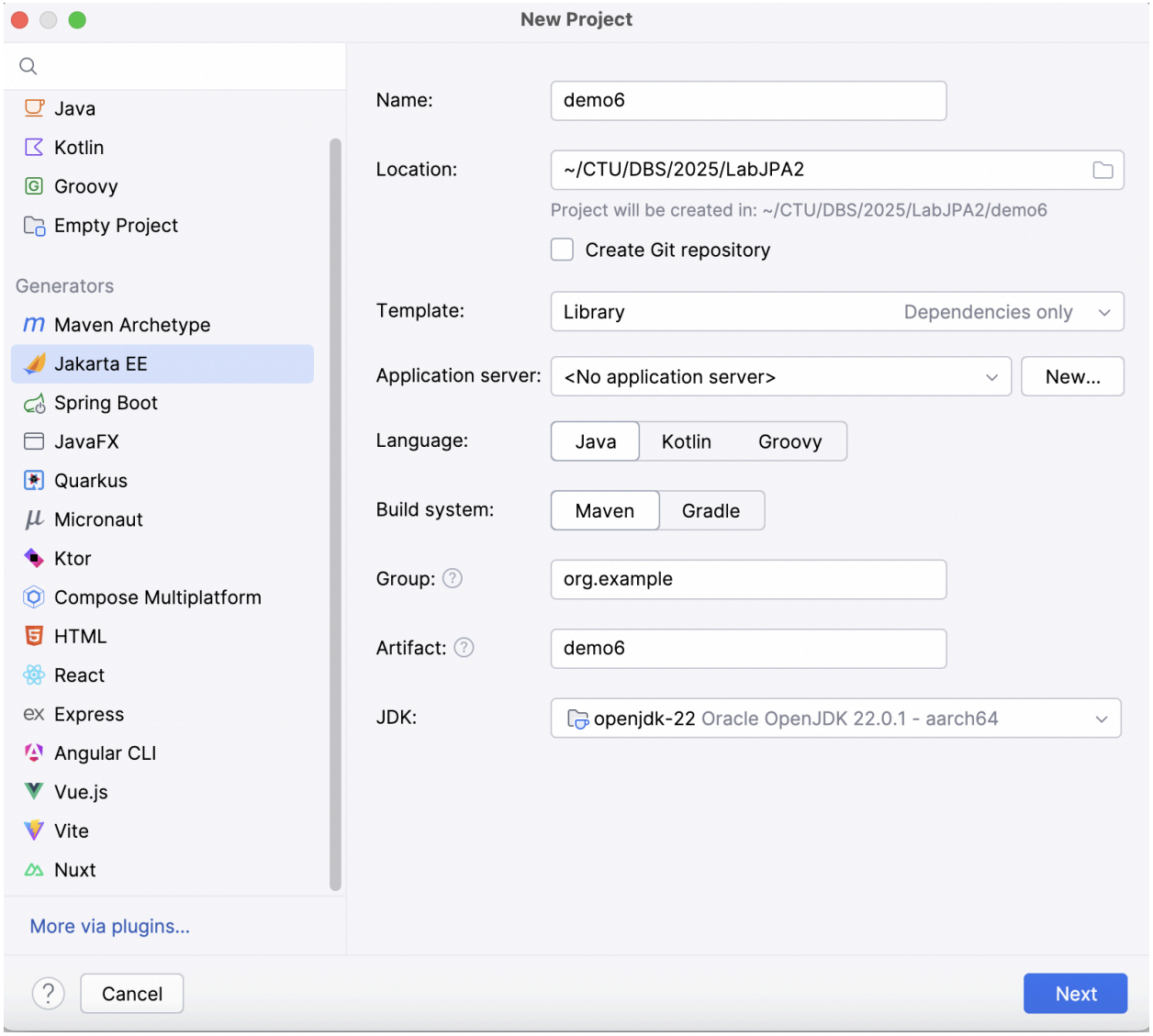
Instructions for connecting to the PostgreSQL DB from a Java project (JPA)

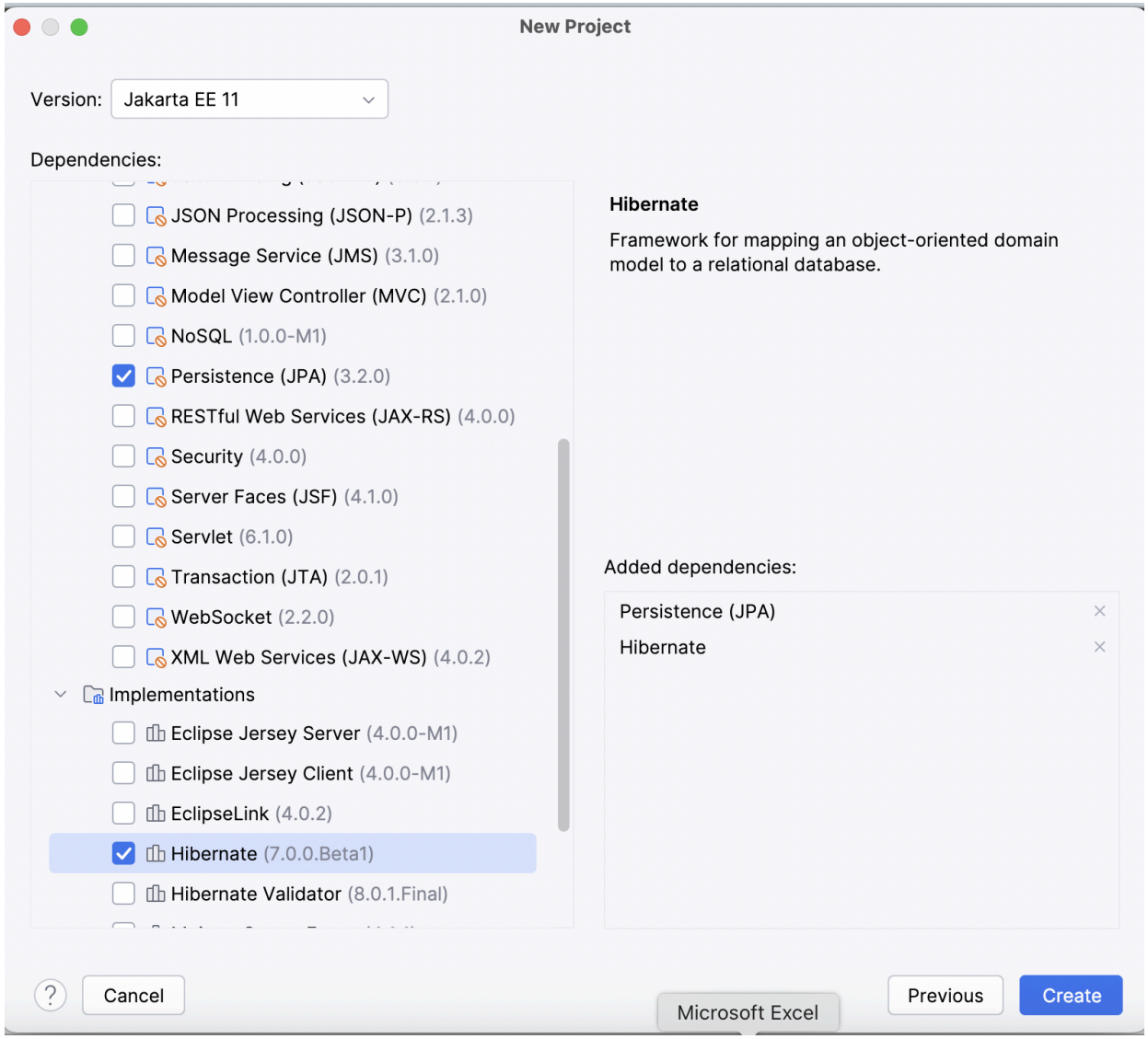
Step-by-step algorithm for creating and customizing an application

