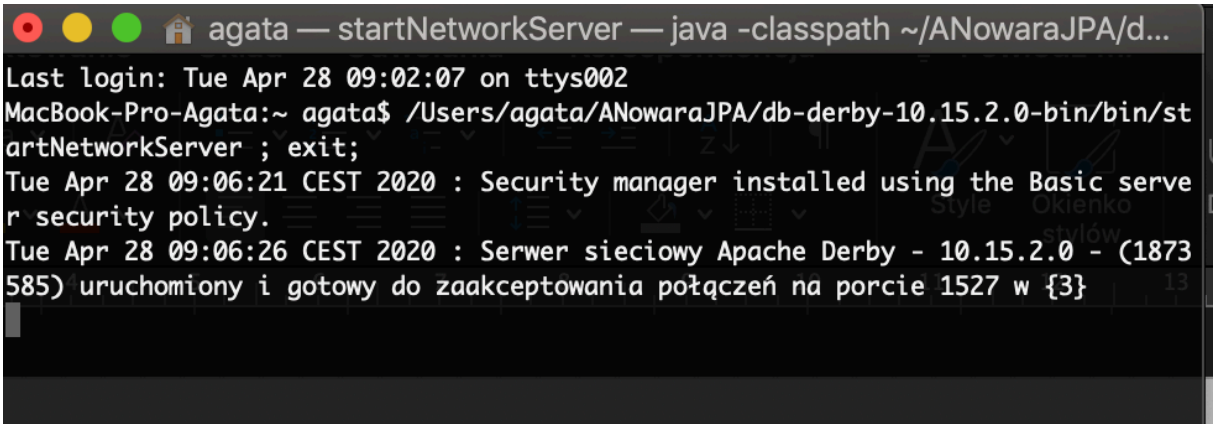
BASICS:

a), b) Pobrano i rozpakowano DBMS Derby, a następnie uruchomiono serwerDerby



c), d), e) –wspomniane w instrukcji błędy nie wystąpiły

f), g), h) Uruchomiono konsolę ij, następnie podpięto się do serwera tworząc bazę o nazwie NowaraJPA. Na koniec przetestowano działanie wywołując komendę show tables;

Uwaga: komendę – connect ‘jdbc:derby://127.0.0.1/NowaraJPA;create=true’; wykonano wcześniej lecz nie udokumentowano zrzutem ekranu.

```
ij> connect 'jdbc:derby://127.0.0.1/NowaraJPA';
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                     |
APP                 |PRODUCT                       |
APP                 |SUPPLIER                      |
```

i), j) , k)

Stworzono projekt z wykorzystaniem narzędzie Intellij Ultimate.

Stworzono klasę Product, zawierającą dwa wskazane pola tj. ProductName, UnitsOnStock oraz dodatkowe pole ProductID. Do klasy dodano konstruktor przyjmujący nazwę produktu oraz ilość na stanie

Klasę uzupełniono także o elementy konieczne do zmapowania klasy do bazy danych tj. @Entity, @Id oraz pusty konstruktor.

Dodatkowo za pomocą @GeneratedValue ustawiono automatyczną generację identyfikatora produktu

```java
import javax.persistence.*;

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

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

m)
Uzupełnienie szkieletu hibconfiga o niezbędne property oraz dodanie mapowania klasy Product

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:derby://localhost:1527/NowaraJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"/>
    </session-factory>
</hibernate-configuration>
```

III. **W maine stwórz przykładowy produkt i utrwal go w BD z wykorzystaniem Hibernate'a.**

Stworzono produkt jabłko i utrwalono w bazie danych z wykorzystaniem Hibernate'a

```java
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();
        Transaction tx = session.beginTransaction();
        Product prod1 = new Product( ProductName: "jabłko", UnitsOnStock: 100);
        session.save(prod1);
        tx.commit();
        try {
            System.out.println("querying all the managed entities...");
            final Metamodel metamodel = session.getSessionFactory().getMetamodel();
            for (EntityType<?> entityType : metamodel.getEntities()) {
                final String entityName = entityType.getName();
                final Query query = session.createQuery( s: "from " + entityName);
                System.out.println("executing: " + query.getQueryString());
                for (Object o : query.list()) {
                    System.out.println("  " + o);
                }
            }
        } finally {
            session.close();
        }
    }
}
```

Efekt wykonania się programu w Intellij

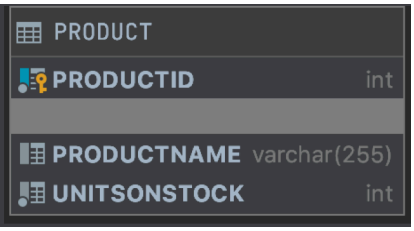
```
Hibernate:

    create table Product (
       ProductID integer not null,
        ProductName varchar(255),
        UnitsOnStock integer not null,
        primary key (ProductID)
    )
kwi 30, 2020 1:22:43 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 Product
        */ insert
        into
            Product
            (ProductName, UnitsOnStock, 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_.UnitsOnStock as unitsons3_0_
    from
        Product product0_
  Product@cdb2d95

Process finished with exit code 0
```

Po podłączeniu się z Apache Derby z poziomu Intellij uzyskano tabelę Product z dodanym przed chwilą obiektem.

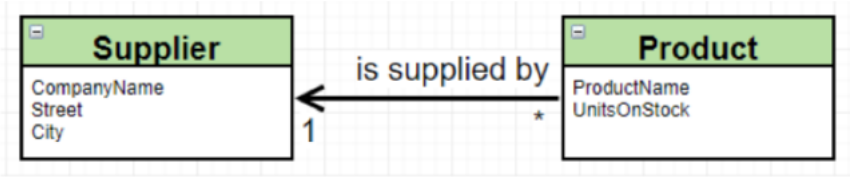
	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	3	jabłko	100

Skrypt tworzący tabelę Product oraz obecny schemat bazy danych

```
create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000000-98ec802c-0171-cabb-5de0-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null
);
```



IV. Zmodyfikuj model wprowadzając pojęcie Dostawcy jak poniżej



Stworzono klasę Supplier z zadanymi polami tj. CompanyName, Street, City, z dodatkowym polem SupplierD oraz z konstruktorem przyjmującym trzy parametry

Dodano elementy niezbędne do zmapowania klasy do bazy danych oraz automatyczne generowanie wartości pola SupplierID

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

@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;
    }
}
```

W hibconfig dodano mapowanie nowostworzonej klasy Supplier

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:derby://localhost:1527/NowaraJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"/>
        <mapping class="Supplier"/>
    </session-factory>
</hibernate-configuration>
```

Aby zamodelować relację przedstawioną na powyższym schemacie wprowadzono dodatkowe pole supplier typu Supplier oraz dodano metodę pod nazwą SetSupplier w celu ustawienia do danego produktu odpowiedniego dostawcy.

Nad polem Supplier supplier dodano deklarację @ManyToOne

```java
import javax.persistence.*;

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

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

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

W klasie Main utworzono nowy obiekt klasy Supplier, dostano się do ostatnio utworzonego produktu oraz ustawiono dostawcę odnalezionego produktu na właśnie dodanego.

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

    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Supplier sup1 = new Supplier( CompanyName: "Lidl", Street: "Sezamkowa", City: "Warszawa");
    Product lastProduct = session.get(Product.class, serializable: 3);
    session.save(sup1);
    session.save(lastProduct);
    lastProduct.SetSupplier(sup1);
    tx.commit();
    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
}
}
```

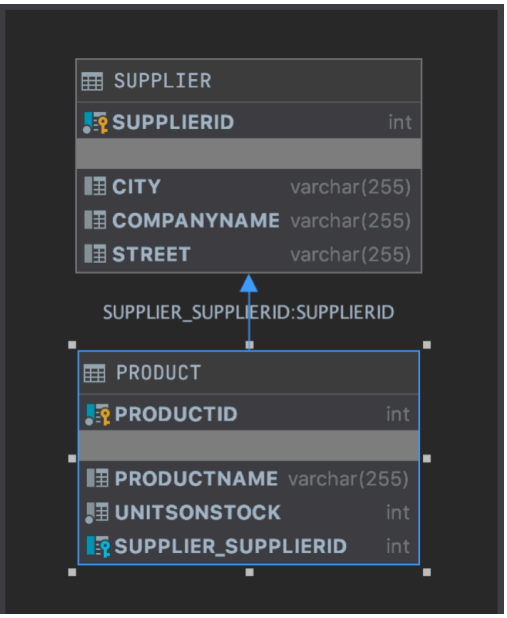
W rezultacie po podłączeniu się do Apache Derby otrzymano następujące zawartości tabel. Jak widać prawidłowo dopasowano dostawcę do produktu

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	4	Warszawa	Lidl	Sezamkowa

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIER_SUPPLIERID
1	3	jabłko	100	4

Skrypt tworzący tabele Product i Supplier ora oraz obecny schemat bazy danych

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000001-d2038474-0171-cabb-5de0-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID           INTEGER not null
        constraint "SQL0000000000-98ec802c-0171-cabb-5de0-000070f7e632"
            primary key,
    PRODUCTNAME         VARCHAR(255),
    UNITSONSTOCK        INTEGER not null,
    SUPPLIER_SUPPLIERID INTEGER
        constraint FK7WKKWFQ164SC612S9XTLYHBGE
            references SUPPLIER
);

create index "SQL0000000002-ad30c47e-0171-cabb-5de0-000070f7e632"
    on PRODUCT (SUPPLIER_SUPPLIERID);
```



V. Odwróć relacje zgodnie z poniższym schematem

a) Wariant z użyciem tabeli łącznikowej

Z klasy Product usunięto pole Supplier supplier oraz metodę SetSupplier. Reszta pozostała bez zmian

```
import javax.persistence.*;

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

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

Do klasy Supplier dodano zbiór produktów (Set<Product>) oraz metodę AddProductToSet.

