Hibernate

Relacyjne bazy danych, a świat obiektowy

- Relacyjne bazy danych są przetestowane pod względem wydajności
- RDB mogą przetrzymywać ogromne ilości danych
- Sortowanie, wyszukiwanie, grupowanie
- Integralność, constrainty
- Brak polimorfizmu
- Brak dziedziczenia
- Paradygmat programowania obiektowego odzwierciedla rzeczywistość

ORM

Mapowanie Obiektowo-Relacyjne
Object-Relational Mapping

ORM

Sposób odwzorowania obiektowej architektury systemu informatycznego na bazę danych (lub inny element systemu) o relacyjnym charakterze.

Implementacja takiego odwzorowania stosowana jest m.in. w przypadku, gdy tworzony system oparty jest na podejściu obiektowym, a system bazy danych operuje na relacjach.

JPA

Java Persistence API

JPA

- oficjalny standard mapowania obiektowo-relacyjnego firmy Sun Microsystems / Oracle dla języka programowania Java
- zbiór interfejsów adnotacji i klas do trwałego przechowywania danych w bazie danych
- definiuje język zapytań JPQL
- przykładowe implementacje: **Hibernate**, TopLink, Kodo
- najnowsza wersja: JPA 2.1
- http://download.oracle.com/otn-pub/jcp/persistence-2_1-fr-eval-spec/JavaPersistence.pdf

JPA, ORM - podstawowe korzyści

- nie trzeba ręcznie pisać instrukcji SQL
- umożliwia posługiwanie się paradygmatem obiektowym przy współpracy z relacyjną bazą danych
- umożliwia uniezależnienie się od konkretnej bazy danych
- zapewnia dodatkowe mechanizmy, np. cache

Hibernate

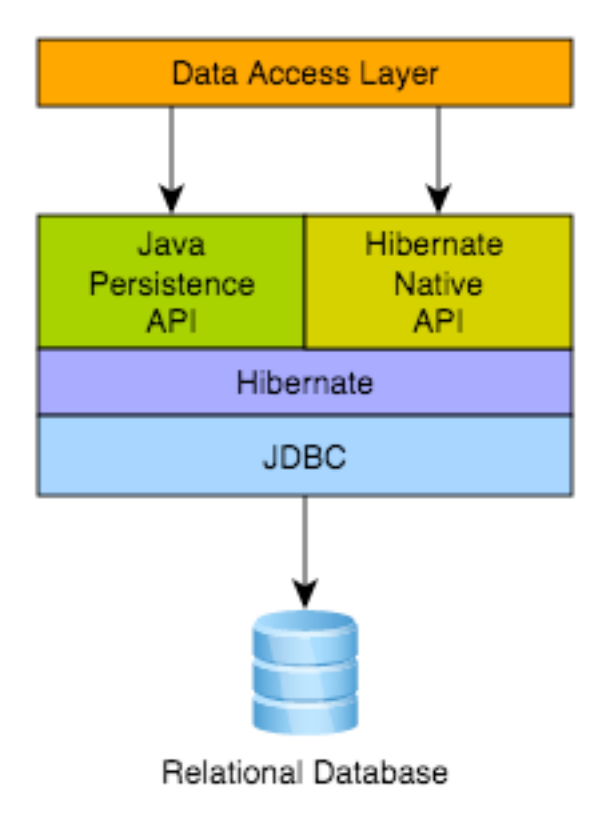
Hibernate

Framework do realizacji warstwy dostępu do danych (ang. persistence layer). Zapewnia przede wszystkim translację danych pomiędzy relacyjną bazą danych a światem obiektowym (ang. O/R mapping). Opiera się na wykorzystaniu opisu struktury danych za pomocą języka XML lub adnotacji, dzięki czemu można "rzutować" obiekty stosowane w obiektowych językach programowania bezpośrednio na istniejące tabele bazy danych. Dodatkowo Hibernate zwiększa wydajność operacji na bazie danych dzięki buforowaniu i minimalizacji liczby przesyłanych zapytań. Jest to projekt rozwijany jako open source. Głównym inicjatorem i liderem projektu jest Gavin King.