This is one possible way; there are other ways too

1. Create a Jakarta EE application:



2. Select Jakarta version, dependencies, and implementations.

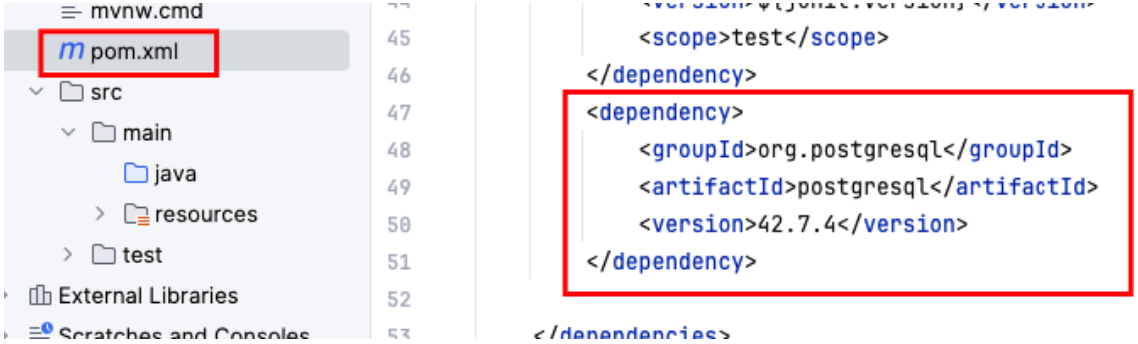


There are two important files to edit: **pom.xml** and **persistence.xml**.

3. Add information about the **PostgreSQL driver** (file **pom.xml**).

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.4</version>
</dependency>
```

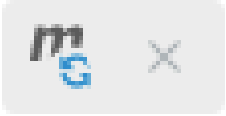
You can enter it manually or generate it.



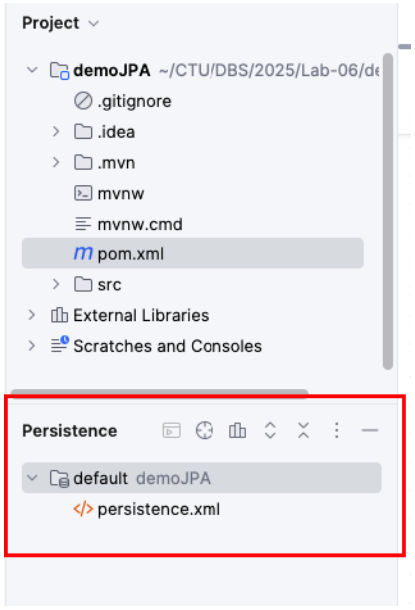
4. Similarly, add a **dependency for Jakarta.persistence-api**