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

@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 HashSet<>();

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

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

}
```

W funkcji main stworzono 3 produkty oraz jednego dostawcę. Następnie dodano wszystkie produkty do stworzonego dostawcy za pomocą metody AddProductToSet.

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

    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "banan",  UnitsOnStock: 300);
    Product prod3 = new Product( ProductName: "jagoda",  UnitsOnStock: 1000);
    Supplier sup1 = new Supplier( CompanyName: "Lidl",  Street: "Sezamkowa",  City: "Warszawa");
    sup1.AddProductToSet(prod1);
    sup1.AddProductToSet(prod2);
    sup1.AddProductToSet(prod3);
    session.save(prod1);
    session.save(prod2);
    session.save(prod3);
    session.save(sup1);
    tx.commit();
    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

W rezultacie po podłączeniu się do Apache Derby otrzymano następujące zawartości tabel. Jak widać prawidłowo dopasowano dostawcę do produktu

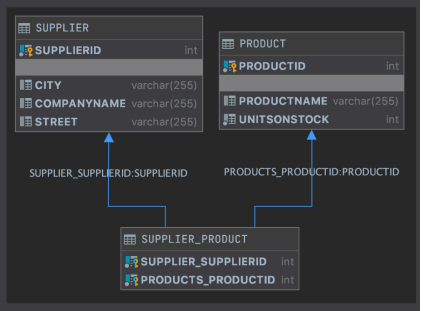
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	8	Warszawa	Lidl	Sezamkowa

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK
1	5	jabłko	100
2	6	banan	300
3	7	jagoda	1000

	SUPPLIER_SUPPLIERID	PRODUCTS_PRODUCTID
1	8	5
2	8	6
3	8	7

Skrypt tworzący tabele PRODUCT, SUPPLIER i tabelę łącznikową SUPPLIER_PRODUCT
oraz obecny schemat bazy danych

```
create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000003-536d0529-0171-cabb-5de0-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000004-bb2e452f-0171-cabb-5de0-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table SUPPLIER_PRODUCT
(
    SUPPLIER_SUPPLIERID INTEGER not null
        constraint FKHG38RTWWPUUXDYN0SKOF2WOU5
            references SUPPLIER,
    PRODUCTS_PRODUCTID  INTEGER not null
        constraint "SQL0000000010-6fe0858c-0171-cabb-5de0-000070f7e632"
            unique
        constraint FKSA8NJ74B82QQBBHDF94JLB28N
            references PRODUCT,
    constraint "SQL0000000005-cc45c536-0171-cabb-5de0-000070f7e632"
        primary key (SUPPLIER_SUPPLIERID, PRODUCTS_PRODUCTID)
);

create index "SQL0000000007-4e998544-0171-cabb-5de0-000070f7e632"
    on SUPPLIER_PRODUCT (PRODUCTS_PRODUCTID);

create index "SQL0000000008-c683454a-0171-cabb-5de0-000070f7e632"
    on SUPPLIER_PRODUCT (SUPPLIER_SUPPLIERID);
```



b) Wariant bez użycia tabeli łącznikowej

Jedyną zmianą jakiej dokonano, aby pozbyć się tabeli łącznikowej było dodanie @JoinColumn w klasie Supplier

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

@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=new HashSet<>();

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

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

}
```

W rezultacie po podłączeniu się do Apache Derby otrzymano następujące zawartości tabeli. Jak widać prawidłowo dopasowano dostawcę do produktu

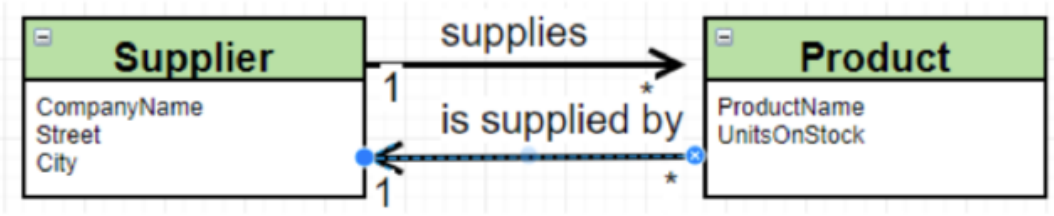
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	12	Warszawa	Lidl	Sezamkowa

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	PRODUCTS_SUPPLIERID
1	9	jabłko	100	12
2	10	banan	300	12
3	11	jagoda	1000	12

Skrypt tworzący tabele PRODUCT, SUPPLIER oraz obecny schemat bazy danych. Jak widać nie dodała się tabelą łącznikowa, a zamiast niej w tabeli Products pojawiło się dodatkowe pole Products_SupplierID

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000012-32954687-0171-cabb-5de0-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID           INTEGER not null
        constraint "SQL0000000011-c8d00681-0171-cabb-5de0-000070f7e632"
            primary key,
    PRODUCTNAME         VARCHAR(255),
    UNITSONSTOCK        INTEGER not null,
    PRODUCTS_SUPPLIERID INTEGER
        constraint FKJT4U6NELKYOUT3RXQNYQGQR2O
            references SUPPLIER
);

create index "SQL0000000013-0606c68e-0171-cabb-5de0-000070f7e632"
    on PRODUCT (PRODUCTS_SUPPLIERID);
```

VI. Zamodeluj relacje dwustronną jak poniżej:



Zmodyfikowano klasę Product dodając do niej ponownie pole Supplier supplier oraz dodając @ManyToOne oraz @JoinColumn.

Dodano metodę SetSupplier, która przypisuje do pola supplier odpowiednia wartość oraz dodaje dany produkt do zbioru produktów wybranego dostawcy (jeżeli już nie należy do tego zbioru)

```java
import javax.persistence.*;

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

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

    public void SetSupplier(Supplier supplier)
    {
        this.supplier=supplier;
        if(!supplier.alreadySupplies( product: this))
            supplier.AddProductToSet(this);
    }
}
```

Dodano adnotację @JoinColumn oraz metodę alreadySupplies typu boolean, która sprawdza, czy dany produkt należy już do zbioru produktów dostarczanych przez tego dostawcę

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

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

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

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

    public boolean alreadySupplies(Product product)
    {
        return products.contains(product);
    }

}
```

Stworzono kilka produktów, jednego dostawcę oraz przypisano stworzonego dostawcę do wszystkich produktów

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

    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "banan",  UnitsOnStock: 300);
    Product prod3 = new Product( ProductName: "jagoda",  UnitsOnStock: 1000);
    Supplier sup1 = new Supplier( CompanyName: "Lidl",  Street: "Sezamkowa",  City: "Warszawa");
    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup1);
    prod3.SetSupplier(sup1);
    session.save(prod1);
    session.save(prod2);
    session.save(prod3);
    session.save(sup1);
    tx.commit();
    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

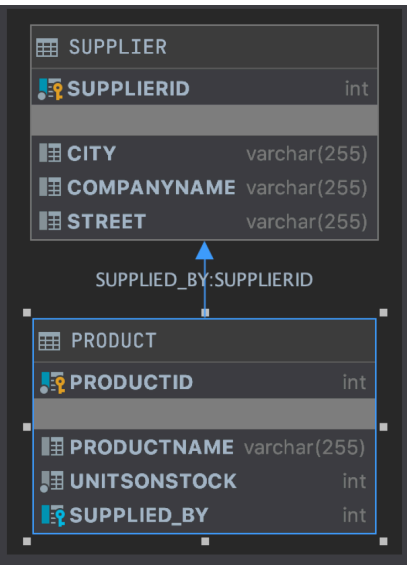
W rezultacie po podłączeniu się do Apache Derby otrzymano następujące zawartości tabel. Jak widać prawidłowo dopasowano dostawcę do produktu

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	20	Warszawa	Lidl	Sezamkowa

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	SUPPLIED_BY
1	17	jabłko	100	20
2	18	banan	300	20
3	19	jagoda	1000	20

Skrypt tworzący tabele PRODUCT, SUPPLIER oraz obecny schemat bazy danych

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000016-6530477f-0171-cabb-5de0-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000015-f9f70779-0171-cabb-5de0-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create index "SQL0000000017-fa53c786-0171-cabb-5de0-000070f7e632"
    on PRODUCT (SUPPLIED_BY);
```



VII. Dodaj klase Category z property int CategoryID, String Name oraz listą produktow List<Product> Products

Dodanie klasy Category zawierającej trzy pola tj. CategoryID, Name oraz listę produktów Products, a także konstruktor

Dodanie elementów niezbędnych do zmapowania klasy do bazy, a następnie zamodelowanie relacji między klasą Category i Product.

Dodano również dwie metody AddProductToCategory oraz AlreadyContains (sprawdza, czy dany produkt należy już do list tej kategorii)

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

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

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

    public List<Product> getProductsFromCategory()
    {
        return this.Products;
    }

    public boolean alreadyContains(Product product)
    {
        return this.Products.contains(product);
    }


}
```

W hibconfig dodano mapowanie klasy Category

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:derby://localhost:1527/NowaraJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"/>
        <mapping class="Supplier"/>
        <mapping class="Category"/>
    </session-factory>
</hibernate-configuration>
```

W klasie Product dodano pole category typu Category wraz z dwiema adnotacjami @ManyToOne oraz @JoinColumn.

Dodano również metodę SetCategory, która ustawia wartość pola category oraz dodaje produkt do listy odpowiedniej kategorii (po uprzednim sprawdzeniu, czy już się tam nie znajduje)

```java
import javax.persistence.*;

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

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

    public void SetSupplier(Supplier supplier)
    {
        this.supplier=supplier;
        if(!supplier.alreadySupplies( product: this))
            supplier.AddProductToSet(this);
    }

    public void SetCategory(Category category)
    {
        this.category=category;
        if(!category.alreadyContains( product: this))
            category.AddProductToCategory(this);
    }
}
```

W klasie main utworzono pięć produktów, dwóch dostawców oraz dwie kategorie i przypisano produktom odpowiednich dostawców oraz kategorie

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

    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "banan",  UnitsOnStock: 300);
    Product prod3 = new Product( ProductName: "jagoda",  UnitsOnStock: 1000);
    Product prod4 = new Product( ProductName: "papryka",  UnitsOnStock: 200);
    Product prod5 = new Product( ProductName: "ogórek",  UnitsOnStock: 100);
    Supplier sup1 = new Supplier( CompanyName: "Lidl",  Street: "Sezamkowa",  City: "Warszawa");
    Supplier sup2 = new Supplier( CompanyName: "Biedronka",  Street: "Miła",  City: "Warszawa");
    Category cat1 = new Category( Name: "owoce");
    Category cat2 = new Category( Name: "warzywa");
    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup1);
    prod3.SetSupplier(sup1);
    prod4.SetSupplier(sup2);
    prod5.SetSupplier(sup2);

    prod1.SetCategory(cat1);
    prod2.SetCategory(cat1);
    prod3.SetCategory(cat1);
    prod4.SetCategory(cat2);
    prod4.SetCategory(cat2);

    session.save(prod1);
    session.save(prod2);
    session.save(prod3);
    session.save(prod4);
    session.save(prod5);
    session.save(sup1);
    session.save(sup2);
    session.save(cat1);
    session.save(cat2);
    tx.commit();
```

W rezultacie po podłączeniu się do Apache Derby otrzymano następujące zawartości tabel. Jak widać prawidłowo dopasowano dostawcę oraz kategorię do produktu

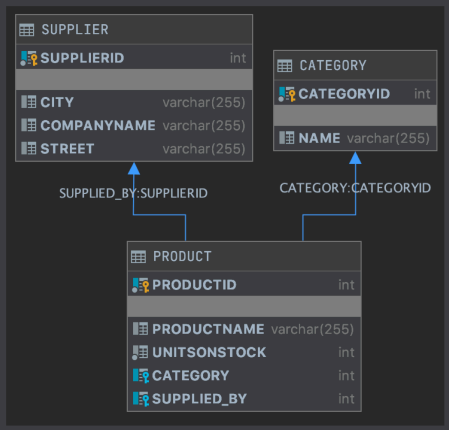
	SUPPLIERID	CITY	COMPANYNAME	STREET
1	35	Warszawa	Lidl	Sezamkowa
2	36	Warszawa	Biedronka	Miła

	CATEGORYID	NAME
1	37	owoce
2	38	warzywa

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	30	jabłko	100	37	35
2	31	banan	300	37	35
3	32	jagoda	1000	37	35
4	33	papryka	200	38	36
5	34	ogórek	100	38	36