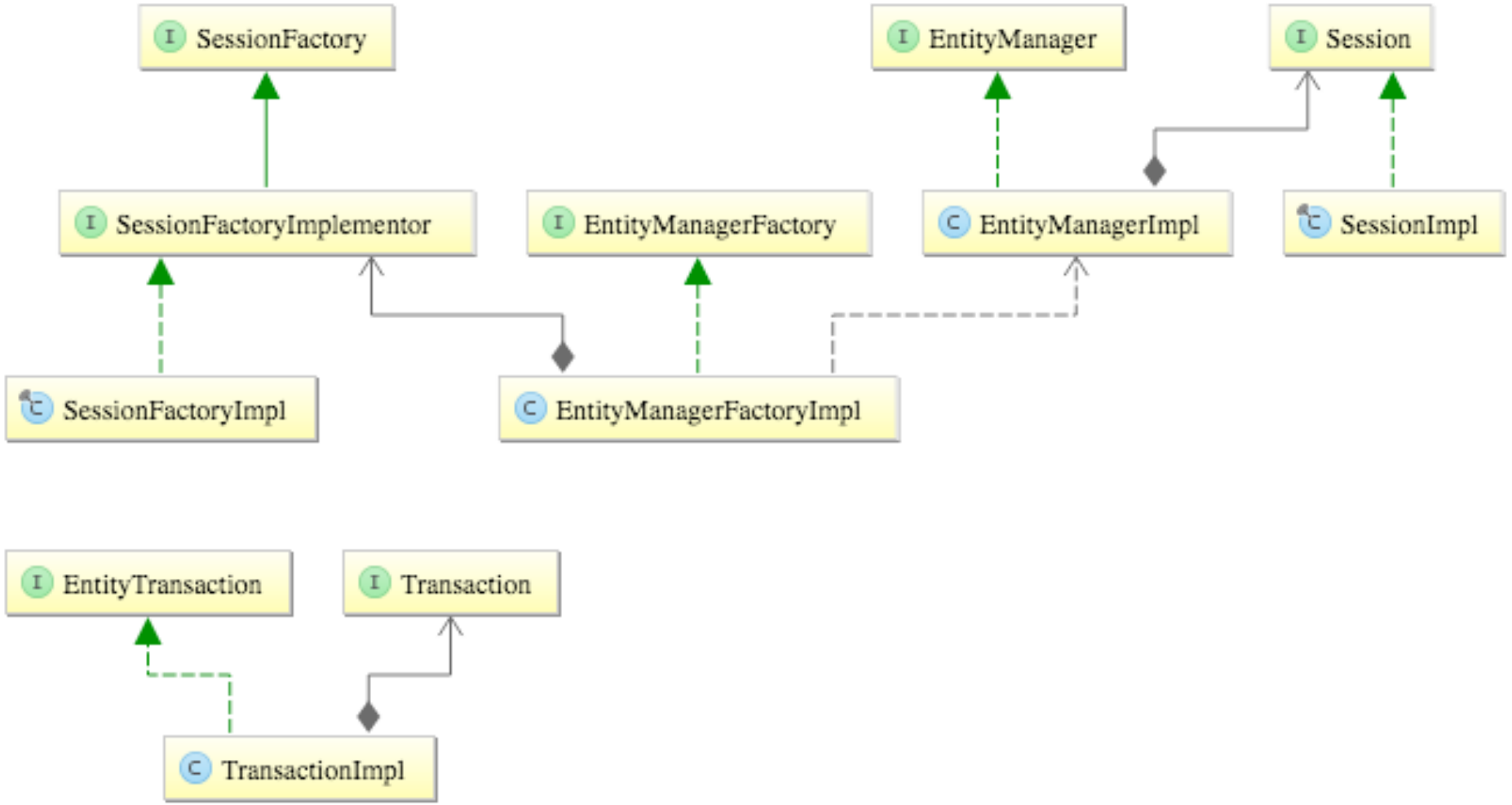
Hibernate - SessionFactory, Session i Transaction

- **SessionFactory** - jest tworzony na podstawie konfiguracji. Służy do produkcji obiektów typu **Session**. Koszt jego utworzenia jest wysoki, dlatego aplikacje zazwyczaj posiadają tylko jedną instancję obiektu tego typu. Jest przystosowany do wykorzystywania w środowisku wielowątkowym.
- **Session** - jest obiektem reprezentującym połączenie z bazą danych. Cała komunikacja jest wykonywana za jego pośrednictwem. obiekty tej klasy nie są przystosowane do wielowątkowości, więc każdy wątek w aplikacji powinien mieć własną instancję.
- **Transaction** - pozwala na zarządzanie transakcjami.

Hibernate - Architektura



Hibernate - Architektura



Hibernate - Encja

Podstawowe pojęcie w przypadku mapowania relacyjno – obiektowego oraz persystencji.

Reprezentacja wyobrażonego lub rzeczywistego obiektu (grupy obiektów) stosowana przy modelowaniu danych.

Encja zawiera w sobie cechy (atrybuty) obiektu, który tworzy.

W przypadku relacyjnych baz danych, encja jest zazwyczaj utożsamiana z wierszem w bazie danych.

Encje definiujemy za pomocą adnotacji lub XML.

Hibernate - Encja - POJO

Encja to klasa POJO, która reprezentuje tabelę w bazie danych. Klasa encji powinna spełniać następujące wymagania:

- jeżeli używamy adnotacji, to powinna mieć adnotację **@Entity**
- powinna posiadać jednoznaczny identyfikator, **@Id**
- klasa powinna posiadać bezargumentowy konstruktor
- klasa nie powinna być finalna
- pola klasy powinny mieć dostęp `private`, `protected` bądź package
- aby przechowywać encje w kolekcjach warto nadpisać metody `hashCode()` i `equals()`
- jeśli chcemy przesyłać encję „przez sieć”, to klasa powinna implementować interfejs `Serializable`

Hibernate - konfiguracja - pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.43</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.11.Final</version>
    </dependency>
    <dependency>
        <groupId>org.javassist</groupId>
        <artifactId>javassist</artifactId>
        <version>3.18.0-GA</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.6.1</version>
    </dependency>
</dependencies>
```

Hibernate - konfiguracja - hibernate.cfg.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQL57Dialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost/hibernate?useSSL=false</property>
        <property name="hibernate.connection.username">user_hibernate</property>
        <property name="hibernate.connection.password">hibernate01</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.use_sql_comments">true</property>

        <mapping class="domain.Employee"/>
        <mapping class="domain.Phone"/>
    </session-factory>
</hibernate-configuration>
```

