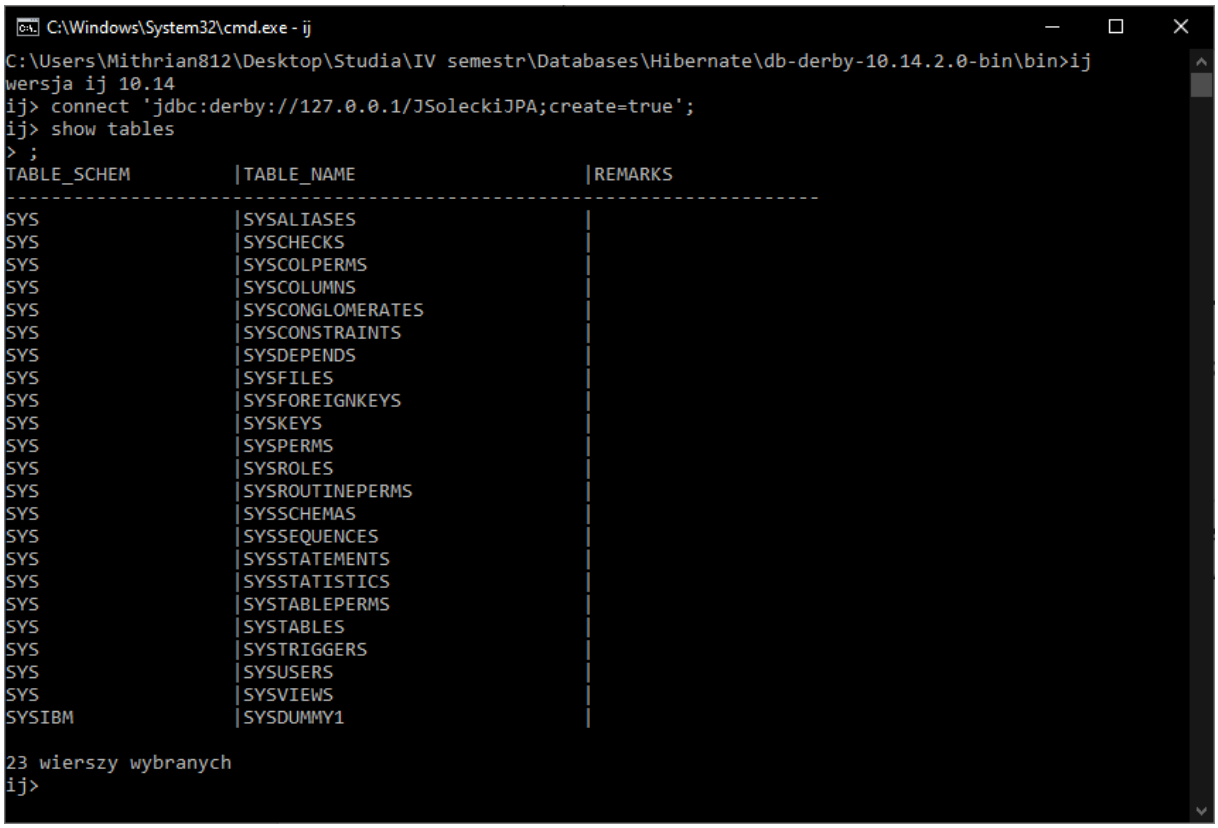
Laboratorium 4 – Hibernate

Jakub Solecki, Wtorek A 11:15

Pierwszym krokiem było skonfigurowanie i serwera Derby. Ponieważ posiadam starszą wersję Javy a nie miałem czasu jej zaktualizować więc skorzystałem z wersji DBMS Derby 10.14.2.0. Niewykluczone, że po zajęciach będę kontynuował zadania na najnowszej wersji.
Serwer oraz baza zostały założone pomyślnie:



Następnie utworzyłem klasę Product:

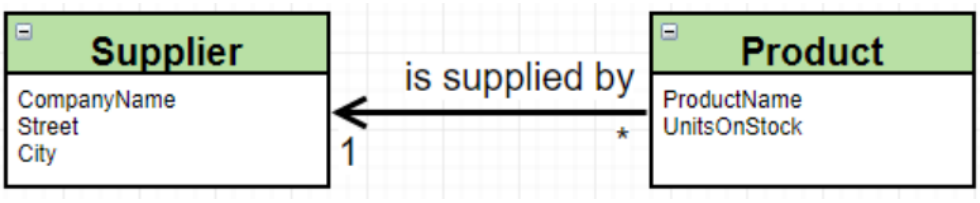
```java
@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int productID;
   private String ProductName;
   private int unitsInStock;

   public Product() {
   }

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

Oraz stworzyłem nowy produkt I zapisałem go w bazie:

```
Hibernate:

values
    next value for hibernate_sequence
Hibernate:
    /* insert Product
        */ insert
        into
            Product
            (ProductName, unitsInStock, productID)
        values
            (?, ?, ?)
querying all the managed entities...
executing: from Product
Hibernate:
    /*
from
    Product */ select
        product0_.productID as producti1_0_,
        product0_.ProductName as productn2_0_,
        product0_.unitsInStock as unitsins3_0_
    from
        Product product0_
  Product@47406941

Process finished with exit code 0
```

Punkt IV

Utworzyłem klasę Supplier, a w klasie Product dodałem pole typu Supplier z adnotacją @ManyToOne w celu stworzenia relacji:



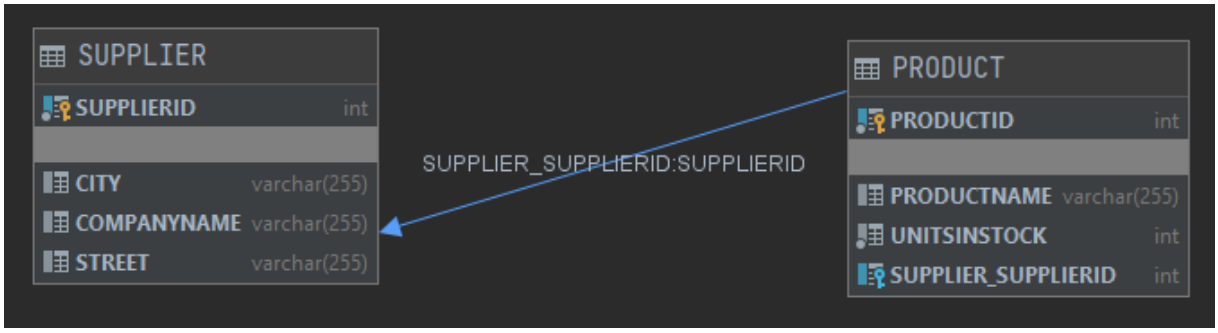