Skrypt tworzący tabele PRODUCT, SUPPLIER, CATEGORY oraz obecny schemat bazy danych

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000023-7b0b09e8-0171-cabb-5de0-000070f7e632"
            primary key,
    NAME       VARCHAR(255)
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000025-b8d389f4-0171-cabb-5de0-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000024-f9eac9ee-0171-cabb-5de0-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create index "SQL0000000026-37c549fa-0171-cabb-5de0-000070f7e632"
    on PRODUCT (CATEGORY);

create index "SQL0000000027-a1c0c9fe-0171-cabb-5de0-000070f7e632"
    on PRODUCT (SUPPLIED_BY);
```



W celu wydobycia produktów z zadanej kategorii do klasy Product dodano metodę getName, która umożliwia dostęp do prywatnego pola ProductName

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

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

    public String GetName()
    {
        return this.ProductName;
    }

    public void SetSupplier(Supplier supplier)
    {
        this.supplier=supplier;
        if(!supplier.alreadySupplies( product: this))
            supplier.AddProductToSet(this);
    }

    public void SetCategory(Category category)
    {
        this.category=category;
        if(!category.alreadyContains( product: this))
            category.AddProductToCategory(this);
    }
}
```

W funkcji main pobrano kategorię o numerze ID 37 (kategoria owoce) i przeiterowano po jej liście produktów wypisując ich nazwy na ekran.

```java
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Category fruits = session.get(Category.class, serializable: 37);
    for (Product prod : fruits.getProductsFromCategory())
        System.out.println(prod.GetName());
    tx.commit();
    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

W wyniku wykonania się programu otrzymano poniższy rezultat, jakim jest wypisanie 3 produktów z kategorii owoce dostępnych w mojej bazie danych tj. jabłko, banan, jagoda

```
Hibernate:
    select
        products0_.CATEGORY as category4_1_0_,
        products0_.ProductID as producti1_1_0_,
        products0_.ProductID as producti1_1_1_,
        products0_.ProductName as productn2_1_1_,
        products0_.UnitsOnStock as unitsons3_1_1_,
        products0_.CATEGORY as category4_1_1_,
        products0_.SUPPLIED_BY as supplied5_1_1_,
        supplier1_.SupplierID as supplier1_2_2_,
        supplier1_.City as city2_2_2_,
        supplier1_.CompanyName as companyn3_2_2_,
        supplier1_.Street as street4_2_2_
    from
        Product products0_
    left outer join
        Supplier supplier1_
            on products0_.SUPPLIED_BY=supplier1_.SupplierID
    where
        products0_.CATEGORY=?
jabłko
banan
jagoda
```

W celu wypisania nazwy kategorii, do której należy produkt w klasie Category dodano nową metodę GetName, która umożliwia dostęp do prywatnego pola Name

```
public String GetName()
{
    return this.Name;
}
```

Z kolei w klasie produkt dodano metodę GetCategoryName w celu uzyskania dostępu do kategorii, a dokładniej do jej nazwy

```
public String GetCategoryName()
{
    return this.category.GetName();
}
```

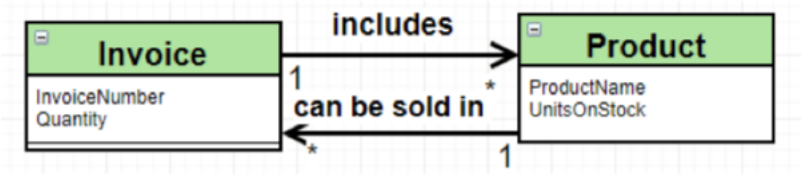
W funkcji main pobrano produkt o numerze ID 32 oraz wypisano kategorię, do której dany produkt należy

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Product prod1 = session.get(Product.class, serializable: 32);
    System.out.println("Product: "+ prod1.GetName()+ " kategoria: "+ prod1.GetCategoryName());
    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

Rezultat wykonania się powyższego program widoczny jest poniżej. Widać, że wypisano nazwę właściwej kategorii

```
Hibernate:
    select
        product0_.ProductID as producti1_1_0_,
        product0_.ProductName as productn2_1_0_,
        product0_.UnitsOnStock as unitsons3_1_0_,
        product0_.CATEGORY as category4_1_0_,
        product0_.SUPPLIED_BY as supplied5_1_0_,
        category1_.CategoryID as category1_0_1_,
        category1_.Name as name2_0_1_,
        supplier2_.SupplierID as supplier1_2_2_,
        supplier2_.City as city2_2_2_,
        supplier2_.CompanyName as companyn3_2_2_,
        supplier2_.Street as street4_2_2_
    from
        Product product0_
    left outer join
        Category category1_
            on product0_.CATEGORY=category1_.CategoryID
    left outer join
        Supplier supplier2_
            on product0_.SUPPLIED_BY=supplier2_.SupplierID
    where
        product0_.ProductID=?
Product: jagoda kategoria: owoce
```

VIII. Zamodeluj relacje wiele-do-wielu, jak poniżej:



Stworzono klasę Invoice z zadanymi polami oraz konstruktorem, dodano elementy konieczne do zmapowania klasy do bazy danych. Później dodano metody dostępowe do prywatnych pól.

W celu zamodelowania relacji wiele-do-wielu umieszczono w klasie zbiór produktów IncludedProducts, który zawiera wszystkie produkty sprzedane w ramach danej faktury, a nad nim adnotację @ManyToMany

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

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

    public void AddProductToSet(Product product)
    {
        IncludedProducts.add(product);
        this.Quantity+=1;
        product.GetSoldIn().add(this);
    }

    public int GetInvoiceNumber()
    {
        return this.InvoiceNumber;
    }

    public int GetQuantity()
    {
        return this.Quantity;
    }

    public Set<Product> GetIncludedProducts()
    {
        return this.IncludedProducts;
    }
}
```

Dodano mapowanie klasy Invoice w hibconfig.

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:derby://localhost:1527/NowaraJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="Product"/>
        <mapping class="Supplier"/>
        <mapping class="Category"/>
        <mapping class="Invoice"/>
    </session-factory>
</hibernate-configuration>
```

Jedynymi zmianami wprowadzonymi w klasie Product jest dodanie zbioru SoldIn (wraz z adnotacją @ManyToMany), zawierającego wszystkie faktury w ramach, których sprzedany był dany produkt oraz stworzenie metody dostępowej do wyżej wymienionego zbioru.

```java
@ManyToMany(mappedBy = "IncludedProducts")
private Set<Invoice> SoldIn = new HashSet<>();
```

```java
public Set<Invoice> GetSoldIn()
{
    return this.SoldIn;
}
```

W funkcji main stworzono kilka produktów, dwie kategorie, dwóch dostawców oraz dwie faktury i połączono je między sobą

```java
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "papryka",  UnitsOnStock: 200);
    Product prod3 = new Product( ProductName: "ogórek",  UnitsOnStock: 100);
    Supplier sup1 = new Supplier( CompanyName: "Lidl",  Street: "Sezamkowa",  City: "Warszawa");
    Supplier sup2 = new Supplier( CompanyName: "Biedronka",  Street: "Miła",  City: "Warszawa");
    Category cat1 = new Category( Name: "owoce");
    Category cat2 = new Category( Name: "warzywa");
    Invoice invoice1 = new Invoice( Quantity: 0);
    Invoice invoice2 = new Invoice( Quantity: 0);

    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup2);
    prod3.SetSupplier(sup2);

    prod1.SetCategory(cat1);
    prod2.SetCategory(cat2);
    prod3.SetCategory(cat2);

    invoice1.AddProductToSet(prod1);
    invoice1.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod1);
    invoice2.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod2);
    ;
    session.save(prod1);
    session.save(prod2);
    session.save(prod3);
    session.save(sup1);
    session.save(sup2);
    session.save(cat1);
    session.save(cat2);
    session.save(invoice1);
    session.save(invoice2);
```

Rezultat wykonania się powyższego programu zaprezentowany dla tabeli: PRODUCT, INVOICE, PRODUCT-INVOICE

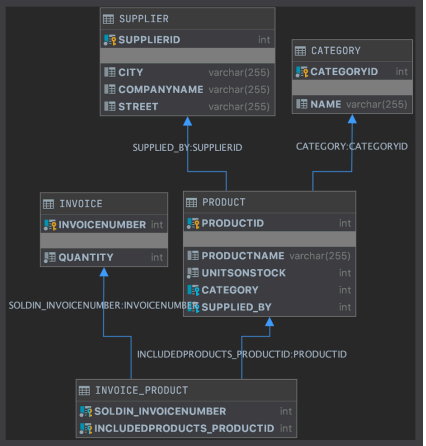
	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	39	jabłko	100	44	42
2	40	papryka	200	45	43
3	41	ogórek	100	45	43

	INVOICENUMBER	QUANTITY
1	46	2
2	47	3

	SOLDIN_INVOICENUMBER	INCLUDEDPRODUCTS_PRODUCTID
1	46	39
2	46	41
3	47	39
4	47	40
5	47	41

Skrypt tworzący tabele PRODUCT,INVOICE, PRODUCT-INVOICE oraz obecny schemat bazy danych

```
create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000029-1f824bb2-0171-cabb-5de0-000070f7e632"
            primary key,
    QUANTITY      INTEGER not null
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000031-9d984bbf-0171-cabb-5de0-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create table INVOICE_PRODUCT
(
    SOLDIN_INVOICENUMBER       INTEGER not null
        constraint FK4GSEJSS8RXX2KG2CA370EFQDG
            references INVOICE,
    INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
        constraint FKJ6G454XCSICKMSRPURMYPUY02
            references PRODUCT,
    constraint "SQL0000000030-2bff0bb9-0171-cabb-5de0-000070f7e632"
        primary key (SOLDIN_INVOICENUMBER, INCLUDEDPRODUCTS_PRODUCTID)
);

create index "SQL0000000033-fbd88bcc-0171-cabb-5de0-000070f7e632"
    on INVOICE_PRODUCT (INCLUDEDPRODUCTS_PRODUCTID);

create index "SQL0000000034-f29a0bd1-0171-cabb-5de0-000070f7e632"
    on INVOICE_PRODUCT (SOLDIN_INVOICENUMBER);
```



W funkcji main pobrano fakturę o numerze ID 47 i przeiterowano po jej zbiorze produktów wypisując ich nazwy na ekran.

