplier(supplier);
    product3.setSupplier(supplier);
    category.addProduct(product1);
    category.addProduct(product2);
    category.addProduct(product3);
    product1.addToInvoice(invoice);
    product2.addToInvoice(invoice);
    product3.addToInvoice(invoice);
    //
    em.persist(product);
    //
    em.persist(supplier);
    //
    em.persist(category);
    em.persist(invoice);
    //
    Product product1 = em.find(Product.class,2);
    //
    System.out.println(product1.getProductName());
    //
    Set<Product> products = invoice1.getProducts();
    //
    System.out.println("Numer faktury :"+invoice.getInvoiceNuber());
}
```

```
//      for (Product prod : products){
//          System.out.println(prod.getProductName());
//      }

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

Sprawdzamy skuteczność działania kaskadowego dodawania rekordów korzystając z DataGripa:



	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID	CATEGORY_ID
1	1	Stolik	112	4	8
2	2	Talerz	11	4	8
3	3	Widelec	22	4	8
4	6	buzz astral	8	5	9
5	7	lalka barbie	3	5	9
6	12	odkurzacz	7	13	14
7	17	szczotka	10	13	14
8	18	miotla	5	13	14
9	19	mop	7	13	14

	INVOICENUMBER	QUANTITY
1	10	3
2	11	2
3	15	4
4	16	3

Widzimy, że odpowiednie produkty zostały odpowiednio dodane do bazy w sposób kaskadowy (zapisywaliśmy ex plicite jedynie fakturę połączoną z tymi produktami)

ZADANIE 9

W pierwszej części polecenia dodałem do modelu klasę Address, którą wbudowałem jako klasę Embedded do tabeli dostawców (wymagało to lekkiej modyfikacji konstruktora klasy Supplier (klasa Address posiada również dodatkowe pole ZipCode)). Kod poniżej :

Klasa Address:

```
package org.example;

import javax.persistence.Embeddable;

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

    public Address() {
    }

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

    private String zipCode;
}
```

Aktualizujemy klasę Supplier :

```
package org.example;

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

@Entity
public class Supplier {
    public Supplier() {
    }

    public Supplier(String companyName, String street, String city, String zipCode) {
        companyName = companyName;
        adress = new Address(street, city, zipCode);
    }

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

    private String companyName;
}
```



```

private Address address;

@OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)






private Set<Product> productsGroup;

public Set<Product> getProductsGroup() {
    return productsGroup;
}

public void addToProductsGroup(Product product) {
    if (this.productsGroup==null){
        this.productsGroup = new HashSet<Product>();
    }
    this.productsGroup.add(product);
}
}

```

Wykonujemy kod w klasie Main2 dodający jeden produkt z nowej kategorii elektronika do nowo utworzonej faktury i suppliera. Zobaczmy wynik w naszej bazie danych:

	 SUPPLIERID	 CITY	 COMPANYNAME	 STREET	 ZIPCODE
1	4	Krakow	Chemwit	Golkowicka	<null>
2	5	Krakow	Zabawki	Zabawowa	<null>
3	13	Gdansk	SprzataniePL	czysta	<null>
4	22	Krakow	ITpl	Komputerowa	30-555

Widzimy, że do tabeli Suppliers zostało dodane pole ZipCode, ustawione defaultowo we wcześniejszych rekordach na null. Supplier został efektywnie dodany. Wszystko zatem jest zgodne z naszymi przewidywaniami.

W kolejnej części polecenia powracamy częściowo do poprzedniego rozwiązania, ponieważ wracamy do przechowywania danych adresowych w klasie Supplier. Jednak tym razem chcemy przechowywać te dane w osobnej tabeli.

W tym celu modyfikujemy klasę Supplier (dodajemy adnotację @SecondaryTable oraz @Column i wskazania do odpowiednich kolumn w tabeli):

```

package org.example;

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

@Entity
@SecondaryTable(name="ADDRESS_TBL")
public class Supplier {
    public Supplier() {
    }

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

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

    private String companyName;
    @Column(table = "ADDRESS_TBL")
    private String street;
    @Column(table = "ADDRESS_TBL")
    private String city;
    @Column(table = "ADDRESS_TBL")
    private String zipCode;

    // private Adress adress;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    // @JoinColumn(name="supplier_id")
    private Set<Product> productsGroup;

    public Set<Product> getProductsGroup() {
        return productsGroup;
    }

    public void addToProductsGroup(Product product) {
        if (this.productsGroup==null){
            this.productsGroup = new HashSet<Product>();
        }
        this.productsGroup.add(product);
    }
}

```

Wykonujemy kod w klasie Main2:

```

public static void main(final String[] args) throws Exception {
    EntityManagerFactory emf =
Persistence.createEntityManagerFactory("myDatabaseConfig");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
}