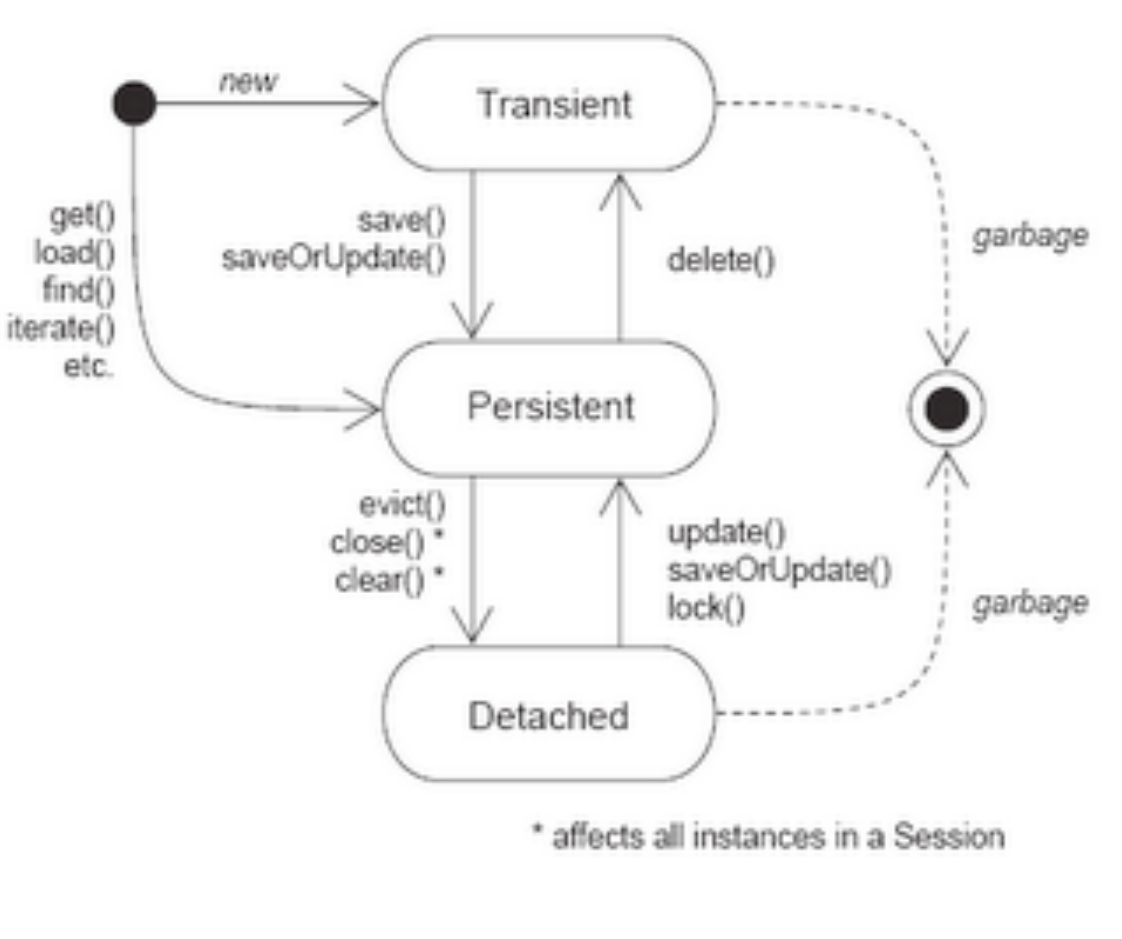
Hibernate - HibernateUtil - przykład

```java
public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder()
                    .configure("hibernate.cfg.xml")
                    .build();
            Metadata metaData = new MetadataSources(standardRegistry)
                    .getMetadataBuilder()
                    .build();
            sessionFactory = metaData.getSessionFactoryBuilder().build();
        } catch (Throwable th) {
            System.err.println("Initial SessionFactory creation failed" + th);
            throw new ExceptionInInitializerError(th);

        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Hibernate - Encja - przykład

```java
@Entity
@Table(name = "employee")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

    @Column(name = "name")
    private String name;

    public Employee() {
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

Hibernate - Encja - cykl życia



Hibernate - Encja - zdarzenia zwrotne i klasy nasłuchujące

- @PrePersist
- @PostPersist
- @PostLoad
- @PreUpdate
- @PostUpdate
- @PreRemove
- @PostRemove

```java
@Entity
@EntityListeners(LastUpdateListener.class)
public class Cat {
    @Id
    private Long id;
    private String name;
    private Date dateOfBirth;
    @Transient
    private long age;
    private Date lastUpdate;

    public void setLastUpdate(Date lastUpdate) {...}

    @PostLoad
    public void calculateAge() {
        age = ChronoUnit.YEARS.between(LocalDateTime.ofInstant(
                Instant.ofEpochMilli(dateOfBirth.getTime()), ZoneOffset.UTC),
                LocalDateTime.now()
        );
    }
}
class LastUpdateListener {
    @PreUpdate
    @PrePersist
    public void setLastUpdate(Cat c) { c.setLastUpdate(new Date()); }
}
```

UWAGA! Działa tylko z EntityManagerem, aby zintegrować z Hibernate Session

https://n1njahacks.wordpress.com/2016/10/07/jpa-callbacks-with-hibernates-sessionfactory-and-no-entitymanager/

https://stackoverflow.com/questions/28340507/hibernate-event-listeners-for-jpa-callbacks

Hibernate - Encja - zapis

```
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();

try {
    Employee employee = new Employee("Janusz", "tajne haslo", 33);
    session.persist(employee);

    transaction.commit();
} catch (Exception e) {
    transaction.rollback();
} finally {
    session.close();
    sessionFactory.close();
}
```

Hibernate - Encja - pobranie i edycja

```
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();

try {
    Employee employee = session.find(Employee.class, 1);
    employee.setName("Heniek");

    transaction.commit();
} catch (Exception e) {
    transaction.rollback();
} finally {
    session.close();
    sessionFactory.close();
}
```

Hibernate - Encja - pobranie i usunięcie

```
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();

try {
    Employee employee = new Employee("Janusz", "tajne haslo", 33);
    session.remove(employee);

    transaction.commit();
} catch (Exception e) {
    transaction.rollback();
} finally {
    session.close();
    sessionFactory.close();
}
```