```java
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Invoice invoice = session.get(Invoice.class, serializable: 47);
    for(Product product: invoice.GetIncludedProducts())
        System.out.println(product.GetName());
    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

Rezultat wykonania się powyższego programu

```
Hibernate:
    select
        includedpr0_.SoldIn_InvoiceNumber as soldin_i1_2_0_,
        includedpr0_.IncludedProducts_ProductID as included2_2_0_,
        product1_.ProductID as producti1_3_1_,
        product1_.ProductName as productn2_3_1_,
        product1_.UnitsOnStock as unitsons3_3_1_,
        product1_.CATEGORY as category4_3_1_,
        product1_.SUPPLIED_BY as supplied5_3_1_,
        category2_.CategoryID as category1_0_2_,
        category2_.Name as name2_0_2_,
        supplier3_.SupplierID as supplier1_4_3_,
        supplier3_.City as city2_4_3_,
        supplier3_.CompanyName as companyn3_4_3_,
        supplier3_.Street as street4_4_3_
    from
        Invoice_Product includedpr0_
    inner join
        Product product1_
            on includedpr0_.IncludedProducts_ProductID=product1_.ProductID
    left outer join
        Category category2_
            on product1_.CATEGORY=category2_.CategoryID
    left outer join
        Supplier supplier3_
            on product1_.SUPPLIED_BY=supplier3_.SupplierID
    where
        includedpr0_.SoldIn_InvoiceNumber=?
papryka
jabłko
ogórek
```

W funkcji main pobrano produkt o numerze ID 41 i przeiterowano po zbiorze transakcji, w których został sprzedany, wypisując numery tych transakcji/faktur na ekran.

```
public static void main(final String[] args) throws Exception {
    final Session session = getSession();
    Transaction tx = session.beginTransaction();
    Product product = session.get(Product.class, serializable: 41);
    for(Invoice invoice: product.GetSoldIn())
        System.out.println(invoice.GetInvoiceNumber());
    try {
        System.out.println("querying all the managed entities...");
        final Metamodel metamodel = session.getSessionFactory().getMetamodel();
        for (EntityType<?> entityType : metamodel.getEntities()) {
            final String entityName = entityType.getName();
            final Query query = session.createQuery( s: "from " + entityName);
            System.out.println("executing: " + query.getQueryString());
            for (Object o : query.list()) {
                System.out.println("  " + o);
            }
        }
    } finally {
        session.close();
    }
}
```

Rezultat wykonania się powyższego programu

```
Hibernate:
    select
        product0_.ProductID as producti1_3_0_,
        product0_.ProductName as productn2_3_0_,
        product0_.UnitsOnStock as unitsons3_3_0_,
        product0_.CATEGORY as category4_3_0_,
        product0_.SUPPLIED_BY as supplied5_3_0_,
        category1_.CategoryID as category1_0_1_,
        category1_.Name as name2_0_1_,
        supplier2_.SupplierID as supplier1_4_2_,
        supplier2_.City as city2_4_2_,
        supplier2_.CompanyName as companyn3_4_2_,
        supplier2_.Street as street4_4_2_
    from
        Product product0_
    left outer join
        Category category1_
            on product0_.CATEGORY=category1_.CategoryID
    left outer join
        Supplier supplier2_
            on product0_.SUPPLIED_BY=supplier2_.SupplierID
    where
        product0_.ProductID=?
Hibernate:
    select
        soldin0_.IncludedProducts_ProductID as included2_2_0_,
        soldin0_.SoldIn_InvoiceNumber as soldin_i1_2_0_,
        invoice1_.InvoiceNumber as invoicen1_1_1_,
        invoice1_.Quantity as quantity2_1_1_
    from
        Invoice_Product soldin0_
    inner join
        Invoice invoice1_
            on soldin0_.SoldIn_InvoiceNumber=invoice1_.InvoiceNumber
    where
        soldin0_.IncludedProducts_ProductID=?
46
47
```

IX/X. **Stwórz nowego maina w którym zrobisz to samo co w punkcie VI ale z wykorzystaniem JPA**

Najpierw stworzono nowy plik konfiguracyjny persistence.xml

```xml
<?xml version="1.0"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation = "http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
    <persistence-unit name="derby" 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/NowaraJPA"/>
            <property name="hibernate.show_sql"
                      value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hmb2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```

Następnie stworzono nową klasę MainJPA. W funkcji main tej klasy utworzono kilka produktów i dwóch dostawców oraz połączono ich ze sobą realizując zadanie z punktu VI

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

public class MainJPA
{
    public static  void main(String argv[])
    {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
        EntityManager em = emf.createEntityManager();
        EntityTransaction etx = em.getTransaction();
        etx.begin();
        Product prod1 = new Product ( ProductName: "jabłko",  UnitsOnStock: 100);
        Product prod2 = new Product( ProductName: "banan",  UnitsOnStock: 200);
        Product prod3 = new Product( ProductName: "jagoda",  UnitsOnStock: 1000);
        Product prod4 = new Product( ProductName: "pomidor",  UnitsOnStock: 400);
        Supplier sup1 = new Supplier( CompanyName: "Lidl",  Street: "Sezamkowa",  City: "Warszawa");
        Supplier sup2 = new Supplier( CompanyName: "Biedronka",  Street: "Miła",  City: "Warszawa");
        sup1.AddProductToSet(prod1);
        sup1.AddProductToSet(prod3);
        sup2.AddProductToSet(prod2);
        sup2.AddProductToSet(prod4);
        em.persist(prod1);
        em.persist(prod2);
        em.persist(prod3);
        em.persist(prod4);
        em.persist(sup1);
        em.persist(sup2);

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

    }
}
```

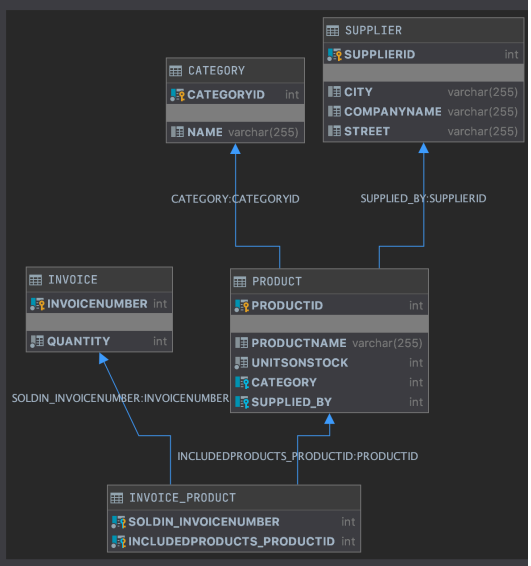
Rezultat wykonania się powyższego programu widoczny po podłączeniu do Apache Derby.

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	1	jabłko	100	<null>	5
2	2	banan	200	<null>	6
3	3	jagoda	1000	<null>	5
4	4	pomidor	400	<null>	6

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	5	Warszawa	Lidl	Sezamkowa
2	6	Warszawa	Biedronka	Miła

Skrypt generujący bazę wraz z obecnym schematem bazy danych. (Pozostawiono wcześniej utworzone tabele Category, Invoice oraz Invoice-Product).

```
create table CATEGORY
(
    CATEGORYID INTEGER not null
        constraint "SQL0000000000-44c541fa-0171-d036-e262-000070f7e632"
            primary key,
    NAME       VARCHAR(255)
);

create table INVOICE
(
    INVOICENUMBER INTEGER not null
        constraint "SQL0000000001-00400201-0171-d036-e262-000070f7e632"
            primary key,
    QUANTITY      INTEGER not null
);

create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000004-f2f9c216-0171-d036-e262-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000003-e75a420f-0171-d036-e262-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);

create table INVOICE_PRODUCT
(
    SOLDIN_INVOICENUMBER       INTEGER not null
        constraint FK4GSEJSS8RXX2KG2CA370EFQDG
            references INVOICE,
    INCLUDEDPRODUCTS_PRODUCTID INTEGER not null
        constraint FKJ6G454XCSICKMSRPURMYPUY02
            references PRODUCT,
    constraint "SQL0000000002-bbc70208-0171-d036-e262-000070f7e632"
        primary key (SOLDIN_INVOICENUMBER, INCLUDEDPRODUCTS_PRODUCTID)
);

create index "SQL0000000005-4ea5821d-0171-d036-e262-000070f7e632"
    on INVOICE_PRODUCT (INCLUDEDPRODUCTS_PRODUCTID);

create index "SQL0000000006-194c4222-0171-d036-e262-000070f7e632"
    on INVOICE_PRODUCT (SOLDIN_INVOICENUMBER);

create index "SQL0000000007-73f94227-0171-d036-e262-000070f7e632"
    on PRODUCT (CATEGORY);

create index "SQL0000000008-2eac822c-0171-d036-e262-000070f7e632"
    on PRODUCT (SUPPLIED_BY);
```



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

W klasie Product nad zbiorem faktur dodano cascade = CascadeType.PERSIST

```
@ManyToMany(mappedBy = "IncludedProducts", cascade = CascadeType.PERSIST)
private Set<Invoice> SoldIn = new HashSet<>();
```

W klasie Invoice również umieszczono cascade = CascadeType.PERSIST

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

Dzięki temu można tworzyć faktury wraz z nowymi produktami, a także produkty wraz z nową fakturą

Najpierw przetestowano wersję tworzenia produktów wraz z fakturą, w tym celu uruchomiono poniżej przedstawiony program.

```
public static  void main(String argv[])
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "papryka",  UnitsOnStock: 200);
    Product prod3 = new Product( ProductName: "ogórek",  UnitsOnStock: 100);
    Supplier sup1 = new Supplier( CompanyName: "Lidl",  Street: "Sezamkowa",  City: "Warszawa");
    Supplier sup2 = new Supplier( CompanyName: "Biedronka",  Street: "Miła",  City: "Warszawa");
    Category cat1 = new Category( Name: "owoce");
    Category cat2 = new Category( Name: "warzywa");
    Invoice invoice1 = new Invoice( Quantity: 0);
    Invoice invoice2 = new Invoice( Quantity: 0);

    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup2);
    prod3.SetSupplier(sup2);
    prod1.SetCategory(cat1);
    prod2.SetCategory(cat2);
    prod3.SetCategory(cat2);

    invoice1.AddProductToSet(prod1);
    invoice1.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod1);
    invoice2.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod2);

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

    em.persist(sup1);
    em.persist(sup2);

    em.persist(cat1);
    em.persist(cat2);

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

Jak widać w bazie danych utrwalamy za pomocą *persist* jedynie nowostworzone faktury, kategorie oraz dostawców, a produkty pomijamy.

Poniżej możemy zaobserwować, iż wszystko dodało się poprawnie.

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	2	ogórek	100	9	7
2	4	jabłko	100	8	6
3	5	papryka	200	9	7

	INVOICENUMBER	QUANTITY
1	1	2
2	3	3

	SOLDIN_INVOICENUMBER	INCLUDEDPRODUCTS_PRODUCTID
1	1	2
2	1	4
3	3	2
4	3	4
5	3	5

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	6	Warszawa	Lidl	Sezamkowa
2	7	Warszawa	Biedronka	Miła

	CATEGORYID	NAME
1	8	owoce
2	9	warzywa

Tym razem przetestowano wariant tworzenia faktur wraz z produktami, w tym celu uruchomiono poniżej przedstawiony program.