```
<dependency>
    <groupId>jakarta.persistence</groupId>
    <artifactId>jakarta.persistence-api</artifactId>
    <version>3.2.0</version>
</dependency>
```

5. Click the **Refresh** button (at the upper-right side of the window)!



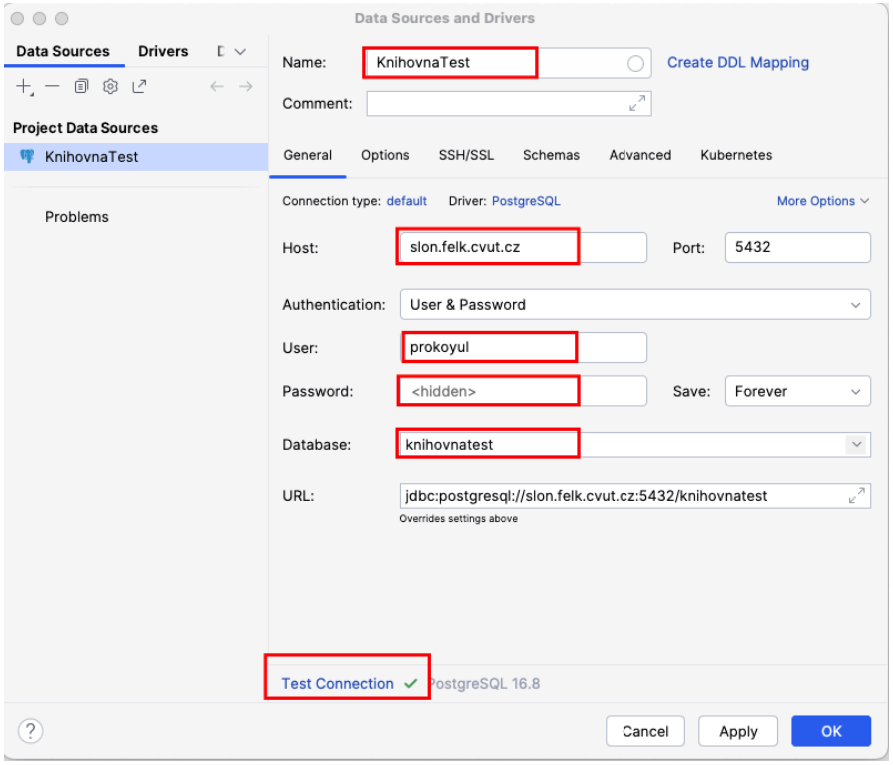
6. View -> **Tool Windows -> Persistence** (this opens a Persistence window).



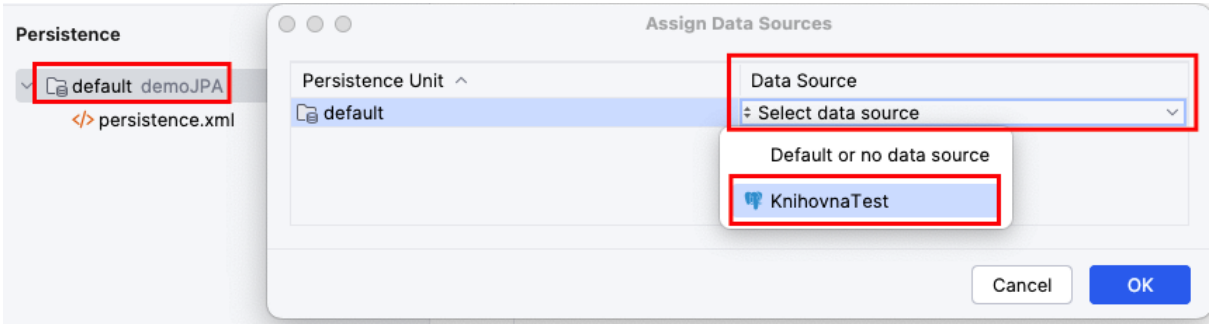
7. Add information about the persistence unit (file persistence.xml). The name of the persistence unit is **“default”** (by default). We can change it.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                                  xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_2.xsd"
             version="3.2">
   <persistence-unit name="default" transaction-type="RESOURCE_LOCAL">

       <properties>
 <property name="jakarta.persistence.jdbc.driver"   value="org.postgresql.Driver"/>
<property        name="jakarta.persistence.jdbc.url"                                                value=
"jdbc:postgresql://slon.felk.cvut.cz:5432/username"/>
           <property name="jakarta.persistence.jdbc.user"     value="username"/>
           <property name="jakarta.persistence.jdbc.password" value="password"/>
          <property name="hibernate.dialect"      value="org.hibernate.dialect.PostgreSQLDialect"/>
           <property name="hibernate.hbm2ddl.auto" value="validate"/>
       </properties>

   </persistence-unit>
</persistence>
```

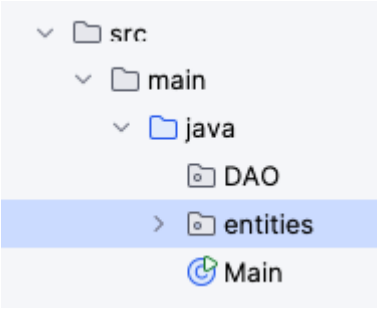
8. Add **a Data source** (your database): **View> Tool Windows> Database**. Add (**+**) – Data Source – PostgreSQL.



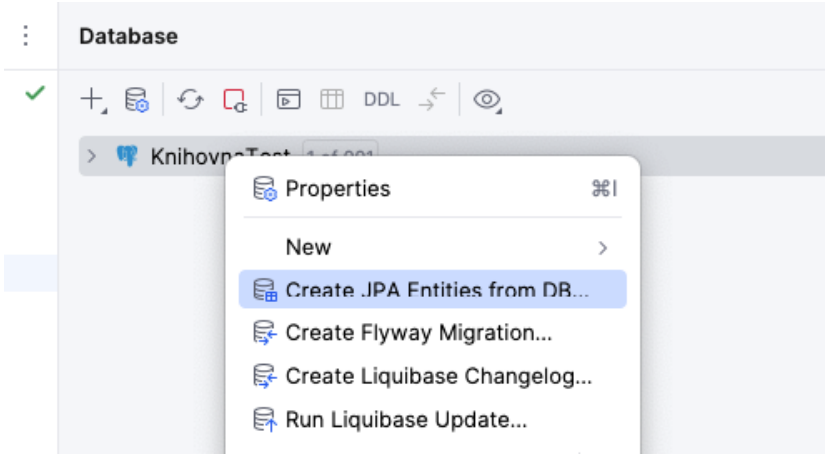
9. In the **Persistence window**, right-click on **default** – *Assign Data Sources* – select your database.



