

# 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.

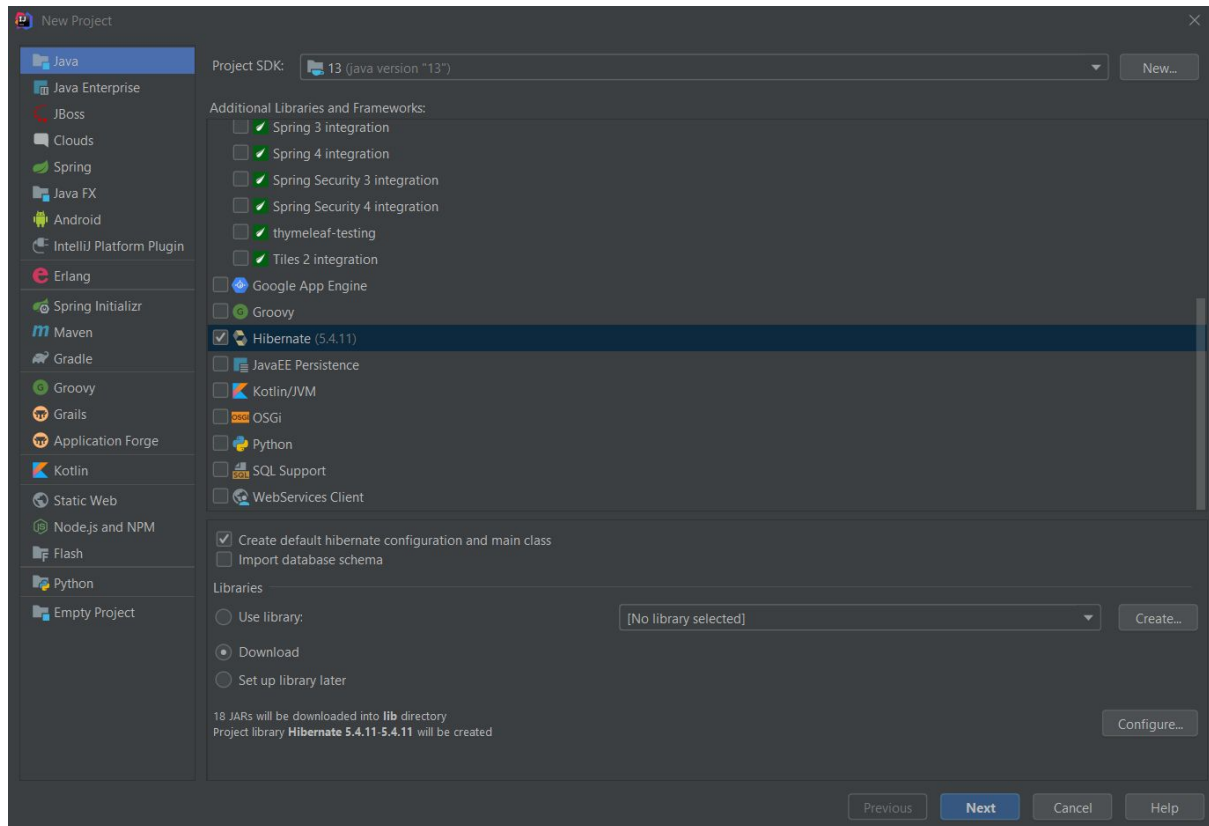
```
C:\WINDOWS\system32\cmd.exe

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       |

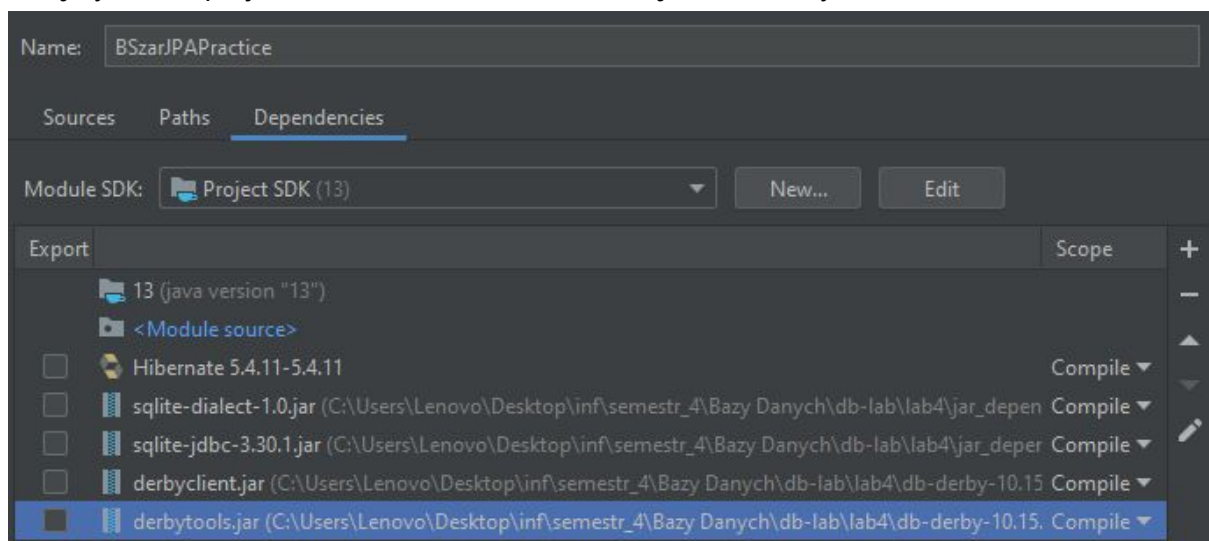
23 wierszy wybranych
ij>
```

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.