```
public static  void main(String argv[])
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "papryka",  UnitsOnStock: 200);
    Product prod3 = new Product( ProductName: "ogórek",  UnitsOnStock: 100);
    Supplier sup1 = new Supplier( CompanyName: "Lidl",  Street: "Sezamkowa",  City: "Warszawa");
    Supplier sup2 = new Supplier( CompanyName: "Biedronka",  Street: "Miła",  City: "Warszawa");
    Category cat1 = new Category( Name: "owoce");
    Category cat2 = new Category( Name: "warzywa");
    Invoice invoice1 = new Invoice( Quantity: 0);
    Invoice invoice2 = new Invoice( Quantity: 0);

    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup2);
    prod3.SetSupplier(sup2);
    prod1.SetCategory(cat1);
    prod2.SetCategory(cat2);
    prod3.SetCategory(cat2);

    invoice1.AddProductToSet(prod1);
    invoice1.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod1);
    invoice2.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod2);

    em.persist(prod1);
    em.persist(prod2);
    em.persist(prod3);

    em.persist(sup1);
    em.persist(sup2);

    em.persist(cat1);
    em.persist(cat2);

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

Jak widać w bazie danych utrwalamy za pomocą *persist* jedynie nowostworzone produkty, kategorie oraz dostawców, a faktury pomijamy.

Poniżej możemy zaobserwować, że i tym razem wszystko dodało się poprawnie.

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	1	jabłko	100	8	6
2	3	ogórek	100	9	7
3	5	papryka	200	9	7

	INVOICENUMBER	QUANTITY
1	2	3
2	4	2

	SOLDIN_INVOICENUMBER	INCLUDEDPRODUCTS_PRODUCTID
1	2	1
2	2	3
3	2	5
4	4	1
5	4	3

	SUPPLIERID	CITY	COMPANYNAME	STREET
1	6	Warszawa	Lidl	Sezamkowa
2	7	Warszawa	Biedronka	Miła

	CATEGORYID	NAME
1	8	owoce
2	9	warzywa

```
select PRODUCTID, PRODUCTNAME, UNITSONSTOCK, CATEGORY, SUPPLIED_BY, COMPANYNAME, INVOICENUMBER, QUANTITY
from PRODUCT
join INVOICE_PRODUCT IP on PRODUCT.PRODUCTID = IP.INCLUDEDPRODUCTS_PRODUCTID
join INVOICE I on IP.SOLDIN_INVOICENUMBER = I.INVOICENUMBER
join SUPPLIER S on PRODUCT.SUPPLIED_BY = S.SUPPLIERID
```

Output | Result 2

5 rows

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY	COMPANYNAME	INVOICENUMBER	QUANTITY
1	1	jabłko	100	8	6	Lidl	2	3
2	1	jabłko	100	8	6	Lidl	4	2
3	3	ogórek	100	9	7	Biedronka	2	3
4	3	ogórek	100	9	7	Biedronka	4	2
5	5	papryka	200	9	7	Biedronka	2	3

XII. Embedded class: Dodaj do modelu klasę adres. „Wbuduj” ją do tabeli Dostawców. Następnie zmodyfikuj model w taki sposób, że dane adresowe znajdują się w klasie dostawców. Zmapuj to do dwóch osobnych tabel

Utworzono nową klasę Adres z andotacją Embeddable (wkazującą na fakt, że klasa ta zostanie wbudowana do innej klasy) z trzema polami:

- street
- city
- zipCode

oraz dwoma konstruktorami.

```java
import javax.persistence.*;

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

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

W klasie Supplier usunięto istniejące dotychczas pola street i city i wbudowano klasę Address dodając pole address typu Address z adnotację @Embeded.

```
@Embedded
private Address address;
```

Wprowadzono również drobne zmiany w konstruktorze, ze względu na zmianę atrybutów klasy

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

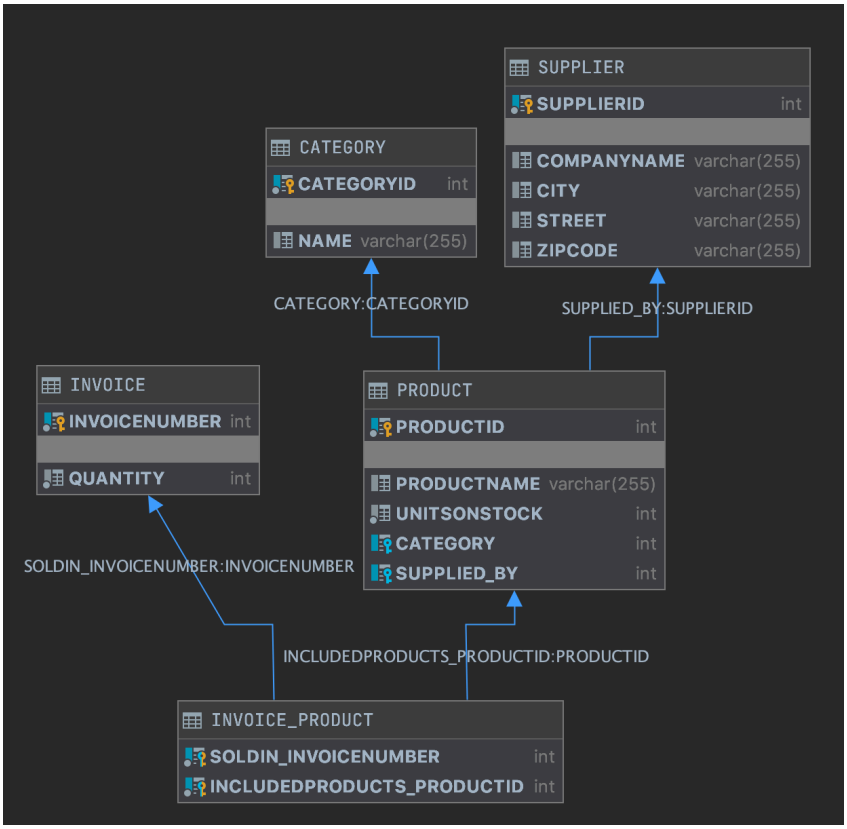
W funkcji main standardowo dodano obiekty wszystkich klas, w celu dodania dostawców, stworzono dla nich najpierw oddzielne adresy.

```
public static  void main(String argv[])
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "papryka",  UnitsOnStock: 200);
    Product prod3 = new Product( ProductName: "ogórek",  UnitsOnStock: 100);
    Address address1 = new Address( street: "Sezamkowa",  city: "Warszawa",  zipCode: "40-308");
    Address address2 = new Address( street: "Miła",  city: "Warszawa",  zipCode: "30-561");
    Supplier sup1 = new Supplier( CompanyName: "Lidl", address1);
    Supplier sup2 = new Supplier( CompanyName: "Biedronka", address2);
    Category cat1 = new Category( Name: "owoce");
    Category cat2 = new Category( Name: "warzywa");
    Invoice invoice1 = new Invoice( Quantity: 0);
    Invoice invoice2 = new Invoice( Quantity: 0);

    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup2);
    prod3.SetSupplier(sup2);
    prod1.SetCategory(cat1);
    prod2.SetCategory(cat2);
    prod3.SetCategory(cat2);

    invoice1.AddProductToSet(prod1);
    invoice1.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod1);
    invoice2.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod2);

    em.persist(prod1);
    em.persist(prod2);
    em.persist(prod3);

    em.persist(sup1);
    em.persist(sup2);

    em.persist(cat1);
    em.persist(cat2);

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

Po wykonaniu się powyższego programu, tabela dostawców wraz z ich adresem została wypełniona w sposób poprawny

	SUPPLIERID	COMPANYNAME	CITY	STREET	ZIPCODE
1	6	Lidl	Warszawa	Sezamkowa	40-308
2	7	Biedronka	Warszawa	Miła	30-561

Schemat bazy danych:



Skrypt tworzący tabelę Supplier:

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000031-b549495a-0171-d036-e262-000070f7e632"
            primary key,
    COMPANYNAME VARCHAR(255),
    CITY        VARCHAR(255),
    STREET      VARCHAR(255),
    ZIPCODE     VARCHAR(255)
);
```

Tym razem wszystkie dane adresowe umieszczono z powrotem w klasie Supplier, ale zmapowano je do osobnej tabeli ADDRESS_TBL. Cen ten osiągnięto dzięki użyciu adnotacji @SecondaryTable.

Zmodyfikowano również konstruktor przywracając go do poprzedniego stanu.

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

@Entity
@SecondaryTable(name = "ADDRESS_TBL")
public class Supplier
{
    @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;
    @OneToMany
    @JoinColumn(name = "SUPPLIED_BY")
    private Set<Product> products=new HashSet<>();

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

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

    public boolean alreadySupplies(Product product) { return products.contains(product); }

}
```

Standardowo stworzono obiekty wszystkich klas, w tym obiekty klasy Supplier.
Tym razem wszystkie parametry zostały podane bezpośrednio do konstruktora obiektu typu Supplier

```java
public static void main(String argv[])
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    Product prod1 = new Product( ProductName: "jabłko", UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "papryka", UnitsOnStock: 200);
    Product prod3 = new Product( ProductName: "ogórek", UnitsOnStock: 100);
    Supplier sup1 = new Supplier( CompanyName: "Lidl", street: "Sezamkowa", city: "Warszawa", zipCode: "40-308");
    Supplier sup2 = new Supplier( CompanyName: "Biedronka", street: "Miła", city: "Warszawa", zipCode: "30-561");
    Category cat1 = new Category( Name: "owoce");
    Category cat2 = new Category( Name: "warzywa");
    Invoice invoice1 = new Invoice( Quantity: 0);
    Invoice invoice2 = new Invoice( Quantity: 0);

    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup2);
    prod3.SetSupplier(sup2);
    prod1.SetCategory(cat1);
    prod2.SetCategory(cat2);
    prod3.SetCategory(cat2);

    invoice1.AddProductToSet(prod1);
    invoice1.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod1);
    invoice2.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod2);

    em.persist(prod1);
    em.persist(prod2);
    em.persist(prod3);

    em.persist(sup1);
    em.persist(sup2);

    em.persist(cat1);
    em.persist(cat2);

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

Zawartość tabeli ADDRESS_TBL po wykonaniu powyższego programu.

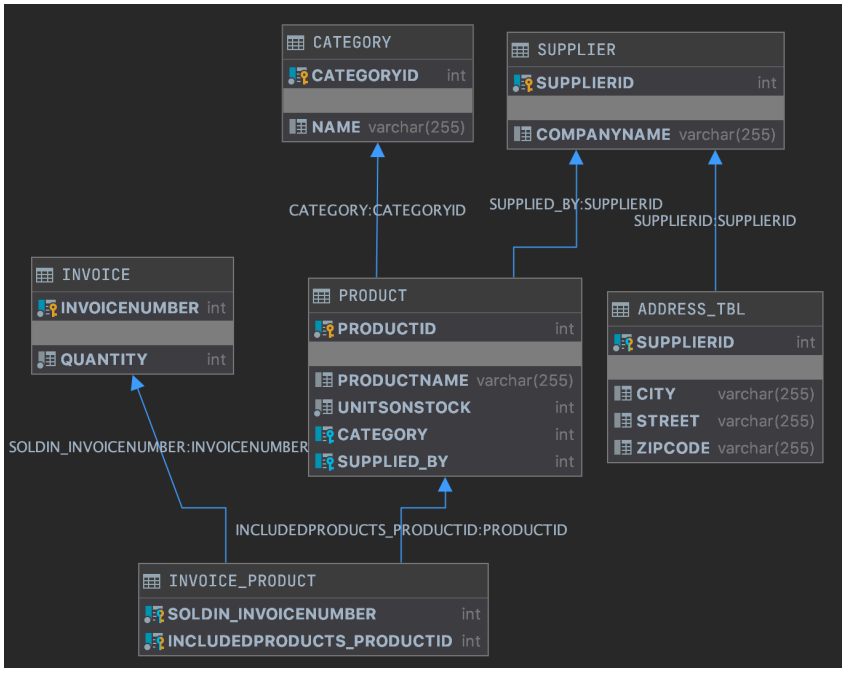
	CITY	STREET	ZIPCODE	SUPPLIERID
1	Warszawa	Sezamkowa	40-308	6
2	Warszawa	Miła	30-561	7

Zawartość tabeli SUPPLIER po wykonaniu powyższego programu.

	SUPPLIERID	COMPANYNAME
1	6	Lidl
2	7	Biedronka

Tak jak zamierzano, dane zmapowano do dwóch osobnych tabeli.

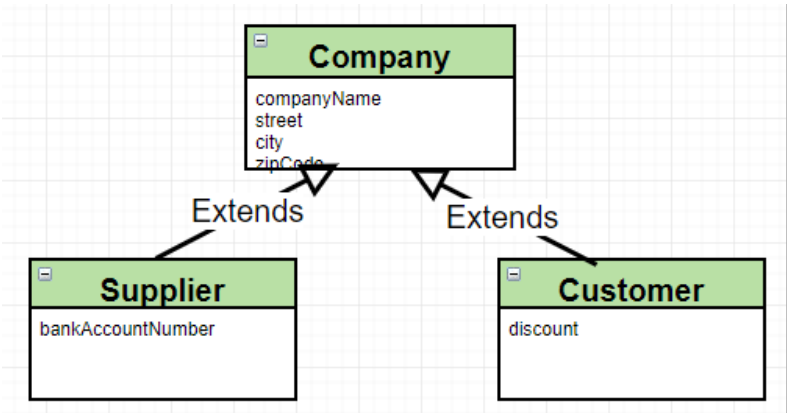
Obecny schemat bazy danych:



Skrypt generujący tabele SUPPLIER oraz ADDRESS_TBL