Hibernate - Adnotacje

- `@Table, @SecondaryTable`
- `@Column`
- `@Id`
- `@GeneratedValue`
- `@Embedded, @Embeddable`
- `@IdClass, @EmbeddedId`
- `@Transient`
- `@Temporal`
- `@Lob`
- `@Enumerated`

https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/

Hibernate - Encja vs tabela - 1 klasa 2 tabele

```sql
create table CUSTOMER_TABLE (
  CUST_ID integer primary key not null,
  FIRST_NAME varchar(20) not null,
  LAST_NAME varchar(50) not null
);

create table ADDRESS_TABLE (
  ADDRESS_ID integer primary key not null,
  STREET varchar(255) not null,
  CITY varchar(255) not null,
  STATE varchar(255) not null
);
```

```java
@Entity
@Table(name="CUSTOMER_TABLE")
@SecondaryTable(name="ADDRESS_TABLE",
        pkJoinColumns={
                @PrimaryKeyJoinColumn(name="ADDRESS_ID")})
public class Customer implements java.io.Serializable {
    private long id;
    private String firstName;
    private String lastName;

    @Column(name = "STREET", table = "ADDRESS_TABLE")
    private String street;
    @Column(name = "CITY", table = "ADDRESS_TABLE")
    private String city;
    @Column(name = "STATE", table = "ADDRESS_TABLE")
    private String state;
}
```

Hibernate - Encja vs tabela - 2 klasy 1 tabela

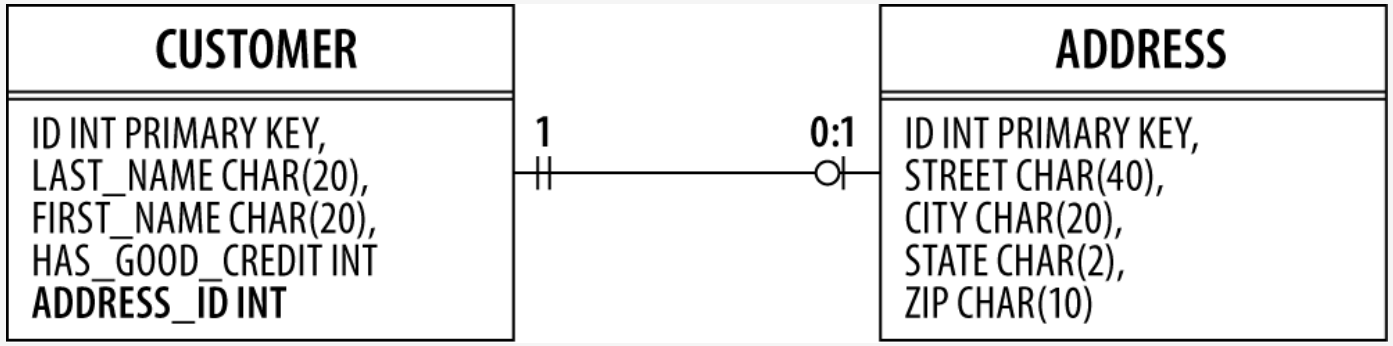
```java
@Embeddable
public class Address implements java.io.Serializable {
    private String street;
    private String city;
    private String state;
```

```java
@Entity
@Table(name = "CUSTOMER_TABLE")
public class Customer implements java.io.Serializable {

    private long id;
    private String firstName;
    private String lastName;

    @Embedded
    @AttributeOverrides({
            @AttributeOverride(name="street", column=@Column(name="CUST_STREET")),
            @AttributeOverride(name="city", column=@Column(name="CUST_CITY")),
            @AttributeOverride(name="state", column=@Column(name="CUST_STATE"))
    })
    private Address address;
```

Hibernate - Rodzaje relacji

- `@OneToOne`
  - `@PrimaryKeyJoinColumn`
  - `@JoinColumn`
  - `@JoinTable`
- `@OneToMany / @ManyToOne`
  - `@JoinColumn`
  - `@JoinTable`
- `@ManyToMany`
  - `@JoinTable`
- jedno- oraz dwukierunkowe
- Uwaga! Właściciel relacji - owning side

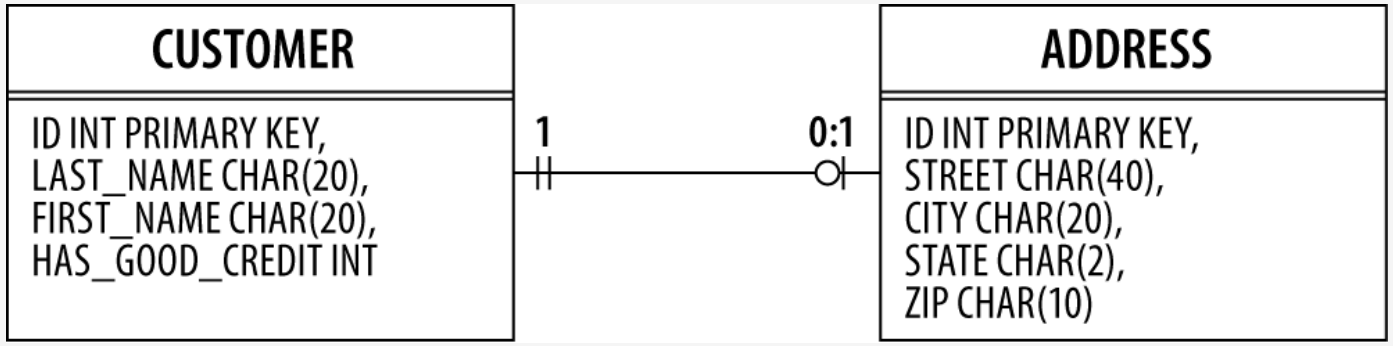
Hibernate - Rodzaje relacji - @OneToOne




```java
@Entity
public class Customer implements Serializable {

    @OneToOne
    @JoinColumn(name = "ADDRESS_ID")
    private Address address;

    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
}
```

```java
@Entity
public class Address implements Serializable {
    @Id
    private long id;
    private String street;
    private String city;
    private String state;
    private String zip;

    public Address() {
    }
}
```