```
@Entity
public class Supplier {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int supplierID;
   private String CompanyName;
   private String Street;
   private String City;

   public Supplier() {
   }

   public Supplier(String companyName, String street, String city) {
      CompanyName = companyName;
      Street = street;
```

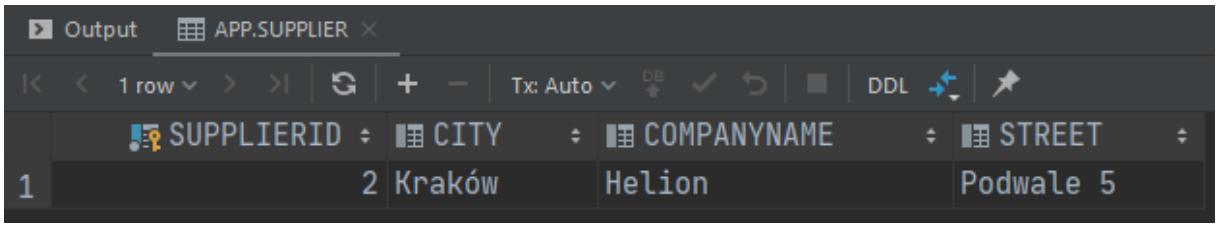
```
    City = city;
  }
}
```

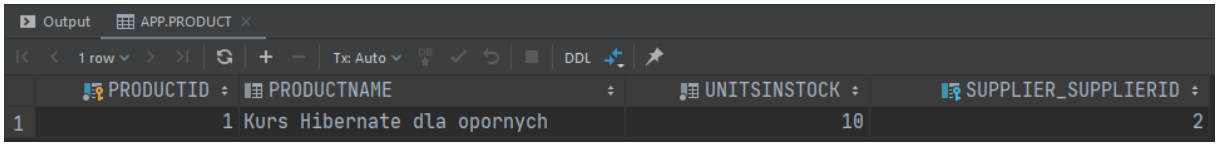
Następnie znalazłem poprzedni produkt i dodałem do niego nowo utworzonego dostawcę:

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    try {
      //      Product product1 = new Product("Kurs Hibernate dla opornych", 10);
      Supplier supplier = new Supplier("Helion", "Podwale 5", "Kraków");
      session.save(supplier);
      Product foundStudent = session.get(Product.class,1);
      foundStudent.setSupplier(supplier);
      session.save(foundStudent);

      Query query = session.createQuery("from  Product");
      System.out.println(query.list());
    } finally {
      tx.commit();
      session.close();
    }
  }
```

Wyniki są widoczne w bazie:

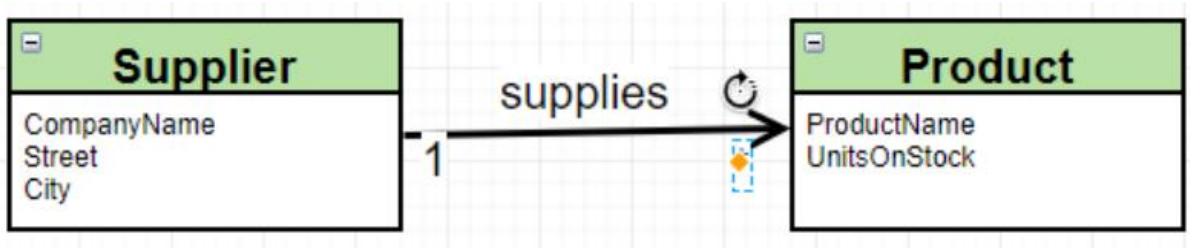






Punkt V

Odwróciłem relację zgodnie z poniższym schematem:



Usunąłem pole Supplier z klasy Product, natomiast w klasie Supplier dodałem pole typu Set<Product> wraz z adnotacją @OneToMany.

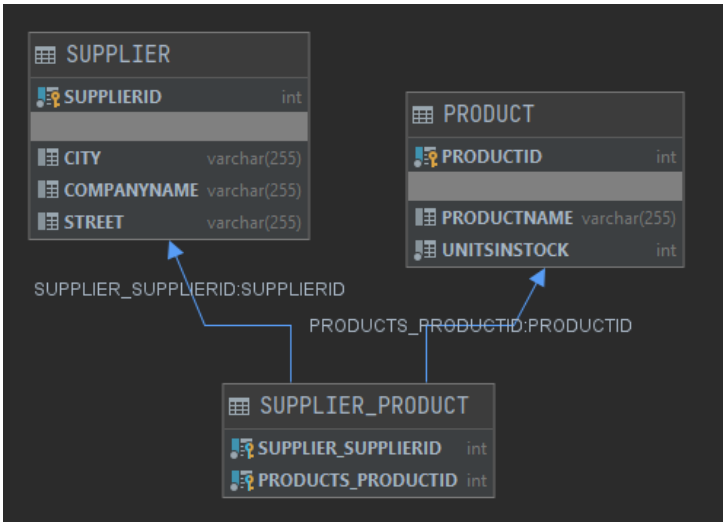
```
@Entity
public class Supplier {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int supplierID;
   private String CompanyName;
   private String Street;
   private String City;
   @OneToMany
   private Set<Product> Products;

   public Supplier() {
   }

   public Supplier(String companyName, String street, String city) {
      CompanyName = companyName;
      Street = street;
      City = city;
      Products = new HashSet<>();
   }

   public void setProducts(Product product) {
      Products.add(product);
   }
}
```

Następnie utworzyłem kilka produktów i dodałem je do dostawcy:

```
public static void main(final String[] args) throws Exception {
   final Session session = getSession();
   Transaction tx = session.beginTransaction();
   try {
      Product product1 = new Product("Gotuj z TurboPascalem", 20);
      Product product2 = new Product("Biblia C++", 20);
      Product product3 = new Product("Hibernate dla opornych", 20);
      Supplier supplier = new Supplier("Helion", "Biliotekarska 3", "Kraków");
      session.save(product1);
      session.save(product2);
      session.save(product3);
      supplier.setProducts(product1);
      supplier.setProducts(product2);
      supplier.setProducts(product3);
```

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

Spowodowało to dodanie tabeli łącznikowej w schemacie.



Natomiast w bazie powstało poprawne połącznie Dostawcy z Produktami:

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Helion	Biliotekarska 3

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK
1	1	Gotuj z TurboPascalem	20
2	2	Biblia C++	20
3	3	Hibernate dla opornych	20

	SUPPLIER_SUPPLIERID	PRODUCTS_PRODUCTID
1	4	1
2	4	2
3	4	3

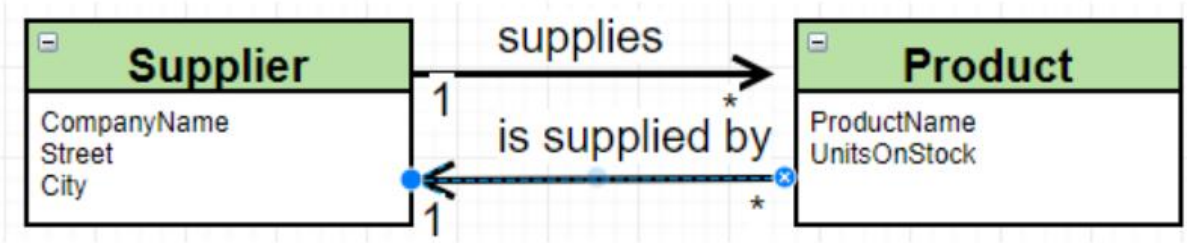
Z kolei zastosowanie adnotacji @JoinColumn pod @OneToMany dla zbioru produktów w klasie Supplier dało następujące rezultaty:

```java
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    private String CompanyName;
    private String Street;
    private String City;
    @OneToMany
    @JoinColumn
    private Set<Product> Products;

    public Supplier() {
    }

    public Supplier(String companyName, String street, String city) {
        CompanyName = companyName;
        Street = street;
        City = city;
        Products = new HashSet<>();
    }

    public void setProducts(Product product) {
        Products.add(product);
    }
}
```

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Helion	Biliotekarska 3

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	PRODUCTS_SUPPLIERID
1	1	Gotuj z TurboPascalem	20	4
2	2	Biblia C++	20	4
3	3	Hibernate dla opornych	20	4

Punkt VI



Na najpierw zastosowałem domyślne podejście z użyciem tabeli łącznikowej:

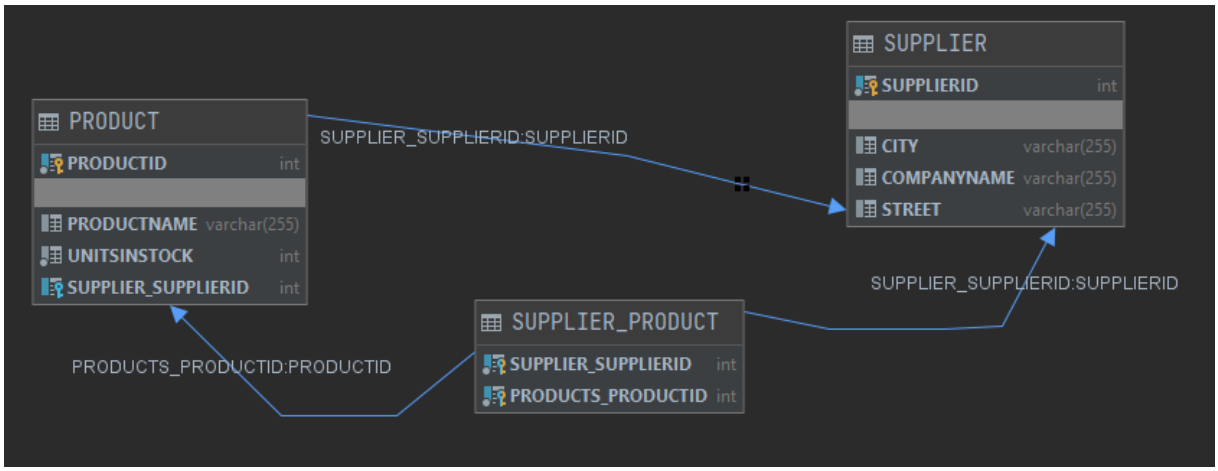
```
@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int productID;
   private String productName;
   private int unitsInStock;
   @ManyToOne
   private Supplier supplier;

   public Product() {
   }

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

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

```
@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;

   public Supplier() {
   }

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

   public void setProducts(Product product) {
      Products.add(product);
```

```
    }
}
```

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    try {
        Product product1 = new Product("Gotuj z TurboPascalem", 20);
        Product product2 = new Product("Biblia C++", 20);
        Product product3 = new Product("Hibernate dla opornych", 20);
        Supplier supplier = new Supplier("Helion", "Biliotekarska 3", "Kraków");
        session.save(product1);
        session.save(product2);
        session.save(product3);
        supplier.setProducts(product1);
        supplier.setProducts(product2);
        supplier.setProducts(product3);
        product1.setSupplier(supplier);
        product2.setSupplier(supplier);
        product3.setSupplier(supplier);
        session.save(supplier);
    } finally {
        tx.commit();
        session.close();
    }
}
```

Rezultat jest zgodny z oczekiwaniami:



	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Kraków	Helion	Biliotekarska 3

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	PRODUCTS_SUPPLIERID
1	1	Gotuj z TurboPascalem	20	4
2	2	Biblia C++	20	4
3	3	Hibernate dla opornych	20	4

Następnie zamodelowałem powyższe relacje z pominięciem tabeli łącznikowej:

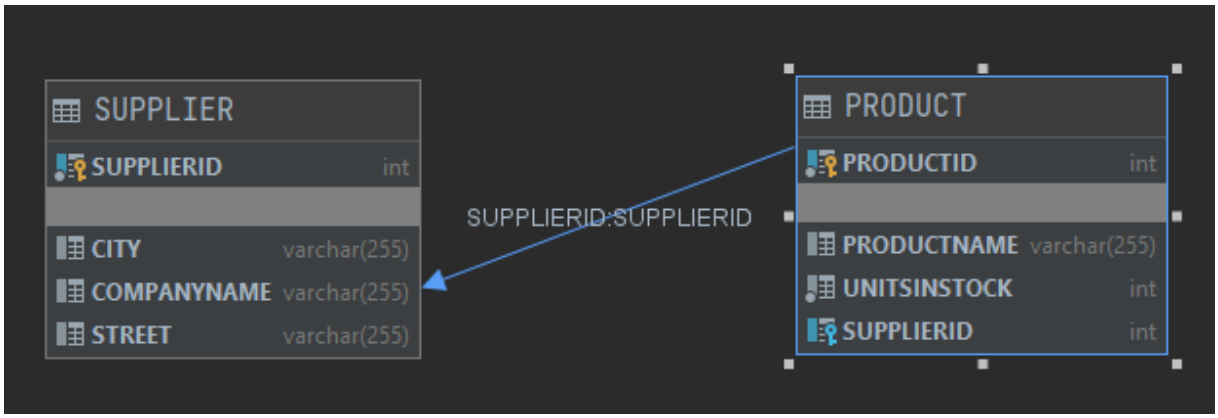
```
@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int productID;
   private String productName;
   private int unitsInStock;
   @ManyToOne
   @JoinColumn(name = "supplierID")
   private Supplier supplier;

   public Product() {
   }

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

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

Ważne było zadbanie o dwustronną obsługę relacji (metoda addProduct jednocześnie dodającą aktualnego dostawce w produkcie):

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

   public Supplier() {
   }

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

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

```java
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    try {
        Product product1 = new Product("Gotuj z TurboPascalem", 20);
        Product product2 = new Product("Biblia C++", 20);
        Product product3 = new Product("Hibernate dla opornych", 20);
        Supplier supplier = new Supplier("Helion", "Biliotekarska 3", "Kraków");
        supplier.addProduct(product1);
        supplier.addProduct(product2);
        supplier.addProduct(product3);
        session.save(supplier);
        session.save(product1);
        session.save(product2);
        session.save(product3);
    } finally {
        tx.commit();
        session.close();
    }
}
```

A oto wyniki:



	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	Helion	Biliotekarska 3

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIERID
1	2	Gotuj z TurboPascalem	20	1
2	3	Biblia C++	20	1
3	4	Hibernate dla opornych	20	1

Zadanie Domowe

Punkt VII

Stworzyłem klasę Categories oraz wprowadziłem odpowiednie modyfikacje w klasie Product:

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


   public Category() {
   }


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


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


   public String getCategoryName() {
      return categoryName;
   }
}
```

```
@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int productID;
   private String productName;
   private int unitsInStock;
   @ManyToOne
   private Supplier supplier;
   @ManyToOne
   private Category category;
```

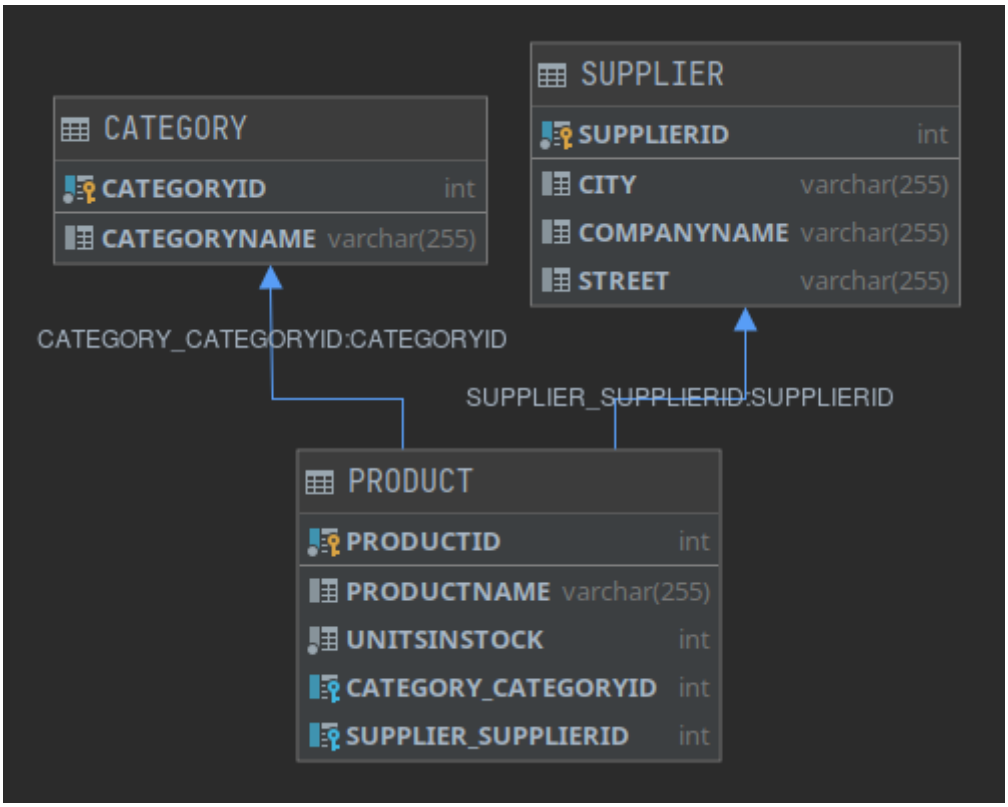
```
    public Product() {
    }


    public Product(String productName, int unitsInStock) {
        this.productName = productName;
        this.unitsInStock = unitsInStock;
    }
    public void setSupplier(Supplier supplier) {
        this.supplier = supplier;
    }


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


    public Category getCategory() {
        return category;
    }
}
```