```
create table SUPPLIER
(
    SUPPLIERID  INTEGER not null
        constraint "SQL0000000041-daf74b2a-0171-d036-e262-000070f7e632"
            primary key,
    COMPANYNAME VARCHAR(255)
);

create table ADDRESS_TBL
(
    CITY       VARCHAR(255),
    STREET     VARCHAR(255),
    ZIPCODE    VARCHAR(255),
    SUPPLIERID INTEGER not null
        constraint "SQL0000000036-5cedcb0b-0171-d036-e262-000070f7e632"
            primary key
        constraint FKCP310M0H5HKQJOODXM6E44992
            references SUPPLIER
);
```

XIII. Dziedziczenie: Wprowadź do modelu następującą hierarchie:



a) SINGLE_TABLE

Stworzono klasę Company, po której będą dziedziczyć klasy Supplier oraz Customer.

Rodzajem mapowania dziedziczenia w tym przypadku jest SINGLE_TABLE.

```java
import javax.persistence.*;

@Entity
@SecondaryTable(name = "ADDRESS_TBL")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Company
{
    @Id
    @GeneratedValue (strategy = GenerationType.AUTO)
    private int CompanyID;
    private String companyName;
    @Column(table = "ADDRESS_TBL")
    private String street;
    @Column(table = "ADDRESS_TBL")
    private String city;
    @Column(table = "ADDRESS_TBL")
    private String zipCode;

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

Z klasy Supplier, dziedziczącej po Company usunięto wszystkie pola, które znajdują się obecnie w klasie Company, a dodano do nie pole bankAccount.

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

@Entity
public class Supplier extends Company
{
    private String bankAccount;
    @OneToMany
    @JoinColumn(name = "SUPPLIED_BY")
    private Set<Product> products=new HashSet<>();

    public Supplier(){};

    public Supplier(String bankAccount)
    {
        this.bankAccount = bankAccount;
    }

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

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

    public boolean alreadySupplies(Product product) { return products.contains(product); }

}
```

Utworzono klasę Customer również dziedziczącą po Company oraz umieszczono tu atrybut discount charakterystyczny dla tej klasy.

```java
import javax.persistence.Entity;

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

    public Customer(){}

    public Customer(double discount)
    {
        this.discount = discount;
    }

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

}
```

W hibconfig dodano mapowanie klasy Company oraz Customer

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="connection.url">jdbc:derby://localhost:1527/NowaraJPA</property>
        <property name="connection.driver_class">org.apache.derby.jdbc.ClientDriver</property>
        <property name="dialect">org.hibernate.dialect.DerbyTenSevenDialect</property>
        <property name="format_sql">true</property>
        <property name="show_sql">true</property>
        <property name="use_sql_comments">true</property>
        <property name="hibernate.hbm2ddl.auto">create</property>
        <mapping class="Product"/>
        <mapping class="Category"/>
        <mapping class="Invoice"/>
        <mapping class="Company"/>
        <mapping class="Supplier"/>
        <mapping class="Customer"/>
    </session-factory>
</hibernate-configuration>
```

Stworzono obiekty wszystkich klas

```java
public static  void main(String argv[])
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    Product prod1 = new Product( ProductName: "jabłko",  UnitsOnStock: 100);
    Product prod2 = new Product( ProductName: "papryka",  UnitsOnStock: 200);
    Product prod3 = new Product( ProductName: "ogórek",  UnitsOnStock: 100);
    Supplier sup1 = new Supplier( companyName: "Supplier1",  street: "Sezamkowa",  city: "Warszawa",  zipCode: "40-308",  bankAccount: "111222333444");
    Supplier sup2 = new Supplier( companyName: "Suppier2",  street: "Miła",  city: "Warszawa",  zipCode: "30-561",  bankAccount: "999888777666");
    Customer customer1 = new Customer( companyName: "Customer1",  street: "Guzikowa",  city: "Kraków",  zipCode: "30-805",  discount: 0.3 );
    Customer customer2 = new Customer( companyName: "Customer2",  street: "Podniebna",  city: "Tychy",  zipCode: "40-815",  discount: 0.5 );
    Category cat1 = new Category( Name: "owoce");
    Category cat2 = new Category( Name: "warzywa");
    Invoice invoice1 = new Invoice( Quantity: 0);
    Invoice invoice2 = new Invoice( Quantity: 0);

    prod1.SetSupplier(sup1);
    prod2.SetSupplier(sup2);
    prod3.SetSupplier(sup2);
    prod1.SetCategory(cat1);
    prod2.SetCategory(cat2);
    prod3.SetCategory(cat2);

    invoice1.AddProductToSet(prod1);
    invoice1.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod1);
    invoice2.AddProductToSet(prod3);
    invoice2.AddProductToSet(prod2);

    em.persist(sup1);
    em.persist(sup2);

    em.persist(customer1);
    em.persist(customer2);

    em.persist(prod1);
    em.persist(prod2);
    em.persist(prod3);

    em.persist(cat1);
    em.persist(cat2);
```

Zawartość tabeli COMPANY

	DTYPE	COMPANYID	COMPANYNAME	BANKACCOUNT	DISCOUNT
1	Supplier	1	Supplier1	111222333444	<null>
2	Supplier	2	Suppier2	999888777666	<null>
3	Customer	3	Customer1	<null>	0.3
4	Customer	4	Customer2	<null>	0.5

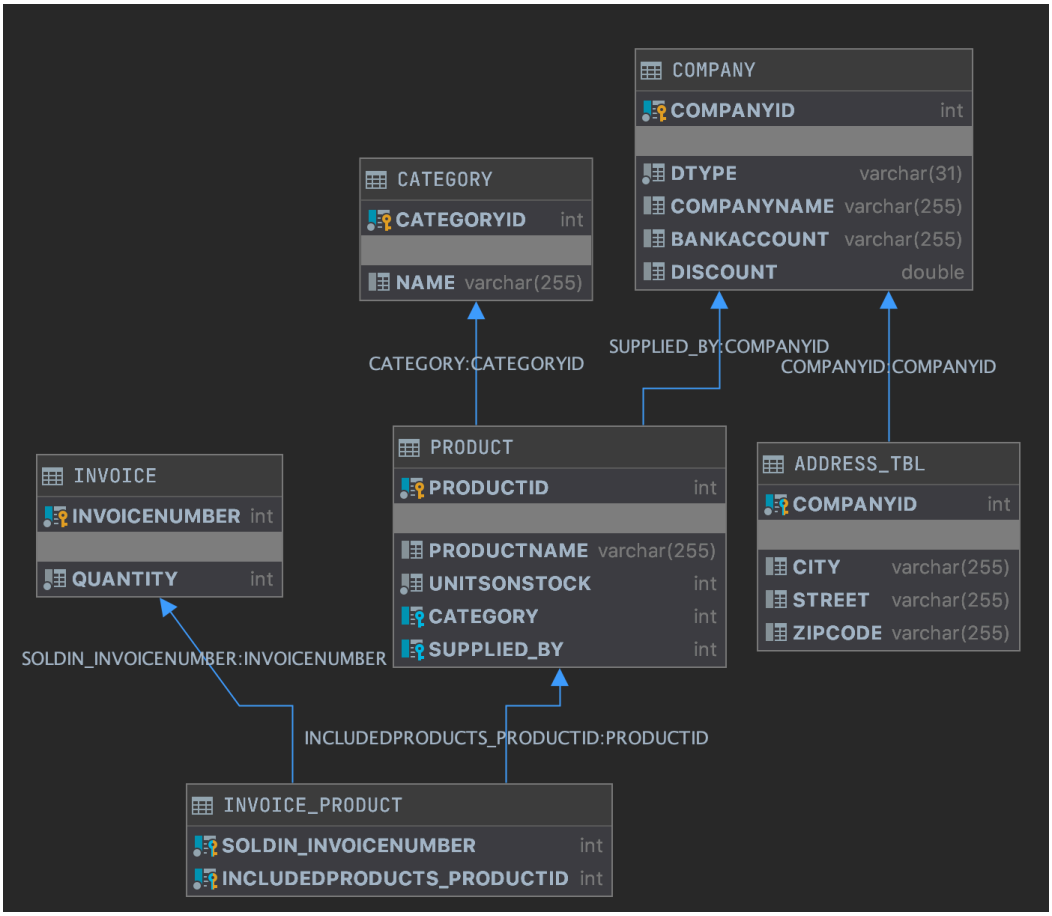
Zawartość tabeli ADDRESS_TBL

	CITY	STREET	ZIPCODE	COMPANYID
1	Warszawa	Sezamkowa	40-308	1
2	Warszawa	Miła	30-561	2
3	Kraków	Guzikowa	30-805	3
4	Tychy	Podniebna	40-815	4

Zawartość tabeli PRODUCT

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	5	jabłko	100	10	1
2	7	papryka	200	11	2
3	8	ogórek	100	11	2

Obecny schemat bazy danych, zawierający tabelę COMPANY przechowuje łączną zawartość klas Supplier oraz Customer



Część skryptu generującego bazę danych dotyczącą tabeli COMPANY oraz związanych z nią ADDRESS_TBL i PRODUCT

```
create table COMPANY
(
    DTYPE       VARCHAR(31) not null,
    COMPANYID   INTEGER     not null
        constraint "SQL0000000060-c5664e12-0171-d036-e262-000070f7e632"
            primary key,
    COMPANYNAME VARCHAR(255),
    BANKACCOUNT VARCHAR(255),
    DISCOUNT    DOUBLE
);

create table ADDRESS_TBL
(
    CITY      VARCHAR(255),
    STREET    VARCHAR(255),
    ZIPCODE   VARCHAR(255),
    COMPANYID INTEGER not null
        constraint "SQL0000000058-cfc28e05-0171-d036-e262-000070f7e632"
            primary key
        constraint FKPSQTLLLGRU9XXVWWOU2IRQF2E
            references COMPANY (COMPANYID)
);

create index "SQL0000000064-5a174e2a-0171-d036-e262-000070f7e632"
    on ADDRESS_TBL (COMPANYID);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000063-d4dd8e24-0171-d036-e262-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY (CATEGORYID),
    SUPPLIED_BY  INTEGER
        constraint FKORWD7TI86M4OHR0CRO5GFGLHG
            references COMPANY (COMPANYID)
);
```

Wypisanie nazw zarówno klientów jak i dostawców.