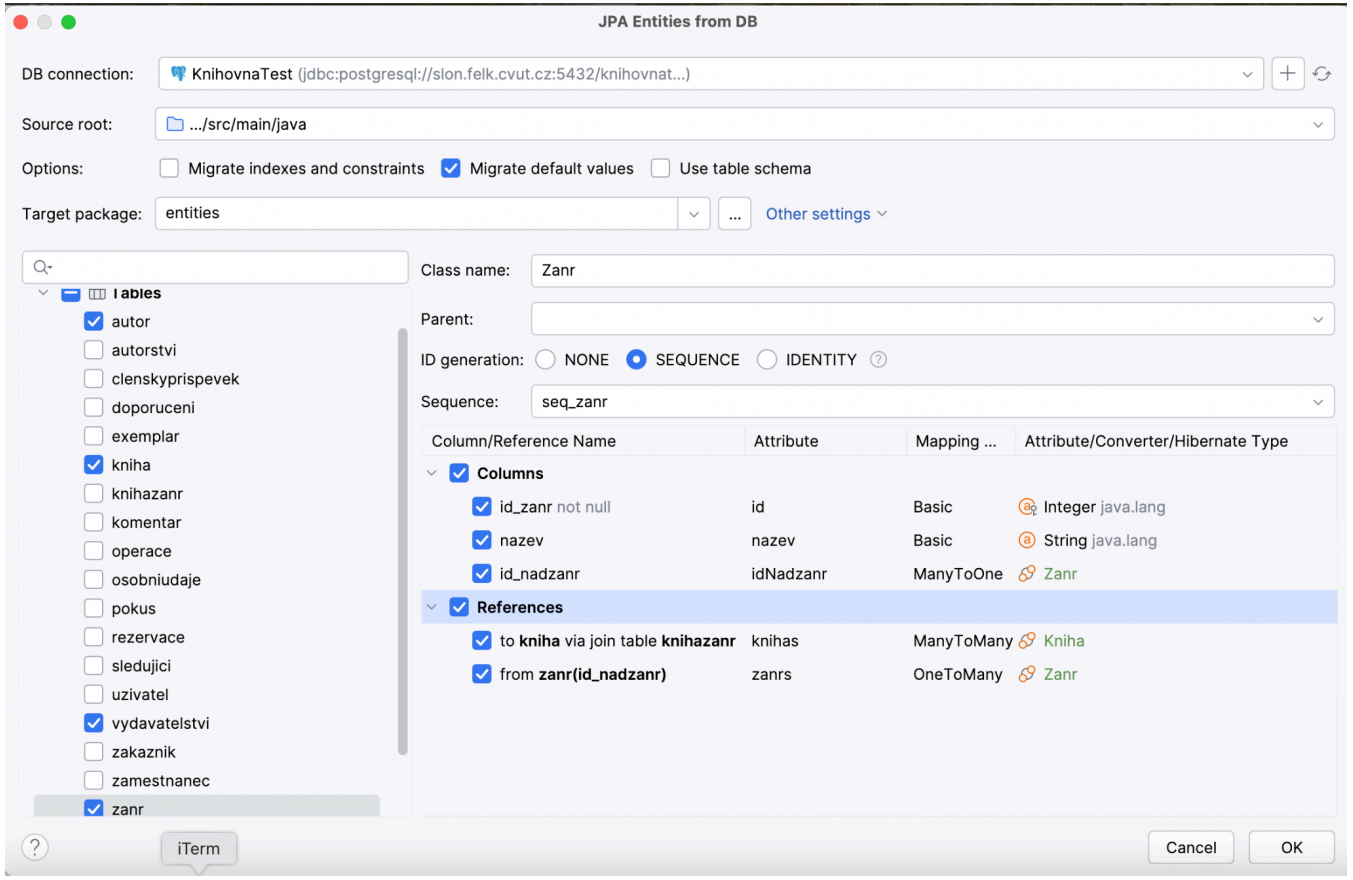
10. Create two packages: entities and DAO.



11. Generate a class for the simple entity from your database. In the Database window, right-click on the database name – **Create JPA Entities from DB**.



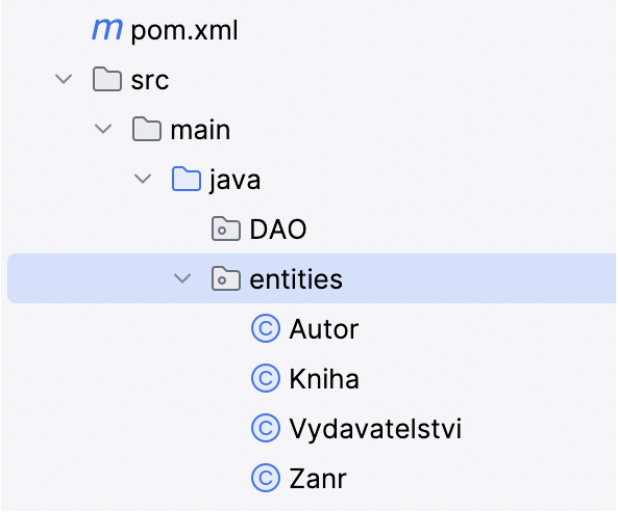
Choose the package and the table.



Pokud označíte související tabulky spolu s hlavní tabulkou, **vazby** mohou být **vygenerovány automaticky**.

Všimněte si, že **tabulky vytvořené v databázi pro vazby mnoho-k-mnoha generovat NEMUSÍTE**!

12. After this, the new entity classes will be generated:



Automatic content of the class **Vydavatelstvi**:

```java
package entities;

import ...

@Entity
@Table(name = "vydavatelstvi")
public class Vydavatelstvi {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "vydavatelstvi_id_gen")
    @SequenceGenerator(name = "vydavatelstvi_id_gen", sequenceName = "vydavatelstvi_id_vydavatelstvi_seq", allocationSize = 1)
    @Column(name = "id_vydavatelstvi", nullable = false)
    private Long id;

    @Column(name = "nazev", length = 64)  2 usages
    private String nazev;

    @OneToMany(mappedBy = "idVydavatelstvi")  2 usages
    private Set<Kniha> knihas = new LinkedHashSet<>();

    public Long getId() { return id; }

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

    public String getNazev() { return nazev; }

    public void setNazev(String nazev) { this.nazev = nazev; }

    public Set<Kniha> getKnihas() { return knihas; }

    public void setKnihas(Set<Kniha> knihas) { this.knihas = knihas; }

}
```

id a **nazev** – pole třídy.
Před nimi jsou uvedeny odpovídající sloupce v asociovaných tabulkách: "id_vydavatelstvi" a "nazev"

Vydavatelství vydalo mnoho knih, proto bylo přidáno pole – **množina** (set unikátních hodnot) knih s anotací @OneToMany:

```
@OneToMany(mappedBy = "idVydavatelstvi")
private Set<Kniha> knihas = new LinkedHashSet<>();
```

Automatic content of the class **Kniha**:

```java
@Entity
@Table(name = "kniha")
public class Kniha {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @ColumnDefault("nextval('kniha_id_kniha_seq')")
    @Column(name = "id_kniha", nullable = false)
    private Integer id;

    @Column(name = "isbn", nullable = false)  2 usages
    private String isbn;

    @Column(name = "nazev", nullable = false, length = 64)  2 usages
    private String nazev;

    @Column(name = "rokvydani")  2 usages
    private Integer rokvydani;

    @Column(name = "anotace", length = Integer.MAX_VALUE)  2 usages
```

```
29
30      @ManyToOne(fetch = FetchType.LAZY, optional = false)  2 usages
31      @JoinColumn(name = "id_vydavatelstvi", nullable = false)
32      private Vydavatelstvi idVydavatelstvi;
33
34      @Column(name = "url", nullable = false)  2 usages
35      private String url;
36
37      @ManyToMany  2 usages
38      @JoinTable(name = "autorstvi",
39              joinColumns = @JoinColumn(name = "id_kniha"),
40              inverseJoinColumns = @JoinColumn(name = "id_autor"))
41      private Set<Autor> autors = new LinkedHashSet<>();
42
43      @ManyToMany  2 usages
44      @JoinTable(name = "knihazanr",
45              joinColumns = @JoinColumn(name = "id_kniha"),
46              inverseJoinColumns = @JoinColumn(name = "id_zanr"))
47      private Set<Zanr> zanrs = new LinkedHashSet<>();
```

Pole třídy: id, isbn, nazev, rokVydani, anotace, url – která odpovídají sloupcům ve spřažené tabulce.