Następnie stworzyłem kilka produktów, kategorii oraz dostawców a następnie odpowiednio ich połączyłem:

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    try {
        Product pascal = new Product("Gotuj z TurboPascalem", 20);
        Product bible = new Product("Biblia C++", 20);
        Product cyberpunk = new Product("Cyberpunk 2077", 0);
        Product witcher = new Product("Wiedźmin 3: Dziki Zgon", 100);
        Supplier helion = new Supplier("Helion", "Bibliotekarska 3", "Kraków");
        Supplier cdProj = new Supplier("CD Projekt Red", "Jagiellońska 74", "Warszawa");
        Category books = new Category("Książki");
        Category games = new Category("Gry");
        pascal.setCategory(books);
        bible.setCategory(books);
        witcher.setCategory(games);
        cyberpunk.setCategory(games);
        helion.addProduct(pascal);
        helion.addProduct(bible);
        cdProj.addProduct(cyberpunk);
        cdProj.addProduct(witcher);


        session.save(pascal);
        session.save(bible);
        session.save(cyberpunk);
        session.save(witcher);
        session.save(helion);
```

```
        session.save(cdProj);
        session.save(books);
        session.save(games);


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

Wszystkie dane poprawnie zostały poprawnie zmapowane do bazy:



	CATEGORYID	CATEGORYNAME
1	7	Książki
2	8	Gry

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_CATEGORYID	SUPPLIER_SUPPLIERID
1	1	Gotuj z TurboPascalem	20	7	5
2	2	Biblia C++	20	7	5
3	3	Cyberpunk 2077	0	8	6
4	4	Wiedźmin 3: Dziki Zgon	100	8	6

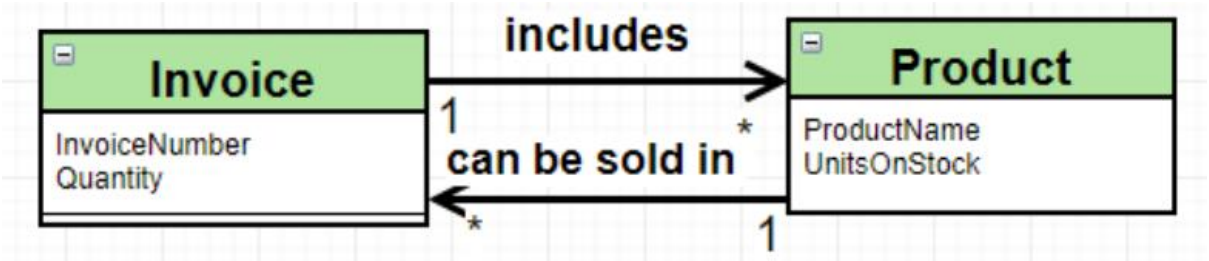
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	5	Kraków	Helion	Bibliotekarska 3
2	6	Warszawa	CD Projekt Red	Jagiellońska 74

Następnie wydobyłem z bazy produkty z kategorii Książki oraz kategorię produktu Gotuj z TurboPascalem:

```
public static void main(final String[] args) throws Exception {
      final Session session = getSession();
      Transaction tx = session.beginTransaction();
      try {
         String hql1 = "from Product as P where P.category.categoryName like 'Książki'";
         Query productQuery = session.createQuery(hql1);
         String product = session.get(Product.class, 1).getCategory().getCategoryName();
         String hql2 = "from Category as C where C.categoryName like :product";
         Query categoryQuery = session.createQuery(hql2).setParameter("product",
product);


         System.out.println("Products with category 'Książki':");
         for(Object p : productQuery.getResultList()) {
            Product prod = (Product)p;
            System.out.println(prod.getProductName() + " : " +
prod.getCategory().getCategoryName());
         }


         System.out.println("Category of Gotuj z TurboPascalem:");
         for(Object o : categoryQuery.getResultList()) {
            Category cat = (Category)o;
            System.out.println(cat.getCategoryName());
         }


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

```
/home/jakubs/.jdks/openjdk-14.0.1/bin/java ...
kwi 30, 2020 6:29:14 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate Core {5.4.11.Final}
kwi 30, 2020 6:29:15 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
INFO: HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
kwi 30, 2020 6:29:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
kwi 30, 2020 6:29:15 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/JSoleckiJPA]
kwi 30, 2020 6:29:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {}
kwi 30, 2020 6:29:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
kwi 30, 2020 6:29:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
kwi 30, 2020 6:29:15 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.DerbyTenSevenDialect
kwi 30, 2020 6:29:15 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnect
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$Co
kwi 30, 2020 6:29:15 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Products with category 'Książki':
Gotuj z TurboPascalem : Książki
Biblia C++ : Książki
Category of Gotuj z TurboPascalem:
Książki
```

Punkt VIII



Stworzyłem klasę Invoice oraz zmodyfikowałem odpowiednio klasę Product:

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


   public Invoice() {
   }


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


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

```
@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int productID;
   private String productName;
   private int unitsInStock;
   @ManyToOne
   private Supplier supplier;
   @ManyToOne
```

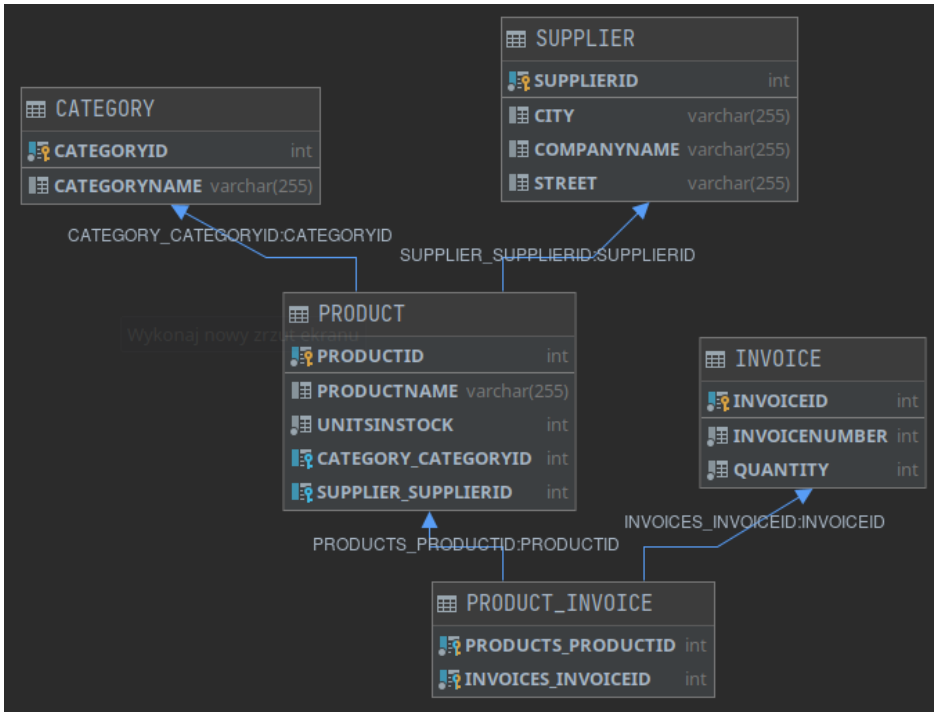
```
    private Category category;
    @ManyToMany
    private Set<Invoice> invoices;


    public Product() {
    }


    public Product(String productName, int unitsInStock) {
        this.productName = productName;
        this.unitsInStock = unitsInStock;
        this.invoices = new HashSet<>();
    }