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

    Customer customer1 = em.find(Customer.class, o: 3 );
    System.out.println("Pobrano klienta o numerze ID 3: "+ customer1.getCompanyName());
    Customer customer2 = em.find(Customer.class, o: 4 );
    System.out.println("Pobrano klienta o numerze ID 4: "+ customer2.getCompanyName());
    Supplier supplier1 = em.find(Supplier.class,  o: 1);
    System.out.println("Pobrano dostawcę o numerze ID 1: "+ supplier1.getCompanyName());
    Supplier supplier2 = em.find(Supplier.class,  o: 2);
    System.out.println("Pobrano dostawcę o numerze ID 2: "+ supplier2.getCompanyName());
    etx.commit();
    em.close();
}
```

Rezultat wykonania się powyższego programu

```
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate:
    select
        customer0_.CompanyID as companyi2_2_0_,
        customer0_.companyName as companyn3_2_0_,
        customer0_1_.city as city1_0_0_,
        customer0_1_.street as street2_0_0_,
        customer0_1_.zipCode as zipcode3_0_0_,
        customer0_.discount as discount4_2_0_
    from
        Company customer0_
    left outer join
        ADDRESS_TBL customer0_1_
            on customer0_.CompanyID=customer0_1_.CompanyID
    where
        customer0_.CompanyID=?
        and customer0_.DTYPE='Customer'
Pobrano klienta o numerze ID 3: Customer1
Hibernate:
    select
        customer0_.CompanyID as companyi2_2_0_,
        customer0_.companyName as companyn3_2_0_,
        customer0_1_.city as city1_0_0_,
        customer0_1_.street as street2_0_0_,
        customer0_1_.zipCode as zipcode3_0_0_,
        customer0_.discount as discount4_2_0_
    from
        Company customer0_
    left outer join
        ADDRESS_TBL customer0_1_
            on customer0_.CompanyID=customer0_1_.CompanyID
    where
        customer0_.CompanyID=?
        and customer0_.DTYPE='Customer'
Pobrano klienta o numerze ID 4: Customer2
```

```
Hibernate:
    select
        supplier0_.CompanyID as companyi2_2_0_,
        supplier0_.companyName as companyn3_2_0_,
        supplier0_1_.city as city1_0_0_,
        supplier0_1_.street as street2_0_0_,
        supplier0_1_.zipCode as zipcode3_0_0_,
        supplier0_.bankAccount as bankacco5_2_0_
    from
        Company supplier0_
    left outer join
        ADDRESS_TBL supplier0_1_
            on supplier0_.CompanyID=supplier0_1_.CompanyID
    where
        supplier0_.CompanyID=?
        and supplier0_.DTYPE='Supplier'
Pobrano dostawcę o numerze ID 1: Supplier1
Hibernate:
    select
        supplier0_.CompanyID as companyi2_2_0_,
        supplier0_.companyName as companyn3_2_0_,
        supplier0_1_.city as city1_0_0_,
        supplier0_1_.street as street2_0_0_,
        supplier0_1_.zipCode as zipcode3_0_0_,
        supplier0_.bankAccount as bankacco5_2_0_
    from
        Company supplier0_
    left outer join
        ADDRESS_TBL supplier0_1_
            on supplier0_.CompanyID=supplier0_1_.CompanyID
    where
        supplier0_.CompanyID=?
        and supplier0_.DTYPE='Supplier'
Pobrano dostawcę o numerze ID 2: Suppier2
```

b) JOINED

W celu zmiany strategii dziedziczenia w klasie Company zmieniono wartość strategy w adnotacji @Inheritance na JOINED

```java
import javax.persistence.*;

@Entity
@SecondaryTable(name = "ADDRESS_TBL")
@Inheritance(strategy = InheritanceType.JOINED)
public class Company
{
    @Id
    @GeneratedValue (strategy = GenerationType.AUTO)
    private int CompanyID;
    private String companyName;
    @Column(table = "ADDRESS_TBL")
    private String street;
    @Column(table = "ADDRESS_TBL")
    private String city;
    @Column(table = "ADDRESS_TBL")
    private String zipCode;

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

    public String getCompanyName()
    {
        return this.companyName;
    }
}
```

Klasy Supplier oraz Customer pozostały bez zmian.

Ponownie uruchomiono program, w którym stworzono nowe obiekty (identyczny jak poprzednio).

Poniżej przedstawiono rezultat wywołania danego programu dla tabeli związanych z klasami dziedziczącymi po klasie Company.

Zawartość tabeli COMPANY

	COMPANYID	COMPANYNAME
1	1	Supplier1
2	2	Suppier2
3	3	Customer1
4	4	Customer2

Zawartość tabeli SUPPLIER

	BANKACCOUNT	COMPANYID
1	111222333444	1
2	999888777666	2

Zawartość tabeli CUSTOMER

	DISCOUNT	COMPANYID
1	0.3	3
2	0.5	4

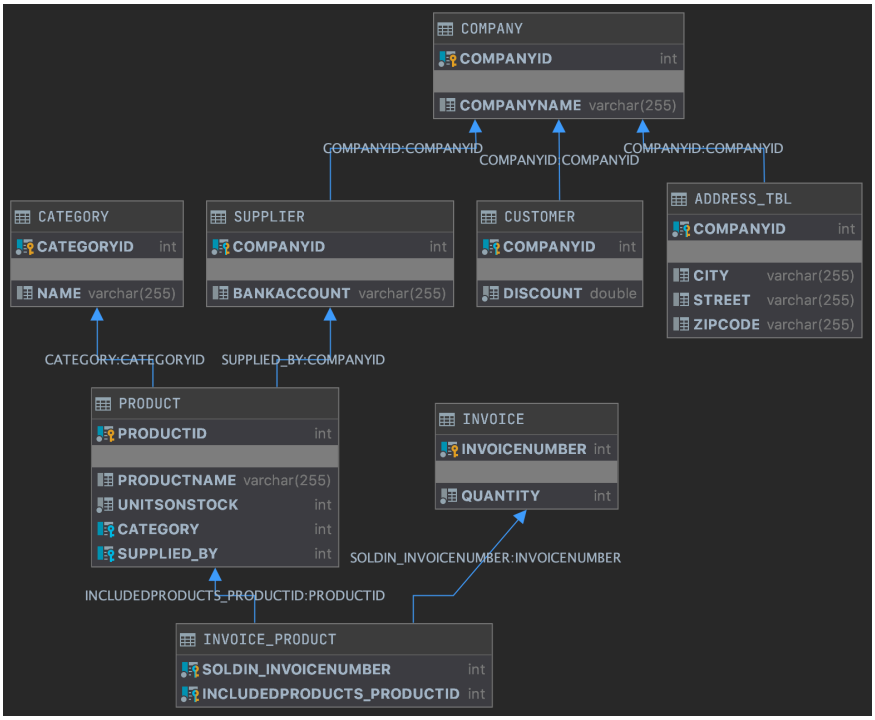
Zawartość tabeli ADDRESS_TBL

	CITY	STREET	ZIPCODE	COMPANYID
1	Warszawa	Sezamkowa	40-308	1
2	Warszawa	Miła	30-561	2
3	Kraków	Guzikowa	30-805	3
4	Tychy	Podniebna	40-815	4

Zawartość tabeli PRODUCT

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	5	jabłko	100	10	1
2	7	ogórek	100	11	2
3	9	papryka	200	11	2

Schemat bazy danych, tym razem w bazie danych znajduje się zarówno tabela COMPANY, jak i tabele SUPPLIER oraz CUSTOMER



Część skryptu generującego bazę danych dotyczącą tabel COMPANY, CUSTOMER, SUPPLIER oraz związanych z nimi ADDRESS_TBL i PRODUCT

```
create table COMPANY
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000071-400e1090-0171-d036-e262-000070f7e632"
            primary key,
    COMPANYNAME VARCHAR(255)
);

create table ADDRESS_TBL
(
    CITY      VARCHAR(255),
    STREET    VARCHAR(255),
    ZIPCODE   VARCHAR(255),
    COMPANYID INTEGER not null
        constraint "SQL0000000069-b250d083-0171-d036-e262-000070f7e632"
            primary key
        constraint FKPSQTLLLGRU9XXVWWOU2IRQF2E
            references COMPANY
);

create index "SQL0000000077-4e15d0b6-0171-d036-e262-000070f7e632"
    on ADDRESS_TBL (COMPANYID);

create table CUSTOMER
(
    DISCOUNT  DOUBLE  not null,
    COMPANYID INTEGER not null
        constraint "SQL0000000072-450f5097-0171-d036-e262-000070f7e632"
            primary key
        constraint FKO6BOYET0XSRWJCUP3LQK2RC7A
            references COMPANY
);
```

```
create table SUPPLIER
(
    BANKACCOUNT VARCHAR(255),
    COMPANYID   INTEGER not null
        constraint "SQL0000000076-c50a10b0-0171-d036-e262-000070f7e632"
            primary key
        constraint FK8CHS3V21RG6V4NCJWMLKNN8H1
            references COMPANY
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000075-afdd10a9-0171-d036-e262-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);
```

Wypisanie nazw zarówno klientów jak i dostawców.