Navázané pole: idVydavatelstvi (jeden záznam, na rozdíl od kolekce v třídě Vydavatelstvi).

!!!
Změňte Kniha.id aby byla Integer místo Long:

```
private Integer id;
```

Automatic content of the class **Zanr**:

```java
@Entity
@Table(name = "zanr")
public class Zanr {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "zanr_id_gen")
    @SequenceGenerator(name = "zanr_id_gen", sequenceName = "seq_zanr", allocationSize = 1)
    @Column(name = "id_zanr", nullable = false)
    private Integer id;

    @Column(name = "nazev", length = 64)  2 usages
    private String nazev;

    @ManyToOne(fetch = FetchType.LAZY)  3 usages
    @JoinColumn(name = "id_nadzanr")
    private Zanr idNadzanr;

    @ManyToMany  2 usages
    @JoinTable(name = "knihazanr",
            joinColumns = @JoinColumn(name = "id_zanr"),
            inverseJoinColumns = @JoinColumn(name = "id_kniha"))
    private Set<Kniha> knihas = new LinkedHashSet<>();

    @OneToMany(mappedBy = "idNadzanr")  2 usages
    private Set<Zanr> zanrs = new LinkedHashSet<>();

```

Pro rekurzivní vazbu **1:N** bylo vytvořeno pole idNadzanr.
Zároveň vznikla množina zanrs pro realizaci této vazby z druhé strany.

Pro vazbu **N:M** s třídou Kniha bylo vytvořeno pole-množina `knihas`.

- Použije se anotace `@ManyToMany`.
- Místo `@Column` se uvede `@JoinTable` a specifikuje se tabulka `"knihazanr"`, která existuje v DB, ale pro kterou jsme třídu **NE**generovali.
- Uvádějí se dva sloupce, jež budou použity pro vazbu:
  - sloupec pro cizí klíč na `Kniha`
  - sloupec pro cizí klíč na `Zanr`

```
33    public Integer getId() { return id; }
36
37    public void setId(Integer id) { this.id = id; }
40
41    public String getNazev() { return nazev; }
44
45    public void setNazev(String nazev) { this.nazev = nazev; }
48
49    public Zanr getIdNadzanr() { return idNadzanr; }
52
53    public void setIdNadzanr(Zanr idNadzanr) { this.idNadzanr = idNadzanr; }
56
57    public Set<Kniha> getKnihas() { return knihas; }
```

13. Create the Main class and write a simple query to test the connection:

Základní kostra třídy Main, která inicializuje EntityManagerFactory, vytvoří EntityManager a následně uzavře oba zdroje.

```
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

public class Main {
   public static void main(String[] args) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
        EntityManager em = emf.createEntityManager();



        // Close EntityManager
        em.close();
        emf.close();

   }
}
```

Píšeme jednoduché dotazy, abychom ověřili, že třídy a vazby jsou správně vytvořeny.