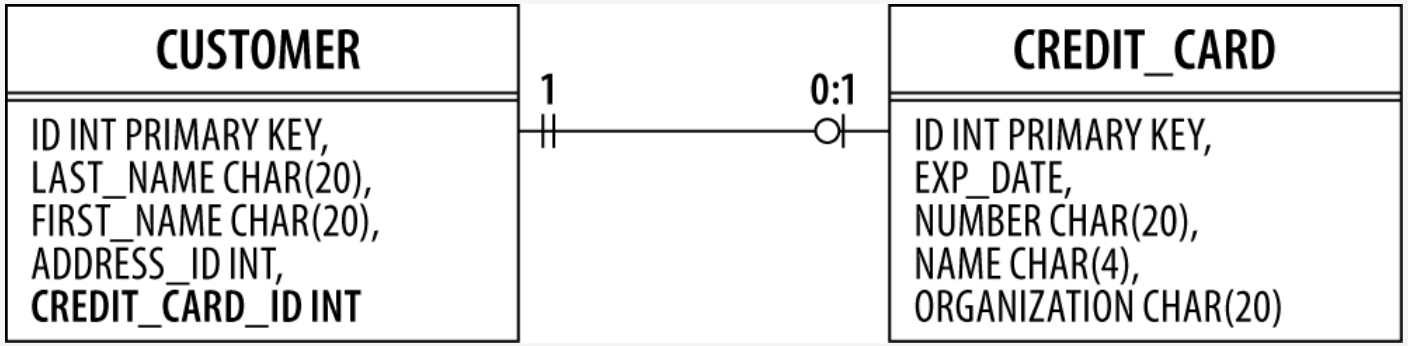
Hibernate - Rodzaje relacji - @OneToOne




```java
@Entity
public class Customer implements Serializable {

    @OneToOne
    @PrimaryKeyJoinColumn
    private Address address;

    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
}
```

```java
@Entity
public class Address implements Serializable {
    @Id
    private long id;
    private String street;
    private String city;
    private String state;
    private String zip;

    public Address() {
    }
}
```

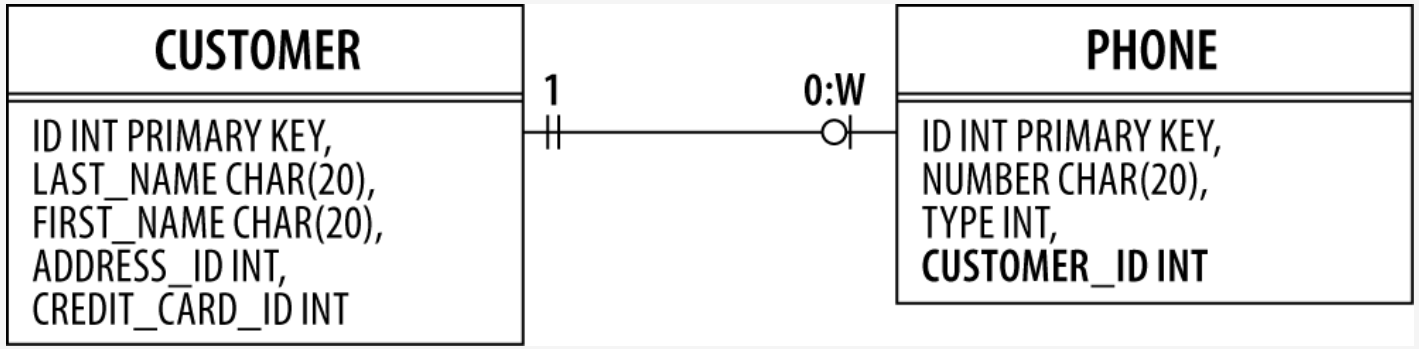
Hibernate - Rodzaje relacji - @OneToOne




```java
@Entity
public class Customer implements Serializable {

    @OneToOne
    @JoinColumn(name="CREDIT_CARD_ID")
    private CreditCard creditCard;

    public CreditCard getCreditCard() {
        return creditCard;
    }
    public void setCreditCard(CreditCard card) {
        this.creditCard = card;
    }
}
```

```java
@Entity
public class CreditCard implements Serializable {
    private int id;

    @OneToOne(mappedBy="creditCard")
    private Customer customer;

    public Customer getCustomer() {
        return this.customer;
    }
    public void setCustomer(Customer customer) {
        this.customer = customer;
    }
}
```

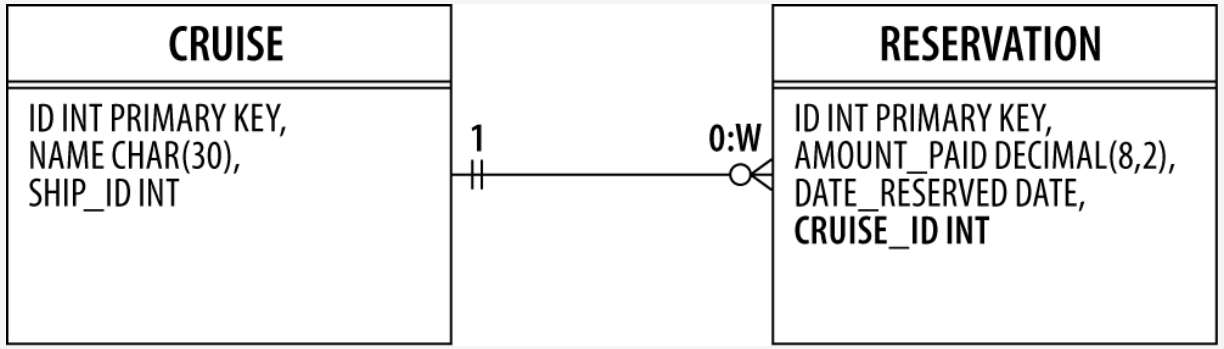
Hibernate - Rodzaje relacji - @OneToMany




```java
@Entity
public class Customer implements Serializable {

    @OneToMany
    @JoinColumn(name="CUSTOMER_ID")
    private Collection<Phone> phoneNumbers = new ArrayList<>();

    public Collection<Phone> getPhoneNumbers() {
        return phoneNumbers;
    }
    public void setPhoneNumbers(Collection<Phone> phones) {
        this.phoneNumbers = phones;
    }
}
```

```java
@Entity
public class Phone implements Serializable {
    @Id
    private int id;
    private String number;
    private int type;

}
```