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


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


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

Następnie stworzyłem kilka produktów i sprzedałem je w 2 transakcjach (2 faktury):

```
public static void main(final String[] args) throws Exception {
        final Session session = getSession();
        Transaction tx = session.beginTransaction();
        try {
            Product pascal = new Product("Gotuj z TurboPascalem", 20);
            Product bible = new Product("Biblia C++", 20);
            Product cyberpunk = new Product("Cyberpunk 2077", 0);
            Product witcher = new Product("Wiedźmin 3: Dziki Zgon", 100);
            Supplier helion = new Supplier("Helion", "Bibliotekarska 3", "Kraków");
            Supplier cdProj = new Supplier("CD Projekt Red", "Jagiellońska 74", "Warszawa");
            Category books = new Category("Książki");
            Category games = new Category("Gry");
            Invoice invoice1 = new Invoice(1, 3);
            Invoice invoice2 = new Invoice(2, 2);
            invoice1.addProduct(pascal);
            invoice1.addProduct(bible);
            invoice1.addProduct(cyberpunk);
            invoice2.addProduct(witcher);
            invoice2.addProduct(cyberpunk);
```

```
        pascal.setCategory(books);
        bible.setCategory(books);
        witcher.setCategory(games);
        cyberpunk.setCategory(games);
        helion.addProduct(pascal);
        helion.addProduct(bible);
        cdProj.addProduct(cyberpunk);
        cdProj.addProduct(witcher);


        session.save(pascal);
        session.save(bible);
        session.save(cyberpunk);
        session.save(witcher);
        session.save(helion);
        session.save(cdProj);
        session.save(books);
        session.save(games);
        session.save(invoice1);
        session.save(invoice2);
    } finally {
        tx.commit();
        session.close();
    }
}
```

Wszystkie dane oraz powiązania między nimi zostały poprawnie zmapowane do bazy:



	CATEGORYID	CATEGORYNAME
1	7	Książki
2	8	Gry

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_CATEGORYID	SUPPLIER_SUPPLIERID
1	1	Gotuj z TurboPascalem	20	7	5
2	2	Biblia C++	20	7	5
3	3	Cyberpunk 2077	0	8	6
4	4	Wiedźmin 3: Dziki Zgon	100	8	6

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	5	Kraków	Helion	Bibliotekarska 3
2	6	Warszawa	CD Projekt Red	Jagiellońska 74

	INVOICEID	INVOICENUMBER	QUANTITY
1	9	1	3
2	10	2	2

	PRODUCTS_PRODUCTID	INVOICES_INVOICEID
1	1	9
2	2	9
3	3	9
4	3	10
5	4	10

Następnie wyciągnąłem z bazy produkty znajdujące się na fakturze o ID 9 oraz fakturę, na której znajduje się produkt o ID 3:

```
public static void main(final String[] args) throws Exception {
        final Session session = getSession();
        Transaction tx = session.beginTransaction();
        try {
            Invoice invoice = session.get(Invoice.class, 9);
            Query productQuery = session.createQuery("select P from Product P " +
                    "join P.invoices i where i.invoiceID = :invID").setParameter("invID",
invoice.getInvoiceID());
            Product product = session.get(Product.class, 3);
            Query invoiceQuery = session.createQuery("select I from Invoice I " +
                    "join I.products p where p.productID = :prodID").setParameter("prodID",
product.getProductID());


            System.out.println("Products with Invoice " + invoice.getInvoiceNumber() + ":");
```

```
        for(Object o : productQuery.getResultList()) {
            Product prod = (Product) o;
            System.out.println(prod.getProductName());
        }


        System.out.println("\nInvoices for product " + product.getProductName() + ":");
        for(Object o : invoiceQuery.getResultList()) {
            Invoice inv = (Invoice) o;
            System.out.println("Invoice number: " + inv.getInvoiceNumber());
        }


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

```
kwi 30, 2020 9:13:10 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate Core {5.4.11.Final}
kwi 30, 2020 9:13:10 PM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit>
INFO: HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
kwi 30, 2020 9:13:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
kwi 30, 2020 9:13:10 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/JSoleckiJPA]
kwi 30, 2020 9:13:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {}
kwi 30, 2020 9:13:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
kwi 30, 2020 9:13:10 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
kwi 30, 2020 9:13:10 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.DerbyTenSevenDialect
kwi 30, 2020 9:13:11 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnec
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$C
kwi 30, 2020 9:13:11 PM org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Products with Invoice 1:
Gotuj z TurboPascalem
Biblia C++
Cyberpunk 2077

Invoices for product Cyberpunk 2077:
Invoice number: 1
Invoice number: 2

Process finished with exit code 0
```

Punkt X

Na początku stworzyłem folder META_INF wraz z plikiem persistence.xml, zawierającym dane konfiguracyjne JPA:

```
<?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/JSoleckiJPA"/>
      <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>
```

Następnie przywróciłem klasy Product oraz Supplier do stanu z punktu VI:

```
@Entity
public class Product {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private int productID;
  private String productName;
  private int unitsInStock;
  @ManyToOne
  private Supplier supplier;


  public Product() {
  }


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


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

```
    }

    public int getProductID() {
        return productID;
    }

    public String getProductName() {
        return productName;
    }

    public int getUnitsInStock() {
        return unitsInStock;
    }

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

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

    public Supplier() {
    }

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

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

    public int getSupplierID() {
        return supplierID;
```

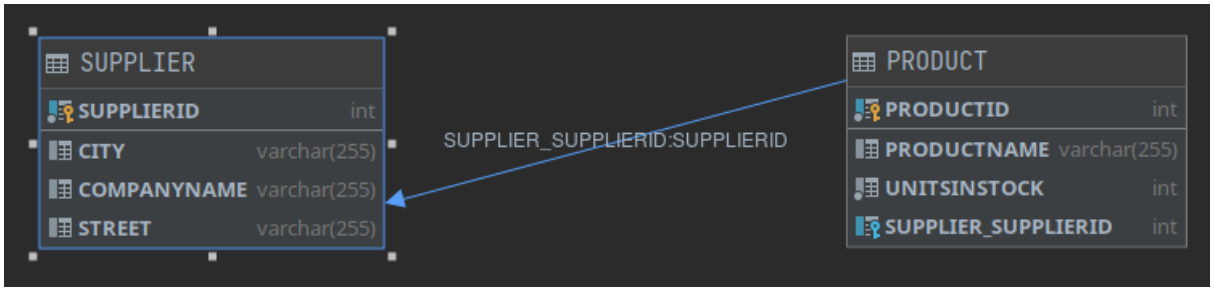
```
    }

    public String getCompanyName() {
        return companyName;
    }

    public String getStreet() {
        return street;
    }

    public String getCity() {
        return city;
    }

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

Na końcu stworzyłem klasę Main wraz z metodą main, w której wykonuję operacje tworzenia oraz zapisywania obiektów:

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


        Product pascal = new Product("Gotuj z TurboPascalem", 20);
        Product bible = new Product("Biblia C++", 20);
        Product cyberpunk = new Product("Cyberpunk 2077", 0);
        Product witcher = new Product("Wiedźmin 3: Dziki Zgon", 100);
        Supplier helion = new Supplier("Helion", "Bibliotekarska 3", "Kraków");
        Supplier cdProj = new Supplier("CD Projekt Red", "Jagiellońska 74", "Warszawa");


        helion.addProduct(bible);
        helion.addProduct(pascal);
        cdProj.addProduct(witcher);
        cdProj.addProduct(cyberpunk);


        em.persist(helion);
        em.persist(cdProj);
        em.persist(pascal);
        em.persist(bible);
```

```
        em.persist(cyberpunk);
        em.persist(witcher);


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

Po wykonaniu powyższych kroków wszystkie dane zostały poprawnie zmapowane do bazy:



	SUPPLIERID	CITY	COMPANYNAME	STREET
1	1	Kraków	Helion	Bibliotekarska 3
2	2	Warszawa	CD Projekt Red	Jagiellońska 74

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	SUPPLIER_SUPPLIERID
1	3	Gotuj z TurboPascalem	20	1
2	4	Biblia C++	20	1
3	5	Cyberpunk 2077	0	2
4	6	Wiedźmin 3: Dziki Zgon	100	2

Punkt XI

W tym punkcie wykorzystałem kaskadowe operacje do tworzenia faktur wraz z produktami oraz na odwrót. W tym celu do wszystkich adnotacji @OneToMany oraz @ManyToOne dodałem opcję cascade = {CascadeType.PERSIST}:

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int productID;
    private String productName;
    private int unitsInStock;
    @ManyToOne(cascade = {CascadeType.PERSIST})
    private Supplier supplier;
    @ManyToOne(cascade = {CascadeType.PERSIST})
    private Category category;
    @ManyToMany(cascade = {CascadeType.PERSIST})
    private Set<Invoice> invoices;


    public Product() {
    }


    public Product(String productName, int unitsInStock) {
        this.productName = productName;
        this.unitsInStock = unitsInStock;
        this.invoices = new HashSet<>();
    }


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


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


    public int getProductID() {
        return productID;
    }


    public String getProductName() {
        return productName;
    }


    public int getUnitsInStock() {
```

```
        return unitsInStock;
    }


    public Supplier getSupplier() {
        return supplier;
    }


    public Category getCategory() {
        return category;
    }


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


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