Tyto dotazy slouží jen pro průběžné testování, nesplňují zadání úlohy.

```java
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;
import java.util.List;

public class Main {
   public static void main(String[] args) {

       EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
       EntityManager em = emf.createEntityManager();

       // 1. Výpis seznamu všech vydavatelství
       List<entities.Vydavatelstvi> vydavatelstvi = em.createQuery(
               "SELECT v FROM Vydavatelstvi v", entities.Vydavatelstvi.class
       ).getResultList();
       System.out.println("Všechna vydavatelství:");
       for (entities.Vydavatelstvi v : vydavatelstvi) {
           System.out.println("  - " + v.getNazev());
       }
```

```java
// 2. Výpis knih: název knihy, název vydavatelství, rok vydání
List<Object[]> knihySInfo = em.createQuery(
        "SELECT k.nazev, k.idVydavatelstvi.nazev, k.rokvydani FROM Kniha k", Object[].class
).getResultList();
System.out.println("\nKnihy s informací o vydavatelství a roce vydání:");
for (Object[] row : knihySInfo) {
    System.out.printf("  - %s | %s | %d%n",
            row[0],    // název knihy
            row[1],    // název vydavatelství
            row[2]     // rok vydání
    );
}

// 3. Výpis všech knih od zvoleného vydavatelství (např. id = 2)
Long publisherId = 2L;
List<entities.Kniha> knihyOd = em.createQuery(
                "SELECT k FROM Kniha k WHERE k.idVydavatelstvi.id = :pid", entities.Kniha.class
        )
        .setParameter("pid", publisherId)
        .getResultList();
System.out.println("\nKnihy vydané vydavatelstvím s ID " + publisherId + ":");
for (entities.Kniha k : knihyOd) {
    System.out.println("  - " + k.getNazev());
}
```

```java
// 4. Výpis všech knih s jejich autory
List<entities.Kniha> knihySAutory = em.createQuery(
        "SELECT DISTINCT k FROM Kniha k LEFT JOIN FETCH k.autors", entities.Kniha.class
).getResultList();
System.out.println("\nKnihy s autory:");
for (entities.Kniha k : knihySAutory) {
    System.out.print("  - " + k.getNazev() + " [");
    String sep = "";
    for (entities.Autor a : k.getAutors()) {
        System.out.print(sep + a.getJmeno() + " " + a.getPrijmeni());
        sep = ", ";
    }
    System.out.println("]");
}
// 5. Výpis všech autorů a jejich knih
List<entities.Autor> autori = em.createQuery(
        "SELECT DISTINCT a FROM Autor a LEFT JOIN FETCH a.knihas", entities.Autor.class
).getResultList();
System.out.println("\nAutoři a jejich knihy:");
for (entities.Autor a : autori) {
    System.out.print("  - " + a.getJmeno() + " " + a.getPrijmeni() + ": ");
    String sep = "";
    for (entities.Kniha k : a.getKnihas()) {
        System.out.print(sep + k.getNazev());
        sep = ", ";
    }
    System.out.println();
}
```

```
        // Close EntityManager
        em.close();
        emf.close();
    }
}
```

```
Všechna vydavatelství:
  - Universum (ČR)
  - Grada
  - Pikola
  - Moderní programování

Knihy s informací o vydavatelství a roce vydání:
  - Učení a paměť pravou hemisférou | Universum (ČR) | 2018
  - Auta první republiky | Grada | 2017
  - Ozubnicové a pozemní lanové dráhy Evropy | Grada | 2023
  - Péťa umí vyprávět | Pikola | 2024
  - Auta v Československu 1945-1990 | Grada | 2024
  - Players handbook | Moderní programování | 2014
  - Bible 2 | Moderní programování | 0
  - Databáze a SQL pro začátečníky | Moderní programování | 2021
  - Pes | Grada | 2024
  - Aboba | Grada | 2024
  - Mordenkainen Presents: Monsters of the multiverse | Grada | 2015

Knihy vydané vydavatelstvím s ID 2:
  - Auta první republiky
  - Ozubnicové a pozemní lanové dráhy Evropy
  - Auta v Československu 1945-1990
  - Pes
  - Aboba
  - Mordenkainen Presents: Monsters of the multiverse
```

```
Knihy s autory:
  - Players handbook []
  - Ozubnicové a pozemní lanové dráhy Evropy [Petr Harák]
  - Auta v Československu 1945-1990 [Jan Tuček]
  - Péťa umí vyprávět [Ondra Klos, Marta Galewska-Kustra]
  - Pes [Martina Kovarova]
  - Auta první republiky [Jan Tuček]
  - Učení a paměť pravou hemisférou [Pepa Vladyka]
  - Bible 2 []
  - Aboba [Martina Kovarova]
  - Databáze a SQL pro začátečníky [Radek Vystavěl]
  - Mordenkainen Presents: Monsters of the multiverse []

Autoři a jejich knihy:
  - Petr Harák: Ozubnicové a pozemní lanové dráhy Evropy
  - Jan Tuček: Auta v Československu 1945-1990, Auta první republiky
  - Jan Novy:
  - Radek Vystavěl: Databáze a SQL pro začátečníky
  - Martina Kovarova: Pes, Aboba
  - Marta Galewska-Kustra: Péťa umí vyprávět
  - Pepa Vladyka: Učení a paměť pravou hemisférou
  - Jiri Novotny:
  - Ondra Klos: Péťa umí vyprávět
```

Vidíme, že vazba **N:M Kniha–Autor** funguje správně, vazba **1:N Kniha–Vydavatelství** také funguje správně.

14. Přejdeme k vytvoření DAO vrstvy.

Můžete vytvořit abstraktní třídu **BaseDao** a poté ji rozšířit pro každou entitu.

AutorDAO

```
package DAO;

import entities.Autor;
import jakarta.persistence.EntityManager;

import java.util.List;

public class AutorDAO extends BaseDAO<Autor, Integer> {
    public AutorDAO(EntityManager em) {
        super(em, Autor.class);
    }

    // 5. všichni autoři s jejich knihami
    public List<Autor> findAllWithBooks() {
        return em.createQuery(
                "SELECT DISTINCT a FROM Autor a LEFT JOIN FETCH a.knihas", Autor.class
        ).getResultList();
    }
}
```

KnihaDAO

```
package DAO;

import entities.Kniha;
import jakarta.persistence.EntityManager;

import java.util.List;

public class KnihaDAO extends BaseDAO<Kniha, Long> {
    public KnihaDAO(EntityManager em) {
        super(em, Kniha.class);
    }

    // 2. název knihy, název vydavatelství, rok vydání
    public List<Object[]> findBookDetails() {
        return em.createQuery(
                "SELECT k.nazev, k.idVydavatelstvi.nazev, k.rokvydani FROM Kniha k", Object[].class
        ).getResultList();
    }

    // 3. knihy podle vydavatele
public List<Kniha> findByPublisherName(String publisherName) {
    return em.createQuery(
                    "SELECT k FROM Kniha k WHERE k.idVydavatelstvi.nazev = :pname", Kniha.class
            )
            .setParameter("pname", publisherName)
            .getResultList();
}
```

```java
// 4. všechny knihy s autory
public List<Kniha> findAllWithAuthors() {
    return em.createQuery(
            "SELECT DISTINCT k FROM Kniha k LEFT JOIN FETCH k.autors", Kniha.class
    ).getResultList();
}

// 6. Pokud nic nenalezeno, vytvořit novou knihu spolu s vydavatelstvím
em.getTransaction().begin();
Vydavatelstvi pub = em.createQuery(
                "SELECT v FROM Vydavatelstvi v WHERE v.nazev = :name", Vydavatelstvi.class)
        .setParameter("name", publisherName)
        .getResultStream().findFirst()
        .orElseGet(() -> {
            Vydavatelstvi v = new Vydavatelstvi();
            v.setNazev(publisherName);
            em.persist(v);
            return v;
        });
Kniha k = new Kniha();
k.setIsbn(isbn);
k.setNazev(nazev);
k.setRokvydani(rok);
k.setUrl(url);
k.setIdVydavatelstvi(pub);
em.persist(k);
// Aktualizace oboustranné vazby: přidáme knihu do kolekce vydavatelství
pub.getKnihas().add(k);
```

```
    em.getTransaction().commit();
    return k;
   }

}
```