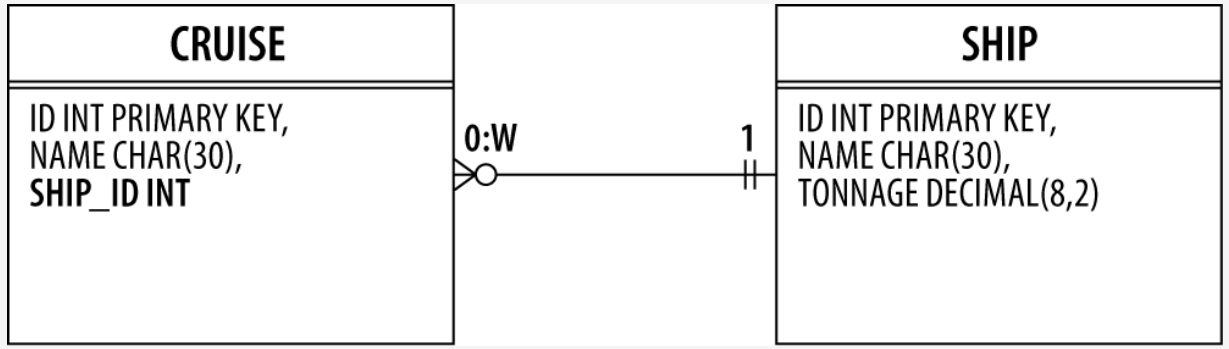
Hibernate - Rodzaje relacji - @OneToMany




```java
@Entity
public class Cruise implements Serializable {
    @Id
    private int id;

    @OneToMany(mappedBy="cruise")
    private Collection<Reservation> reservations = new ArrayList<>();
}
```

```java
@Entity
public class Reservation implements Serializable {
    @Id
    private int id;
    private float amountPaid;
    private Date date;
    @ManyToOne
    @JoinColumn(name = "CRUISE_ID")
    private Cruise cruise;
}
```

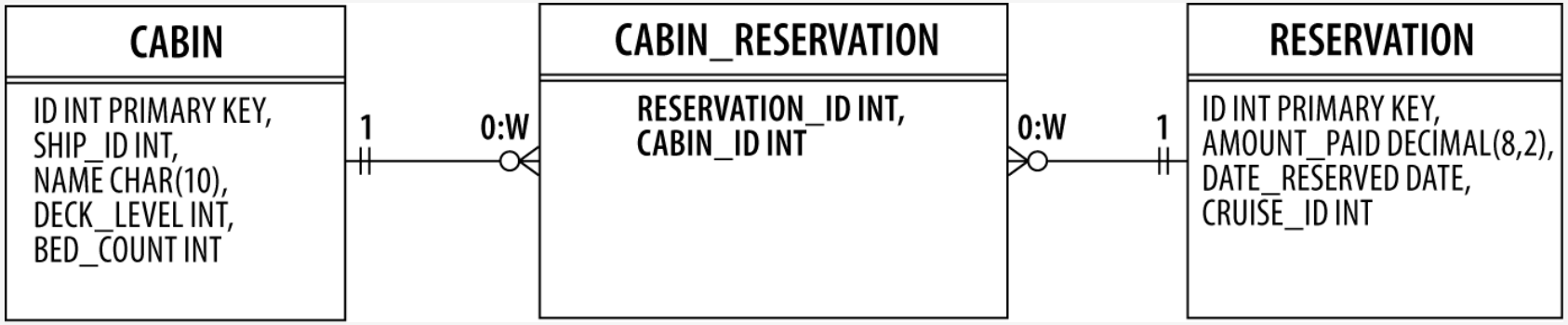
Hibernate - Rodzaje relacji - @ManyToOne




```java
@Entity
public class Cruise implements Serializable {
    @Id
    private int id;
    private String name;
    @ManyToOne
    @JoinColumn(name = "SHIP_ID")
    private Ship ship;

    public Ship getShip() { return ship; }

    public void setShip(Ship ship) { this.ship = ship; }
}
```

```java
@Entity
public class Ship implements Serializable {
    @Id
    private int id;
    private String name;
    private double tonnage;

}
```