```

```

    etx.begin();
    Category category = new Category("Ubrania");
    Product product = new Product("gucci flipflops",2);
    Supplier supplier = new Supplier("gucci", "bogolska", "Krakow",
"30-355");
    Invoice invoice = new Invoice(1);
    product.setSupplier(supplier);
    category.addProduct(product);
    product.addToInvoice(invoice);

    em.persist(supplier);
    em.persist(category);
    em.persist(invoice);
//

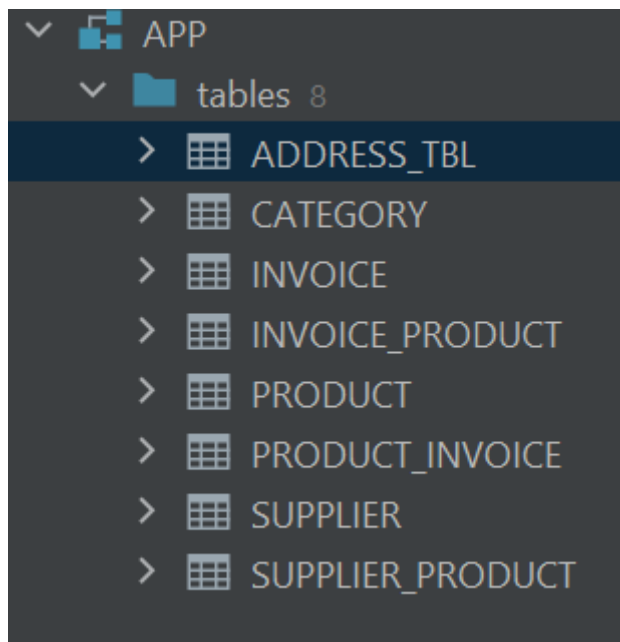
    etx.commit();
    em.close();

}

```

Wynik w DataGripie:

Faktycznie została utworzona tabela pomocnicza ADDRESS_TBL, zawartość tabeli zgodna z oczekiwaniami:



	CITY	STREET	ZIPCODE	SUPPLIERID
1	Krakow	bogolska	30-355	26

ZADANIE 10

W tej części chcemy wprowadzić dziedziczenie w naszym modelu przy zastosowaniu 3 różnych strategii mapowanie dziedziczenia:

W tym celu stworzyłem klasę Company, która następnie będę rozszerzał klasami Supplier oraz Customer:

Klasa Company:

```
package org.example;

import javax.persistence.*;

@Entity
@SecondaryTable(name="ADDRESS_TBL")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company {
    public Company() {
    }

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

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

    public String getCompanyName() {
        return companyName;
    }

    public void setCompanyName(String companyName) {
        companyName = companyName;
    }

    protected String companyName;
    @Column(table = "ADDRESS_TBL")
    protected String street;
    @Column(table = "ADDRESS_TBL")
    protected String city;
    @Column(table = "ADDRESS_TBL")
    protected String zipCode;
}
```

Klasa Supplier:

```
package org.example;

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

@Entity
public class Supplier extends Company {
    public Supplier() {
    }

    private String bankAccountNumber;

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

    // private Address adress;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    // @JoinColumn(name="supplier_id")
    private Set<Product> productsGroup;

    public Set<Product> getProductsGroup() {
        return productsGroup;
    }

    public void addToProductsGroup(Product product) {
        if (this.productsGroup==null){
            this.productsGroup = new HashSet<Product>();
        }
        this.productsGroup.add(product);
    }
}
```

Klasa Customer:

```
package org.example;

import javax.persistence.Entity;

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

    public Customer(){
    }

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

```
}
}
```

1. Jedna tabela na całą hierarchię

W tym celu w klasie z której dziedziczymy stosujemy adnotację:

```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
```

W klasie Main2 tworzymy instancje Customera i Supliera i zapisujemy w bazie w każdym przypadku w tym zadaniu. Zobaczmy wyniki w bazie danych:

DTYPE	SUPPLIERID	COMPANYNAME	BANKACCOUNTNUMBER	DISCOUNT
1 Supplier	1	CompanyOneTab	444 444 444	<null>
2 Customer	2	CustomerOneTab	<null>	0.4

Widzimy, że została utworzona tabela Company, w której wprzechowywane są redundantne pola dla wszystkich klas rozszerzających tą klasę. Wynik zgadza się z przewidywaniami.

2. Tabele łączone

Tym razem stosujemy adnotacje :

```
@Inheritance(strategy = InheritanceType.JOINED)
```

Wykonujemy kod w klasie Main2 i analizujemy wyniki:

Wyniki w DataGripie:

	BANKACCOUNTNUMBER	SUPPLIERID
1	444 444 444	1

	SUPPLIERID	COMPANYNAME
1	1	CompanyOneTab
2	2	CustomerOneTab

	DISCOUNT	SUPPLIERID
1	0.4	2

	BANKACCOUNTNUMBER	SUPPLIERID
1	444 444 444	1

	CITY	STREET	ZIPCODE	SUPPLIERID
1	Krakow	bogońska	30-355	1
2	Krakow	bogońska	30-355	2

Wyniki są zgodne z oczekiwaniami.

2.Table per Class

Na koniec stosujemy adnotacje :

```
@Inheritance(strategy = InheritanceType.JOINED)
```

Z klasy Company usunąłem strategię generowanie osobnej tabeli na adres.

Wykonujemy kod w klasie Main2 i analizujemy wyniki:

Wyniki w DataGripie:

	SUPPLIERID	COMPANYNAME	CITY	STREET	ZIPCODE
--	------------	-------------	------	--------	---------

	SUPPLIERID	COMPANYNAME	CITY	STREET	ZIPCODE	DISCOUNT
1	2	CustomerOneTab	Krakow	bogońska	30-355	0.4

	SUPPLIERID	COMPANYNAME	CITY	STREET	ZIPCODE	BANKACCOUNTNUMBER
1	1	CompanyOneTab	Krakow	bogońska	30-355	444 444 444

Wyniki ponownie są zgodne z oczekiwaniami.