VydavatelstviDAO

```
package DAO;

import entities.Vydavatelstvi;
import jakarta.persistence.EntityManager;
import java.util.List;

public class VydavatelstviDAO extends BaseDAO<Vydavatelstvi, Long> {
    public VydavatelstviDAO(EntityManager em) {
        super(em, Vydavatelstvi.class);
    }

      // Vrátí vydavatelství dle názvu
    public List<Vydavatelstvi> findByName(String name) {
        return em.createQuery(
                        "SELECT v FROM Vydavatelstvi v WHERE v.nazev = :name", Vydavatelstvi.class
                )
                .setParameter("name", name)
                .getResultList();
    }
```

```
    // Vloží nové vydavatelství se zadaným názvem
    public void createPublisher(String name) {
        Vydavatelstvi v = new Vydavatelstvi();
        v.setNazev(name);
        save(v);
    }
    // Aktualizuje název vydavatelství dle názvu
public int updateName(String currentName, String newName) {
    em.getTransaction().begin();
    int count = em.createQuery(
                    "UPDATE Vydavatelstvi v SET v.nazev = :newName WHERE v.nazev = :currentName"
            )
            .setParameter("newName", newName)
            .setParameter("currentName", currentName)
            .executeUpdate();
    em.getTransaction().commit();
    return count;
}
    //Smaže vydavatelství dle názvu
public int delete(String name) {
    em.getTransaction().begin();
    int count = em.createQuery(
                    "DELETE FROM Vydavatelstvi v WHERE v.nazev = :name"
            )
            .setParameter("name", name)
            .executeUpdate();
    em.getTransaction().commit();
    return count;
}
}
```

Main

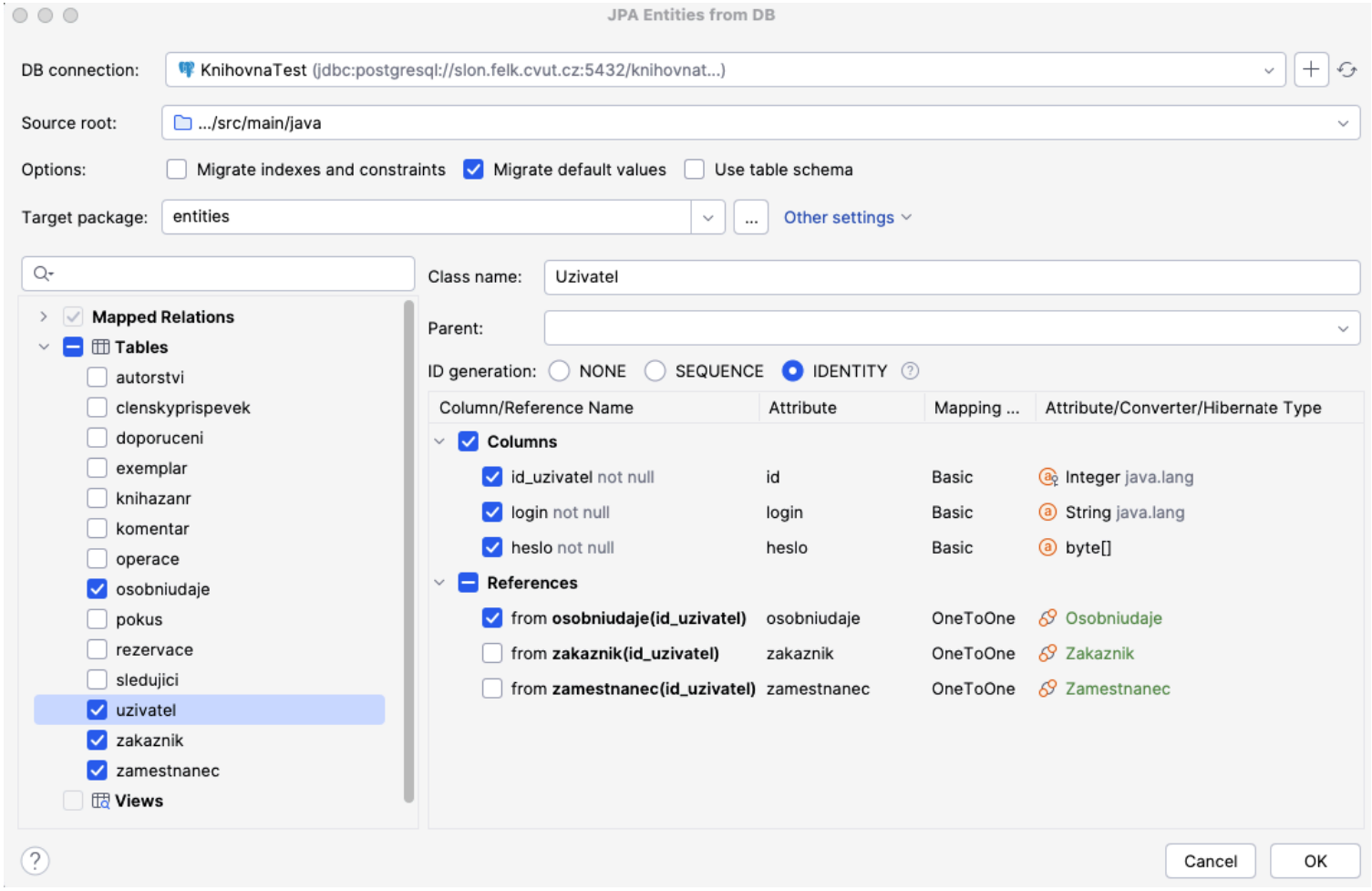
```
import DAO.*;
import entities.*;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;
import java.util.List;

public class Main {
   public static void main(String[] args) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");
        EntityManager em = emf.createEntityManager();

        AutorDAO autorDAO = new AutorDAO(em);
        KnihaDAO knihaDAO = new KnihaDAO(em);
        VydavatelstviDAO vydDAO = new VydavatelstviDAO(em);

        // 1. Všechna vydavatelství
        List<Vydavatelstvi> allPub = vydDAO.findAll();
        System.out.println("Vydavatelství:");
        allPub.forEach(v -> System.out.println(" - " + v.getNazev()));

        // 2. Detaily knih
        System.out.println("\nKnihy (název | vydavatelství | rok):");
        knihaDAO.findBookDetails().forEach(r ->
                System.out.printf(" - %s | %s | %s\n", r[0], r[1], r[2])
        );
```

```java
// 3. Knihy od vydavatelství s názvem 'Grada'
System.out.println("\nKnihy vydavatelství 'Grada':");
knihaDAO.findByPublisherName("Grada").forEach(k -> System.out.println(" - " + k.getNazev()));

// 4. Knihy s autory
System.out.println("\nKnihy s autory:");
knihaDAO.findAllWithAuthors().forEach(k -> {
    System.out.print(" - " + k.getNazev() + ": ");
    k.getAutors().forEach(a -> System.out.print(a.getJmeno() + " " + a.getPrijmeni() + ", "));
    System.out.println();
});

// 5. Autoři a jejich knihy
System.out.println("\nAutoři a jejich knihy:");
autorDAO.findAllWithBooks().forEach(a -> {
    System.out.print(" - " + a.getJmeno() + " " + a.getPrijmeni() + ": ");
    a.getKnihas().forEach(k -> System.out.print(k.getNazev() + ", "));
    System.out.println();
});

// Příklad vložení, aktualizace a smazání vydavatelství podle názvu
 vydDAO.createPublisher("Nové Vydavatelství");
 vydDAO.updatePublisherName("Nové Vydavatelství", "Updated Vydavatelství");
 vydDAO.deleteByName("Updated Vydavatelství");
```

```
        // 6. Příklad transakce: pokud neexistuje, vytvořit vydavatelství, poté vytvořit knihu
        Kniha book;
        try {
            book = knihaDAO.createWithPublisher(
                    "978-80-247-0000-0", "Nová Kniha", 2025,
                    "http://example10.com", "Grada"
            );
            System.out.println("Created book " + book.getNazev() + " with publisher " +
        book.getIdVydavatelstvi().getNazev());
        } catch (Exception e) {
            System.err.println("Nepodařilo se přidat knihu: " + e.getMessage());
        }
    // Close EntityManager
    em.close();
    emf.close();
  }
}
```