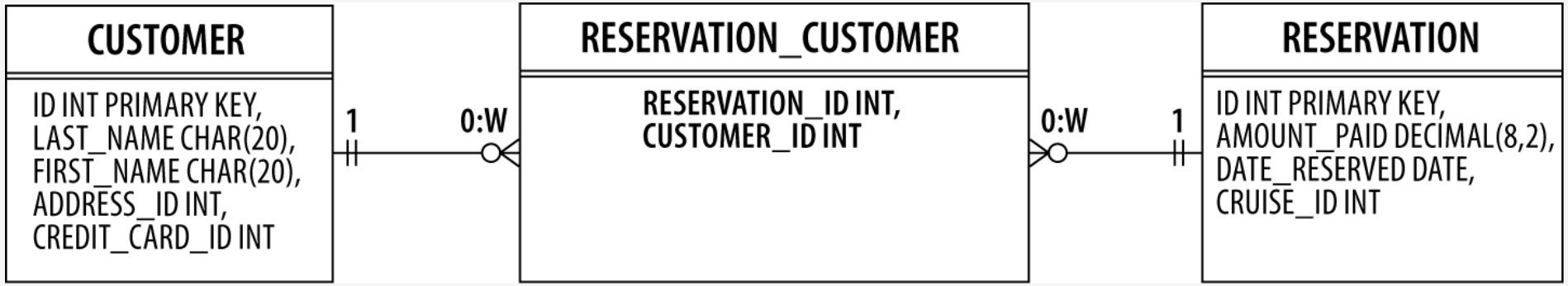
Hibernate - Rodzaje relacji - @ManyToMany




```java
@Entity
public class Cabin implements Serializable {
    @Id
    private int id;

}
```

```java
@Entity
public class Reservation implements Serializable {

    @Id
    private int id;

    @ManyToMany
    @JoinTable(name="CABIN_RESERVATION",
            joinColumns={@JoinColumn(name="RESERVATION_ID")},
            inverseJoinColumns={@JoinColumn(name="CABIN_ID")})
    private Set<Cabin> cabins = new HashSet<>();
}
```

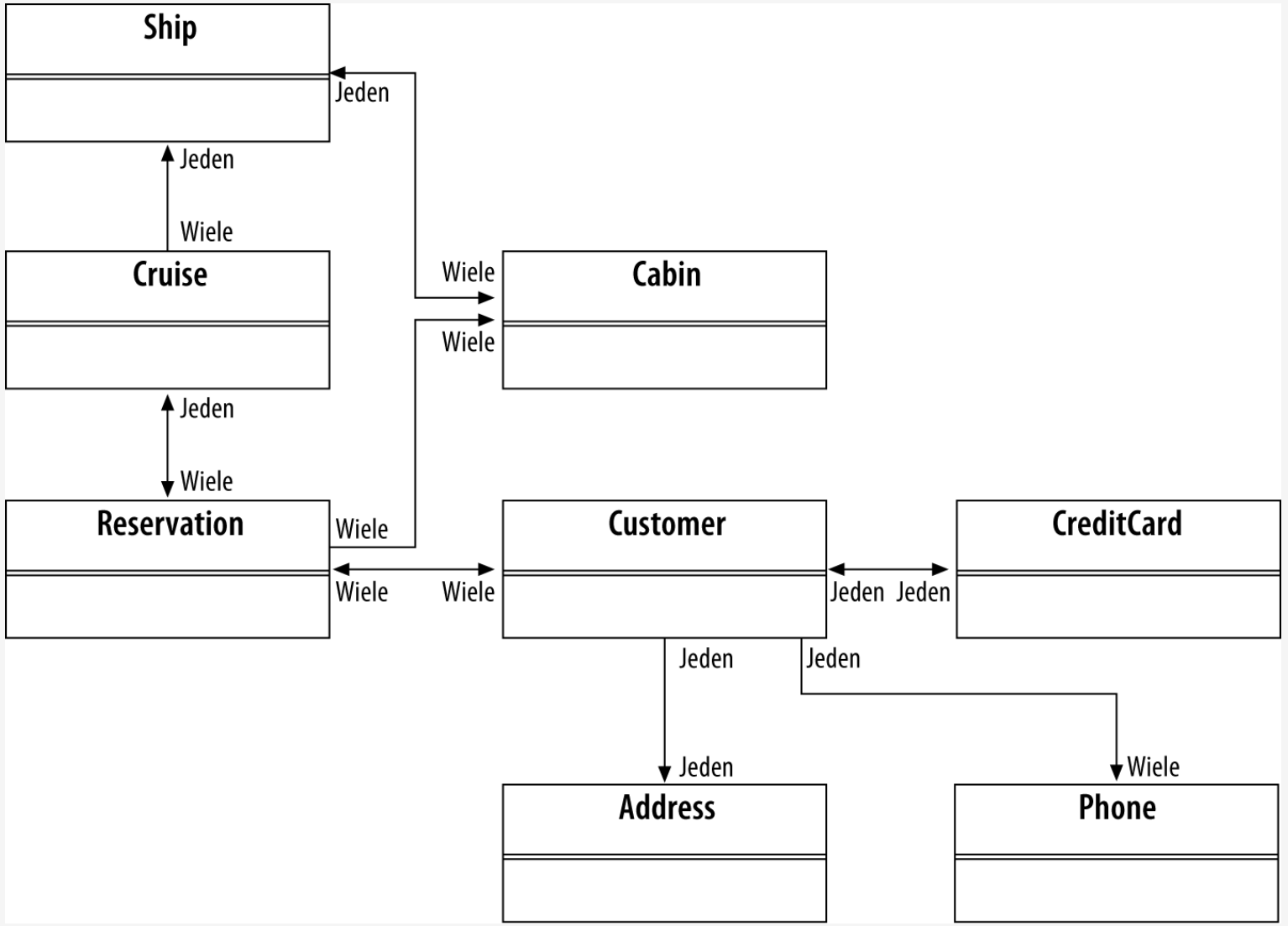
Hibernate - Rodzaje relacji - @ManyToMany




```java
@Entity
public class Customer implements Serializable {
    @Id
    private int id;

    @ManyToMany(mappedBy="customers")
    private Collection<Reservation> reservations = new ArrayList<>();
}
```

```java
@Entity
public class Reservation implements Serializable {
    @Id
    private int id;

    @ManyToMany
    @JoinTable(name="RESERVATION_CUSTOMER",
            joinColumns={@JoinColumn(name="RESERVATION_ID")},
            inverseJoinColumns={@JoinColumn(name="CUSTOMER_ID")})
    private Set<Customer> customers = new HashSet<>();
}
```