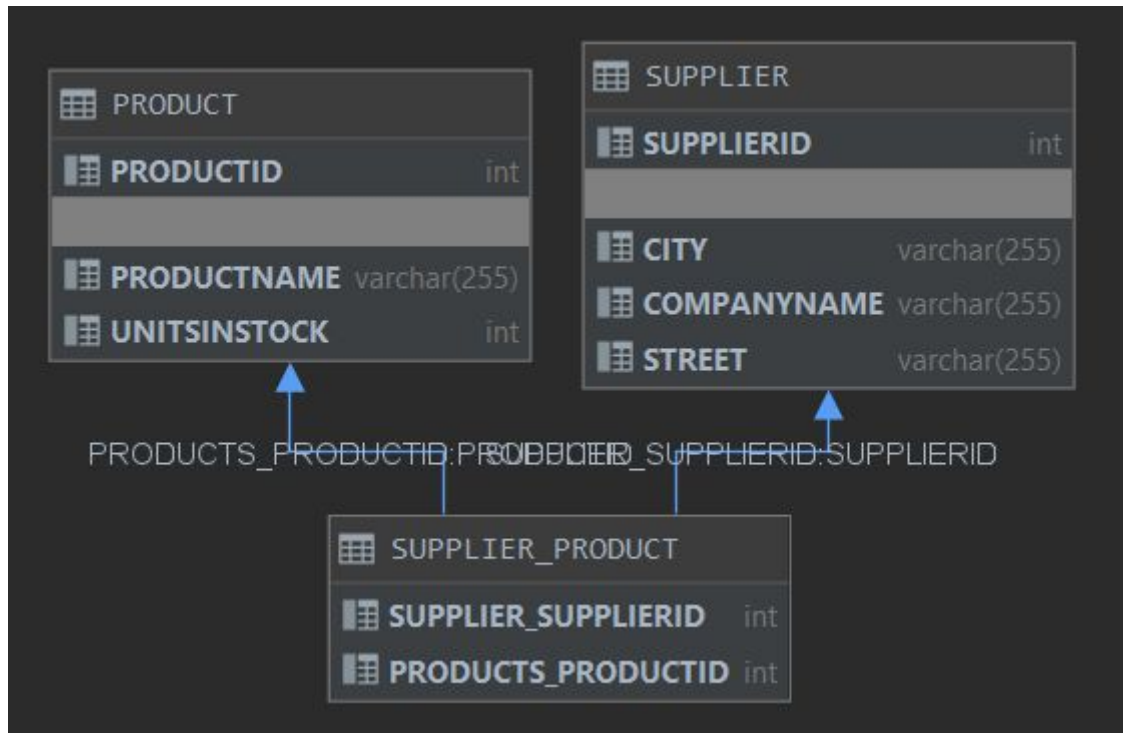
```
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int SupplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    private Set<Product> products = new LinkedHashSet<>();

    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.

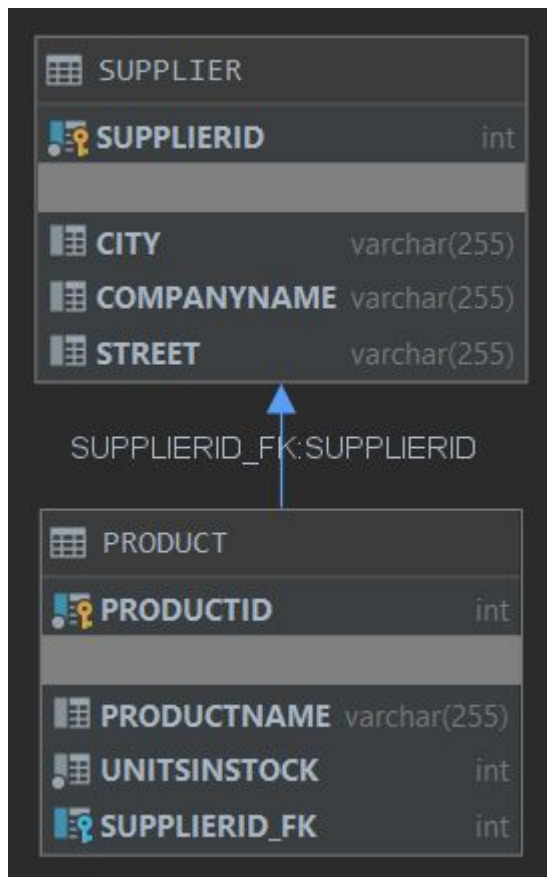


wersja bez tabeli łącznikowej:

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)
```