```
@Entity
public class Invoice {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int invoiceID;
    private int invoiceNumber;
    private int quantity;
    @ManyToMany(mappedBy = "invoices", cascade = {CascadeType.PERSIST})
    private Set<Product> products;


    public Invoice() {
    }


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


    public int getInvoiceID() {
        return invoiceID;
    }


    public int getInvoiceNumber() {
        return invoiceNumber;
    }
```

```
    public int getQuantity() {
        return quantity;
    }


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


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

```
@Entity
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int categoryID;
    private String categoryName;
    @OneToMany(mappedBy = "category", cascade = {CascadeType.PERSIST})
    private List<Product> products;


    public Category() {
    }


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


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


    public int getCategoryID() {
        return categoryID;
    }


    public String getCategoryName() {
        return categoryName;
    }


    public List<Product> getProducts() {
```

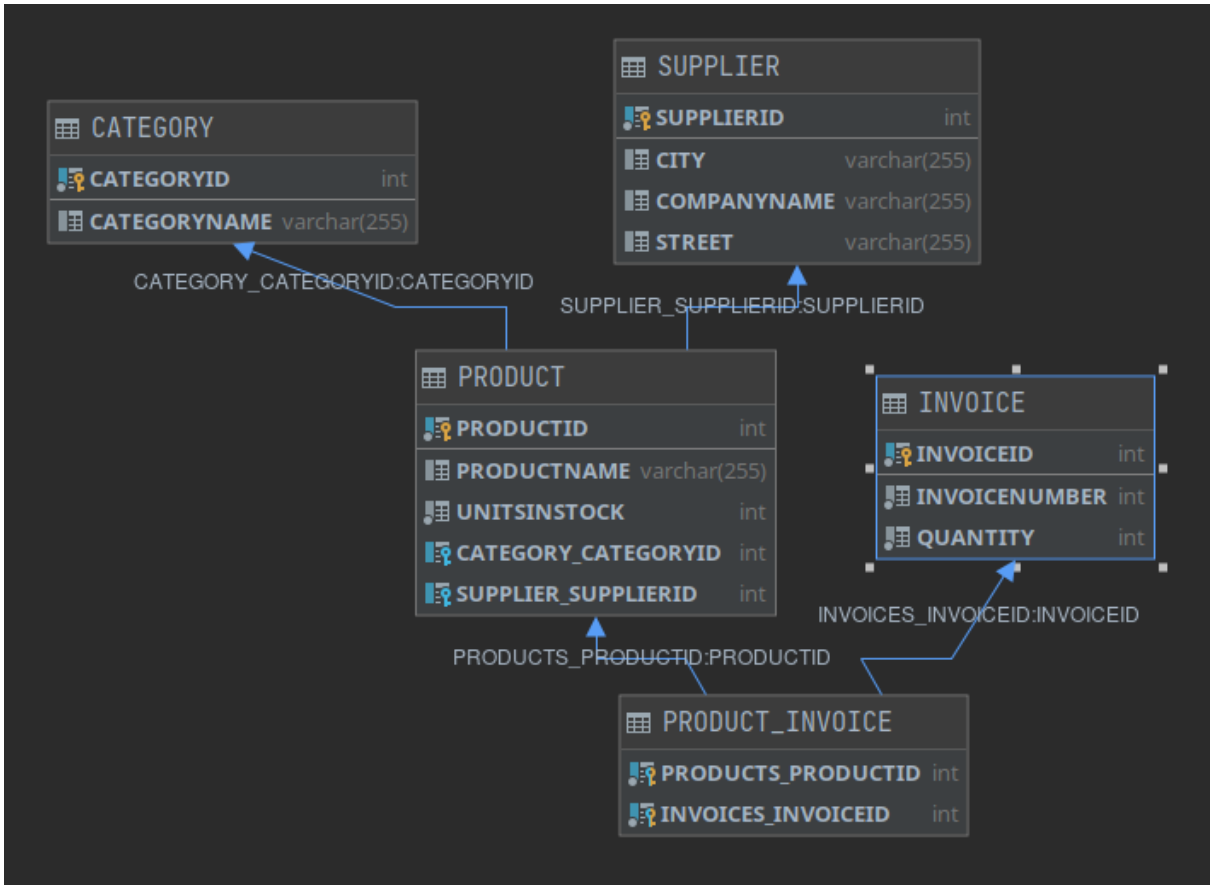
```
        return products;
    }
}
```

```
@Entity
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    private String companyName;
    private String street;
    private String city;
    @OneToMany(mappedBy = "supplier", cascade = {CascadeType.PERSIST})
    private Set<Product> products;


    public Supplier() {
    }


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


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


    public int getSupplierID() {
        return supplierID;
    }


    public String getCompanyName() {
        return companyName;
    }


    public String getStreet() {
        return street;
    }


    public String getCity() {
        return city;
    }
```

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

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


        Product pascal = new Product("Gotuj z TurboPascalem", 20);
        Product bible = new Product("Biblia C++", 20);
        Product cyberpunk = new Product("Cyberpunk 2077", 0);
        Product witcher = new Product("Wiedźmin 3: Dziki Zgon", 100);
        Supplier helion = new Supplier("Helion", "Bibliotekarska 3", "Kraków");
        Supplier cdProj = new Supplier("CD Projekt Red", "Jagiellońska 74", "Warszawa");
        Category books = new Category("Książki");
        Category games = new Category("Gry");
        Invoice invoice1 = new Invoice(1, 3);
        Invoice invoice2 = new Invoice(2, 2);
        pascal.setCategory(books);
        bible.setCategory(books);
        witcher.setCategory(games);
        cyberpunk.setCategory(games);
        invoice1.addProduct(pascal);
        invoice1.addProduct(bible);
        invoice1.addProduct(cyberpunk);
        invoice2.addProduct(witcher);
        invoice2.addProduct(cyberpunk);
        helion.addProduct(bible);
        helion.addProduct(pascal);
        cdProj.addProduct(witcher);
        cdProj.addProduct(cyberpunk);