```java
public static  void main(String argv[])
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    Company company1 = em.find(Customer.class, o: 3 );
    System.out.println("Pobrano klienta o numerze ID 3: "+ company1.getCompanyName());
    Company company2 = em.find(Customer.class, o: 4 );
    System.out.println("Pobrano klienta o numerze ID 4: "+ company2.getCompanyName());
    Company company3 = em.find(Supplier.class,  o: 1);
    System.out.println("Pobrano dostawcę o numerze ID 1: "+ company3.getCompanyName());
    Company company4 = em.find(Supplier.class,  o: 2);
    System.out.println("Pobrano dostawcę o numerze ID 2: "+ company4.getCompanyName());

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

Rezultat wykonania się powyższego programu

```
Hibernate:
    select
        customer0_.CompanyID as companyi1_2_0_,
        customer0_1_.companyName as companyn2_2_0_,
        customer0_2_.city as city1_0_0_,
        customer0_2_.street as street2_0_0_,
        customer0_2_.zipCode as zipcode3_0_0_,
        customer0_.discount as discount1_3_0_
    from
        Customer customer0_
    inner join
        Company customer0_1_
            on customer0_.CompanyID=customer0_1_.CompanyID
    left outer join
        ADDRESS_TBL customer0_2_
            on customer0_.CompanyID=customer0_2_.CompanyID
    where
        customer0_.CompanyID=?
Pobrano klienta o numerze ID 3: Customer1
Hibernate:
    select
        customer0_.CompanyID as companyi1_2_0_,
        customer0_1_.companyName as companyn2_2_0_,
        customer0_2_.city as city1_0_0_,
        customer0_2_.street as street2_0_0_,
        customer0_2_.zipCode as zipcode3_0_0_,
        customer0_.discount as discount1_3_0_
    from
        Customer customer0_
    inner join
        Company customer0_1_
            on customer0_.CompanyID=customer0_1_.CompanyID
    left outer join
        ADDRESS_TBL customer0_2_
            on customer0_.CompanyID=customer0_2_.CompanyID
    where
        customer0_.CompanyID=?
Pobrano klienta o numerze ID 4: Customer2
```

```
Hibernate:
    select
        supplier0_.CompanyID as companyi1_2_0_,
        supplier0_1_.companyName as companyn2_2_0_,
        supplier0_2_.city as city1_0_0_,
        supplier0_2_.street as street2_0_0_,
        supplier0_2_.zipCode as zipcode3_0_0_,
        supplier0_.bankAccount as bankacco1_7_0_
    from
        Supplier supplier0_
    inner join
        Company supplier0_1_
            on supplier0_.CompanyID=supplier0_1_.CompanyID
    left outer join
        ADDRESS_TBL supplier0_2_
            on supplier0_.CompanyID=supplier0_2_.CompanyID
    where
        supplier0_.CompanyID=?
Pobrano dostawcę o numerze ID 1: Supplier1
Hibernate:
    select
        supplier0_.CompanyID as companyi1_2_0_,
        supplier0_1_.companyName as companyn2_2_0_,
        supplier0_2_.city as city1_0_0_,
        supplier0_2_.street as street2_0_0_,
        supplier0_2_.zipCode as zipcode3_0_0_,
        supplier0_.bankAccount as bankacco1_7_0_
    from
        Supplier supplier0_
    inner join
        Company supplier0_1_
            on supplier0_.CompanyID=supplier0_1_.CompanyID
    left outer join
        ADDRESS_TBL supplier0_2_
            on supplier0_.CompanyID=supplier0_2_.CompanyID
    where
        supplier0_.CompanyID=?
Pobrano dostawcę o numerze ID 2: Suppier2
```

c) TABLE_PER_CLASS

W celu zmiany strategii dziedziczenia w klasie Comapny zmieniono wartość strategy w adnotacji @Inheritance na TABLE_PER_CLASS

Usunięto też adnotację @SecondaryTable i @Column

```java
import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Company
{
    @Id
    @GeneratedValue (strategy = GenerationType.AUTO)
    private int CompanyID;
    private String companyName;
    private String street;
    private String city;
    private String zipCode;

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

    public String getCompanyName()
    {
        return this.companyName;
    }
}
```

Klasy Supplier oraz Customer pozostały bez zmian.

Ponownie uruchomiono program, w którym stworzono nowe obiekty (identyczny jak poprzednio).

Poniżej przedstawiono rezultat wywołania danego programu dla tabeli związanych z klasami dziedziczącymi po klasie Company.

Zawartość tabeli CUSTOMER

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	DISCOUNT
1	3	Kraków	Customer1	Guzikowa	30-805	0.3
2	4	Tychy	Customer2	Podniebna	40-815	0.5

Zawartość tabeli SUPPLIER

	COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE	BANKACCOUNT
1	1	Warszawa	Supplier1	Sezamkowa	40-308	111222333444
2	2	Warszawa	Suppier2	Miła	30-561	999888777666

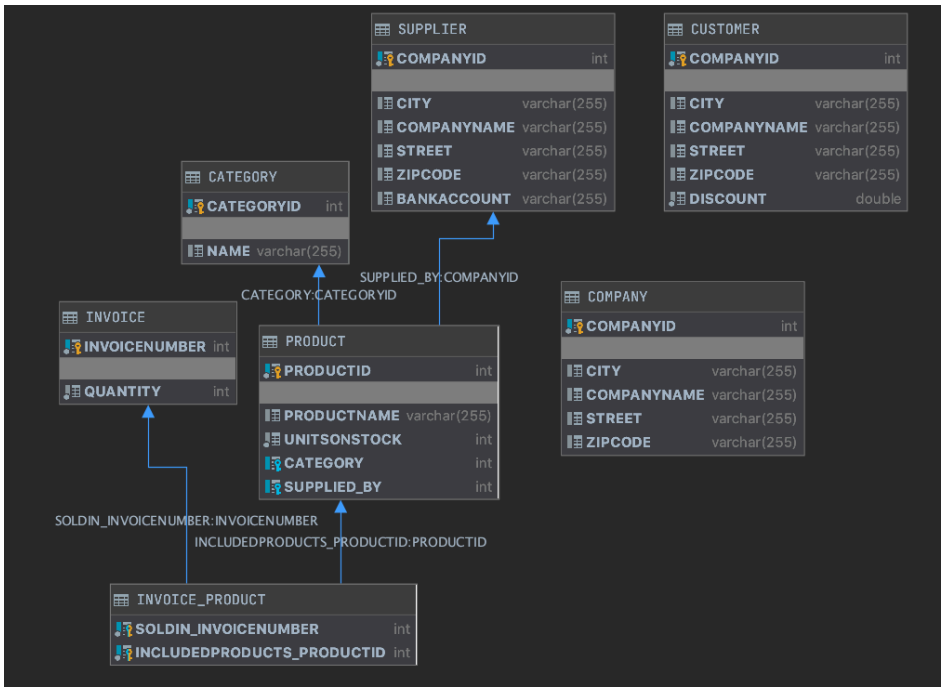
Pusta tabela COMPANY

COMPANYID	CITY	COMPANYNAME	STREET	ZIPCODE

Zawartość tabeli PRODUCT

	PRODUCTID	PRODUCTNAME	UNITSONSTOCK	CATEGORY	SUPPLIED_BY
1	5	jabłko	100	10	1
2	7	ogórek	100	11	2
3	9	papryka	200	11	2

Schemat bazy danych:



Część skryptu generującego bazę danych dotyczącą tabel COMPANY, CUSTOMER, SUPPLIER oraz związaną z nimi tabelą PRODUCT

```
create table COMPANY
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000119-7a3ad96e-0171-d036-e262-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255),
    ZIPCODE     VARCHAR(255)
);

create table CUSTOMER
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000120-50639974-0171-d036-e262-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255),
    ZIPCODE     VARCHAR(255),
    DISCOUNT    DOUBLE  not null
);
```

```
create table SUPPLIER
(
    COMPANYID   INTEGER not null
        constraint "SQL0000000124-2960998c-0171-d036-e262-000070f7e632"
            primary key,
    CITY        VARCHAR(255),
    COMPANYNAME VARCHAR(255),
    STREET      VARCHAR(255),
    ZIPCODE     VARCHAR(255),
    BANKACCOUNT VARCHAR(255)
);

create table PRODUCT
(
    PRODUCTID    INTEGER not null
        constraint "SQL0000000123-5313d986-0171-d036-e262-000070f7e632"
            primary key,
    PRODUCTNAME  VARCHAR(255),
    UNITSONSTOCK INTEGER not null,
    CATEGORY     INTEGER
        constraint FKEDNTEOGHHMGHKELRH3HBU75LP
            references CATEGORY,
    SUPPLIED_BY  INTEGER
        constraint FKRKVCU2QJUUMNU5IHG9CWRBTTN
            references SUPPLIER
);
```

Wypisanie nazw zarówno klientów jak i dostawców.

```
public static  void main(String argv[])
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "derby");
    EntityManager em = emf.createEntityManager();
    EntityTransaction etx = em.getTransaction();
    etx.begin();
    Company company1 = em.find(Customer.class, o: 3 );
    System.out.println("Pobrano klienta o numerze ID 3: "+ company1.getCompanyName());
    Company company2 = em.find(Customer.class, o: 4 );
    System.out.println("Pobrano klienta o numerze ID 4: "+ company2.getCompanyName());
    Company company3 = em.find(Supplier.class,  o: 1);
    System.out.println("Pobrano dostawcę o numerze ID 1: "+ company3.getCompanyName());
    Company company4 = em.find(Supplier.class,  o: 2);
    System.out.println("Pobrano dostawcę o numerze ID 2: "+ company4.getCompanyName());

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

Rezultat wykonania się powyższego programu

```
Hibernate:
    select
        customer0_.CompanyID as companyi1_1_0_,
        customer0_.city as city2_1_0_,
        customer0_.companyName as companyn3_1_0_,
        customer0_.street as street4_1_0_,
        customer0_.zipCode as zipcode5_1_0_,
        customer0_.discount as discount1_2_0_
    from
        Customer customer0_
    where
        customer0_.CompanyID=?
Pobrano klienta o numerze ID 3: Customer1
Hibernate:
    select
        customer0_.CompanyID as companyi1_1_0_,
        customer0_.city as city2_1_0_,
        customer0_.companyName as companyn3_1_0_,
        customer0_.street as street4_1_0_,
        customer0_.zipCode as zipcode5_1_0_,
        customer0_.discount as discount1_2_0_
    from
        Customer customer0_
    where
        customer0_.CompanyID=?
Pobrano klienta o numerze ID 4: Customer2
```

```
Hibernate:
    select
        supplier0_.CompanyID as companyi1_1_0_,
        supplier0_.city as city2_1_0_,
        supplier0_.companyName as companyn3_1_0_,
        supplier0_.street as street4_1_0_,
        supplier0_.zipCode as zipcode5_1_0_,
        supplier0_.bankAccount as bankacco1_6_0_
    from
        Supplier supplier0_
    where
        supplier0_.CompanyID=?
Pobrano dostawcę o numerze ID 1: Supplier1
Hibernate:
    select
        supplier0_.CompanyID as companyi1_1_0_,
        supplier0_.city as city2_1_0_,
        supplier0_.companyName as companyn3_1_0_,
        supplier0_.street as street4_1_0_,
        supplier0_.zipCode as zipcode5_1_0_,
        supplier0_.bankAccount as bankacco1_6_0_
    from
        Supplier supplier0_
    where
        supplier0_.CompanyID=?
Pobrano dostawcę o numerze ID 2: Suppier2
```