Hibernate - Rodzaje relacji - podsumowanie



Hibernate - zapytania - przykłady

```
SELECT c.creditCard FROM Customer AS c

SELECT c.address.city FROM Customer AS c

SELECT c.creditCard.creditCompany.address FROM Customer AS c

SELECT c.creditCard.creditCompany.address.city FROM Customer AS c

SELECT r FROM Customer AS c, IN( c.reservations ) r

SELECT r.cruise FROM Customer AS c, IN( c.reservations ) r

SELECT r.cruise FROM Customer c INNER JOIN c.reservations r

SELECT cbn.ship FROM Customer AS c,
IN ( c.reservations ) r,
IN( r.cabins ) cbn

SELECT cbn.ship FROM Customer c
INNER JOIN c.reservations r
INNER JOIN r.cabins cbn

SELECT cbn.ship FROM Customer c
JOIN c.reservations r
JOIN r.cabins cbn
```

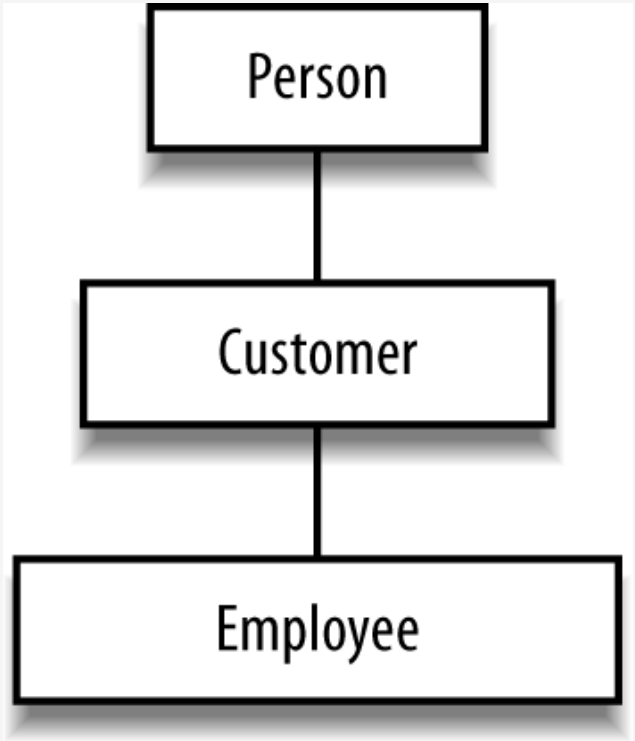
Hibernate - zapytania - przykłady

```
@NamedQueries({
        @NamedQuery(
                name = "get_employee_by_age",
                query = "select e " +
                        "from Employee e " +
                        "where e.age = :age"
        )
})
@Entity
public class Employee {
```

```
List<Employee> all =
        (List<Employee>) session.createQuery("from Employee")
        .setMaxResults(3)
        .getResultList();

List<Employee> byLogin =
        (List<Employee>) session.createQuery("from Employee e where e.login = :login")
        .setParameter("login", "Heniek")
        .getResultList();

List<Employee> byAge =
        (List<Employee>) session.createNamedQuery("get_employee_by_age")
        .setParameter("age", 18)
        .getResultList();
```

Hibernate - Strategie dziedziczenia

- Hierarchia klas w pojedynczej tabeli
- Jedna tabela dla konkretnej klasy
- Jedna tabela dla konkretnej podklasy



Hibernate - Strategie dziedziczenia - SINGLE_TABLE

```
create table PERSON_HIERARCHY (
   id integer primary key not null,
   firstName varchar(255),
   lastName varchar(255),
   street varchar(255),
   city varchar(255),
   state varchar(255),
   zip varchar(255),
   employeeId integer,
   DISCRIMINATOR varchar(31) not null
);
```

```
@Entity
@Table(name = "PERSON_HIERARCHY")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "DISCRIMINATOR",
        discriminatorType = DiscriminatorType.STRING)
@DiscriminatorValue("PERSON")
public class Person {
    @Id
    @GeneratedValue
    private int id;
    private String firstName;
    private String lastName;

}
```

```
@Entity
@DiscriminatorValue("CUST")
public class Customer extends Person {
```

```
// domyślna wartość dyskryminatora
@Entity
public class Employee extends Customer {
```

Hibernate - Strategie dziedziczenia - TABLE_PER_CLASS

```sql
create table Person (
  id integer primary key not null,
  firstName varchar(255),
  lastName varchar(255)
);
create table Customer (
  id integer primary key not null,
  firstName varchar(255),
  lastName varchar(255),
  street varchar(255),
  city varchar(255),
  state varchar(255),
  zip varchar(255)
);
create table Employee (
  id integer primary key not null,
  firstName varchar(255),
  lastName varchar(255),
  street varchar(255),
  city varchar(255),
  state varchar(255),
  zip varchar(255),
  employeeId integer
);
```

```java
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Person {
```

```java
@Entity
public class Customer extends Person {
```

```java
@Entity
public class Employee extends Customer {
```

Hibernate - Strategie dziedziczenia - JOINED

```sql
create table Person (
  id integer primary key not null,
  firstName varchar(255),
  lastName varchar(255)
);
create table Customer (
  id integer primary key not null,
  street varchar(255),
  city varchar(255),
  state varchar(255),
  zip varchar(255)
);
create table Employee (
  EMP_PK integer primary key not null,
  employeeId integer
);
```

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

```java
@Entity
public class Customer extends Person {
```

```java
@Entity
@PrimaryKeyJoinColumn(name="EMP_PK")
public class Employee extends Customer {
```