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


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

Zgodnie z oczekiwaniami zapisanie “wprost” samych faktur kaskadowo zapisało również wszystkie pozostałe dane:



	CATEGORYID	CATEGORYNAME
1	3	Książki
2	7	Gry

	PRODUCTID	PRODUCTNAME	UNITSINSTOCK	CATEGORY_CATEGORYID	SUPPLIER_SUPPLIERID
1	4	Biblia C++	20	3	5
2	2	Gotuj z TurboPascalem	20	3	5
3	8	Wiedźmin 3: Dziki Zgon	100	7	9
4	6	Cyberpunk 2077	0	7	9

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	5	Kraków	Helion	Bibliotekarska 3
2	9	Warszawa	CD Projekt Red	Jagiellońska 74

	INVOICEID	INVOICENUMBER	QUANTITY
1	1	1	3
2	10	2	2

	PRODUCTS_PRODUCTID	INVOICES_INVOICEID
1	2	1
2	4	1
3	6	1
4	6	10
5	8	10

Punkt XII

Stworzyłem klasę Address, która następnie “wbudowałem” do klasy Supplier:

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


   public Address() {
   }


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


}
```

```
package entities;


import entities.Product;


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


@Entity
```

```java
public class Supplier {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int supplierID;
    private String companyName;
//    private String street;
//    private String city;
    @OneToMany(mappedBy = "supplier", cascade = {CascadeType.PERSIST})
    private Set<Product> products;
    @Embedded
    private Address address;


    public Supplier() {
    }


    public Supplier(String companyName, String country, String city, String street, String zip)
{
        this.companyName = companyName;
        this.address = new Address(country, city, street, zip);
        products = new HashSet<>();
    }


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


    public int getSupplierID() {
        return supplierID;
    }


    public String getCompanyName() {
        return companyName;
    }


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

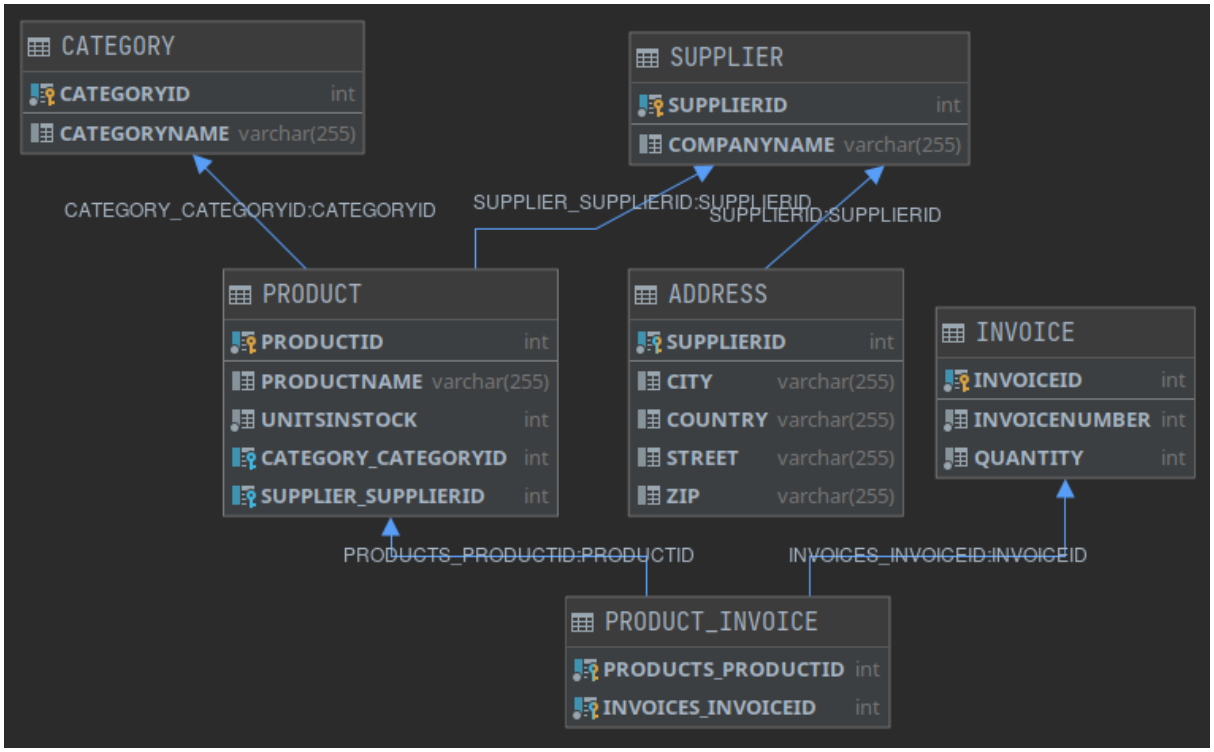
	SUPPLIERID	CITY	COUNTRY	STREET	ZIP	COMPANYNAME
1	5	Kraków	Polska	Bibliotekarska 3	30-130	Helion
2	9	Warszawa	Polska	Jagiellońska 74	12-590	CD Projekt Red

W drugim podejściu umieściłem wszystkie dane adresowe z powrotem w klasie Supplier ale zmapowałem je do osobnej tabeli:

```
@Entity
@SecondaryTable(name="ADDRESS")
public class Supplier {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int supplierID;
   private String companyName;
   @Column(table = "ADDRESS")
   private String country;
   @Column(table = "ADDRESS")
   private String city;
   @Column(table = "ADDRESS")
   private String street;
   @Column(table = "ADDRESS")
   private String zip;
   @OneToMany(mappedBy = "supplier", cascade = {CascadeType.PERSIST})
   private Set<Product> products;


   public Supplier() {
   }


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


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


   public int getSupplierID() {
      return supplierID;
   }


   public String getCompanyName() {
      return companyName;
   }


   public Set<Product> getProducts() {
```

```
        return products;
    }
}
```

Wyniki prezentuję poniżej:

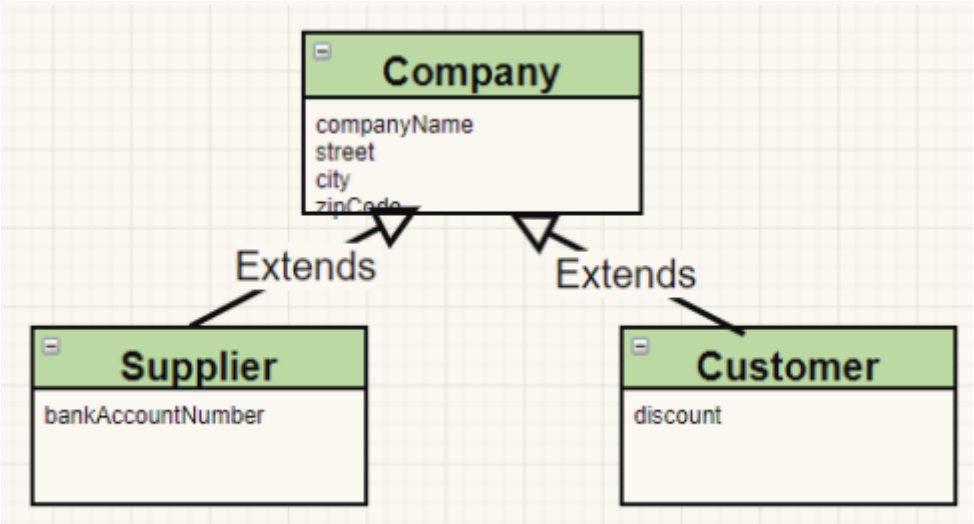


	SUPPLIERID	COMPANYNAME
1	5	Helion
2	9	CD Projekt Red

	CITY	COUNTRY	STREET	ZIP	SUPPLIERID
1	Kraków	Polska	Bibliotekarska 3	30-130	5
2	Warszawa	Polska	Jagiellońska 74	12-590	9