15. Vygenerujme třídy pro tabulky **osobniudaje**, **uzivatel**, **zakaznik**, **zamestnanec**

Budeme mít **dědičnost**, takže nepotřebujeme vztahy, které se nabízejí automaticky.



Vztah mezi **Uzivatel** a **Osobniudaje** teď funguje takto:

Tabulky

i. **uzivatel** má svůj PK **id_uzivatel** a sloupce **login**, **heslo**…
ii. **osobniudaje** má taky PK **id_uzivatel**, ale ten zároveň slouží jako FK na **uzivatel.id_uzivatel**.

V Osobniudaje máte

```
@Id

@Column(name = "id_uzivatel", nullable = false)

private Integer id;



@MapsId

@OneToOne(fetch = FetchType.LAZY, optional = false)

@JoinColumn(name = "id_uzivatel", nullable = false)

private Uzivatel uzivatel;
```

– to říká: „Pole **id** je zároveň FK na uživatele a sdílí s ním stejnou hodnotu.“

V Uzivatel máte

```
@OneToOne(mappedBy="uzivatel", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
private Osobniudaje osobniudaje;
```

– to říká: „Nejádelní FK tady, ale navigační pole; vlastník vazby je Osobniudaje.uzivatel.“

Test (v Main)

```
List<Object[]> rows = em.createQuery(
        "SELECT u.login, o.jmeno, o.prijmeni " +
                "FROM Uzivatel u JOIN u.osobniudaje o", Object[].class
).getResultList();

for (Object[] r : rows) {
    System.out.printf("Login=%s, Jméno=%s %s%n", r[0], r[1], r[2]);
}
```

```
Login=rimnacm, Jméno=Martin Řimnáč
Login=rimnaca, Jméno=Adéla Řimnáčová
Login=uzitnyjosef, Jméno=Jan Novak
```

Uzivatel.java – supertřída

```
@Entity

@Table(name = "uzivatel")

// Supertřída pro všechny typy uživatelů.

// Dědičnost JOINED: obecná data v tabulce uzivatel, specifika v tabulkách potomků.

@Inheritance(strategy = InheritanceType.JOINED)

public class Uzivatel {...
```

Zakaznik.java – potomek

```
@Entity

@Table(name = "zakaznik")

// PK zakaznik.id_uzivatel je FK na uzivatel.id_uzivatel.

@PrimaryKeyJoinColumn(name="id_uzivatel")

public class Zakaznik extends Uzivatel {

...
```

!!! Odstrante id a gettery-settery

@Id

@Column(name = "id_uzivatel", nullable = false)

private Integer id;

`Zamestnanec.java` – potomek

Podobně

Test (v Main)

```
List<Object[]> rows = em.createQuery(
        "SELECT z.login, z.telefon FROM Zamestnanec z", Object[].class
).getResultList();
for (Object[] row : rows) {
    System.out.println("Login: " + row[0] + ", Telefon: " + row[1]);
}
```

```
Login: rimnacm, Telefon: +420 666 666 666
Login: rimnack, Telefon: +420 777 661 235
Login: rimnaca, Telefon: +420 123 456 789
Login: rimnacz, Telefon: +420 023 456 789
```

16. **Dědičnost** v DAO

```
public class ZamestnanecDAO extends BaseDAO<Zamestnanec, Integer> {

    public ZamestnanecDAO(EntityManager em) {

        super(em, Zamestnanec.class);

    }

    //Vrátí login a telefon všech zaměstnanců.

    public List<Object[]> findAllLoginsAndPhones() {

        return em.createQuery(

                "SELECT z.login, z.telefon FROM Zamestnanec z", Object[].class

        ).getResultList();

    }

}
```

V Main

```
ZamestnanecDAO zamDAO = new ZamestnanecDAO(em);

System.out.println("\nSeznam zaměstnanců (login | telefon):");

List<Object[]> list = zamDAO.findAllLoginsAndPhones();

list.forEach(r -> System.out.printf(" - %s | %s%n", r[0], r[1]));
```

```
Seznam zaměstnanců (login | telefon):
 - rimnacm | +420 666 666 666
 - rimnack | +420 777 661 235
 - rimnaca | +420 123 456 789
 - rimnacz | +420 023 456 789
```

17. Musíte vytvořit hlavní aplikaci, která používá vaše třídy a demonstruje například 5 dotazů:
    1. Operace vložení
    2. Operace aktualizace
    3. Operace mazání
    4. Výstup potomků s rodičovskými poli
    5. Výstup dat z tabulky, která má vztah many-to-many s jinou tabulkou. Výstup musí obsahovat pole z obou tabulek.
    6. Tranzakce (CP4)