Punkt XIII

W tym punkcie wyprowadziłem z modelu na 3 sposoby następującą hierarchię:



Single Table:

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int id;
   private String companyName;
   private String street;
   private String city;
   private String zip;


   public Company() {
   }


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

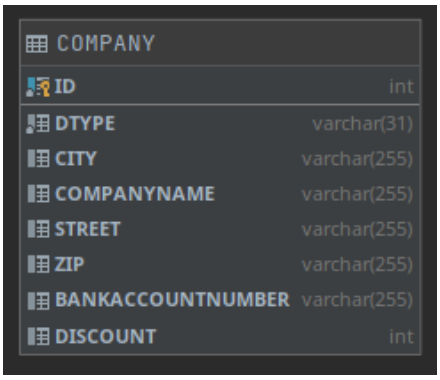
```
public class Supplier extends Company{
   private String bankAccountNumber;


   public Supplier() {
      super();
   }


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

```
@Entity
public class Customer extends Company{
   private int discount;


   public Customer() {
      super();
   }


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

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


      Supplier helion = new Supplier("7476845067904878", "Helion", "Bibliotekarska
3","Kraków",  "30-130");
      Supplier cdProj = new Supplier("0202342032132032", "CD Projekt Red",
"Jagiellońska 74", "Warszawa", "12-590");
      Customer tesco = new Customer("Tesco", "Biedna 3", "Sosnowiec", "90-112", 30);
      Customer empik = new Customer("Empik", "Czytelnicza 5", "Krynica Morska", "12-
266", 50);


      em.persist(helion);
```

```
        em.persist(cdProj);
        em.persist(tesco);
        em.persist(empik);


        List<Object[]> result1 = em.createNativeQuery("select * from Company where DTYPE
= 'Supplier'").getResultList();
        for (Object[] next: result1 )
            System.out.println(next[0]+" --- "+next[1]+" --- "+next[2]+" --- "+next[3]+" ---
"+next[4]
                    +" --- "+next[5]+" --- "+next[6]);


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

```
Hibernate:
    select
        *
    from
        Company
    where
        DTYPE = 'Supplier'
Supplier --- 1 --- Kraków --- Helion --- Bibliotekarska 3 --- 30-130 --- 7476845067904878
Supplier --- 2 --- Warszawa --- CD Projekt Red --- Jagiellońska 74 --- 12-590 --- 0202342032132032
```

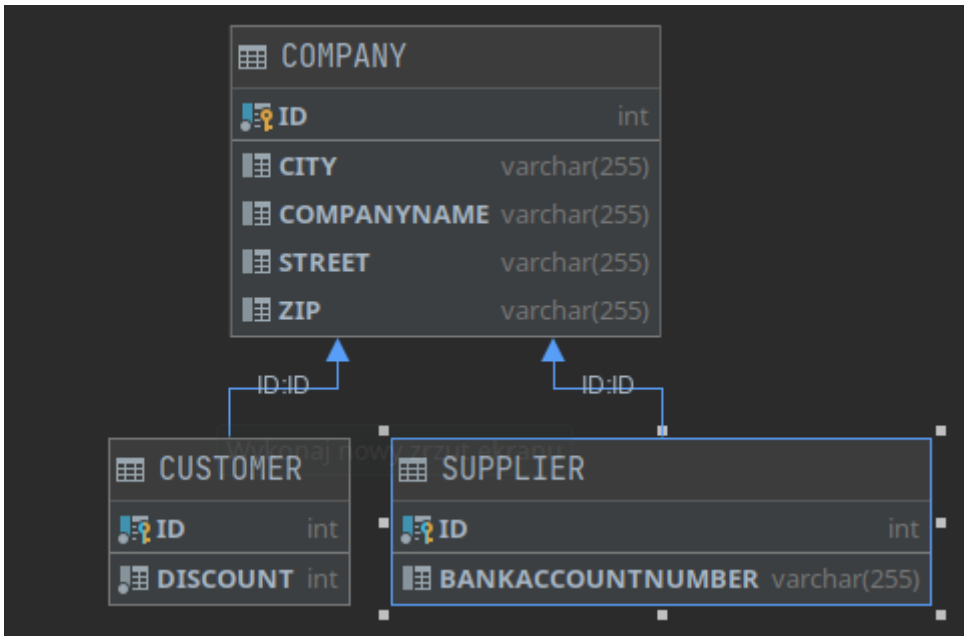


	DTYPE	ID	CITY	COMPANYNAME	STREET	ZIP	BANKACCOUNTNUMBER	DISCOUNT
1	Supplier	1	Kraków	Helion	Bibliotekarska 3	30-130	7476845067904878	<null>
2	Supplier	2	Warszawa	CD Projekt Red	Jagiellońska 74	12-590	0202342032132032	<null>
3	Customer	3	Sosnowiec	Tesco	Biedna 3	90-112	<null>	30
4	Customer	4	Krynica Morska	Empik	Czytelnicza 5	12-266	<null>	50

Joined:

```java
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String companyName;
    private String street;
    private String city;
    private String zip;


    public Company() {
    }


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

Klasy Customer oraz Supplier pozostały niezmienione.



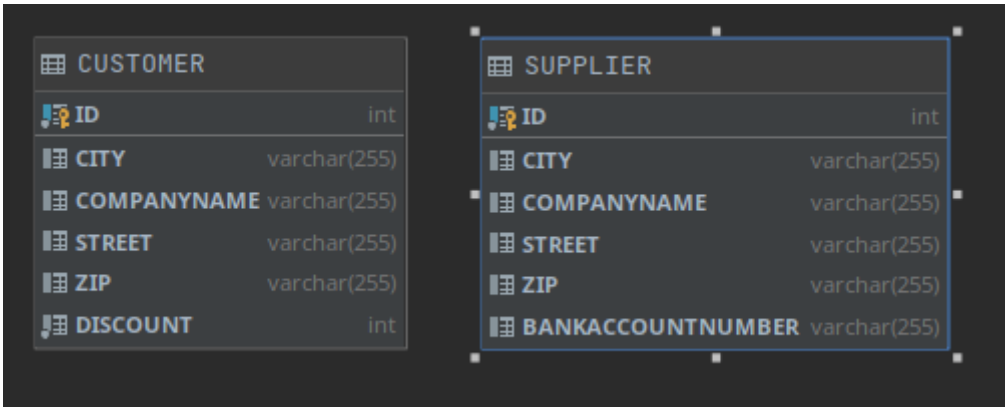
	ID	CITY	COMPANYNAME	STREET	ZIP
1	1	Kraków	Helion	Bibliotekarska 3	30-130
2	2	Warszawa	CD Projekt Red	Jagiellońska 74	12-590
3	3	Sosnowiec	Tesco	Biedna 3	90-112
4	4	Krynica Morska	Empik	Czytelnicza 5	12-266

	BANKACCOUNTNUMBER	ID
1	7476845067904878	1
2	0202342032132032	2

	DISCOUNT	ID
1	30	3
2	50	4

Table Per Class:

Jedyną zmianą wprowadzoną względem poprzedniego podejścia była zmiana adnotacji nad klasą Company z @Inheritance(strategy = InheritanceType.JOINED) na @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS).



	ID	CITY	COMPANYNAME	STREET	ZIP	BANKACCOUNTNUMBER
1	1	Kraków	Helion	Bibliotekarska 3	30-130	7476845067904878
2	2	Warszawa	CD Projekt Red	Jagiellońska 74	12-590	0202342032132032

	ID	CITY	COMPANYNAME	STREET	ZIP	DISCOUNT
1	3	Sosnowiec	Tesco	Biedna 3	90-112	30
2	4	Krynica Morska	Empik	Czytelnicza 5	12-266	50