Зміст

[JPA1 4](#_Toc166953896)

[Poznamky relačných diagramov ER (синие табл) + UML diagramov tried OO 4](#_Toc166953897)

[(p1jpa) 5](#_Toc166953898)

[create EntityManager 5](#_Toc166953899)

[čítanie dat z databazy 5](#_Toc166953900)

[modifikacia dat v databaze 5](#_Toc166953901)

[vyhľadávanie podľa kľúča 5](#_Toc166953902)

[vytvorenie nového záznamu 5](#_Toc166953903)

[JPA2 - Entity a JPA anotácie, JPQL 6](#_Toc166953904)

[Generovanie entitnej triedy a entitnej triedy z tabuľky 6](#_Toc166953905)

[(p23jpql) JPQL dotazy 6](#_Toc166953906)

[(p-- CV2-3) jpql dotazy 7](#_Toc166953907)

[(p21) equals, hash, toStraing, clear, detach, refresh, merge, remove 7](#_Toc166953908)

[(P22 CV2-2) @Temporal and columnDefinition=” BOOLEAN” 8](#_Toc166953909)

[JPA3 - Jednosmerné asociácie (a kolekcie) 9](#_Toc166953910)

[1. @ElementCollection (p31kolekcia) 9](#_Toc166953911)

[2. Jednosmerná N:N asociácia (p32uniNN) 9](#_Toc166953912)

[3. Jednosmerná N:1 asociácia (p33uniN1) 9](#_Toc166953913)

[4. Jednosmerná 1:N asociácia (p34uni1N) 10](#_Toc166953914)

[(Cv1 - p31kolekcia) vyhľadá a načíta knihu podľa id (pomocou find) a vypíše jej autorov + vyhľadá knihy podľa autora 11](#_Toc166953915)

[(Cv2 - p32uniNN) upravte nazov pomocou @JoinTable 12](#_Toc166953916)

[(Cv3 - p33uniN1) Modifikujte: not null stlpec + zmenit nazov stlpca 12](#_Toc166953917)

[(Cv4 - p32uniNN) nájdiAutorov, pocetKnih, nájdiKnihy + podmienky 12](#_Toc166953918)

[(Cv5 - p33uniN1 + p34uni1N) pridajKnihu + vela podmienok 13](#_Toc166953919)

[JPA4 - Obojsmerné asociácie 15](#_Toc166953920)

[1. Obojsmerná 1:N - N:1 (p41biN1) 15](#_Toc166953921)

[2. Obojsmerná N:N - N:N (p42biNN) 16](#_Toc166953922)

[Stratégia dotazovania - Fetch type (EAGER, LAZY) 17](#_Toc166953923)

[(Cv4-4 – 4cv4) 2xObojsmerne + read(String meno) + release(Osoba o) 18](#_Toc166953924)

[(Cv4-5 – 4cv5) Vela tabuliek + find by meno + 20](#_Toc166953925)

[JPA5 - Špecifické situácie 22](#_Toc166953926)

[Kompozícia (OneToMany, p51composit) 22](#_Toc166953927)

[Asociácia 1:1 (OneToOne, p52bi11) 23](#_Toc166953928)

[Mapovanie dvoch tried do jednej tabuľky (kompozicia 1:1, p53embed) 24](#_Toc166953929)

[(Cv5 p53embed) funkcia List findByYear(int year),ktorá pomocou JPQL-dotazu vyhľadá všetky poistné zmluvy pre autá so zadaným rokom výroby (похожее) 24](#_Toc166953930)

[Mapovanie jednej triedy do viacerých tabuliek (p54secondary) 24](#_Toc166953931)

[Zložené kľúče (p55compkey) 25](#_Toc166953932)

[(cv6 p55compkey) Vytvor zlozeny kluc a kompoziciu 25](#_Toc166953933)

[(cv7) Práca s historizovanou tabuľkou, zlozeny kluc ID + TIMESTAMP 26](#_Toc166953934)

[JPA6 - Mapovanie hierarchie tried (p61) 27](#_Toc166953935)

[Mapovanie abstraktnej rodičovskej triedy 28](#_Toc166953936)

[REST2 30](#_Toc166953937)

[Jedalen Post: 30](#_Toc166953938)

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

JPA – читать данные с датабазы

**//jpa1**

@Entity, @Id + @GeneratedValue(strategy = GenerationType.AUTO)

@Transient — nebude namapovaný na žiadny stĺpec tabuľky (jpa1 cv3)

@Enumerated(EnumType.STRING) // varchar or .ORDINAL – int4

enum Dostupnost { NASLKADE, NAOBJEDNAVKU, VYPREDANA };

Зображення, що містить текст, класна дошка, почерк, крейда

Автоматично згенерований опис@Column(nullable=false, unique=true) — **NOT NULL** and **UNIQ** in table

***double (null = 0.0 ) and Double (null = null)***

**//jpa2**

@Temporal(javax.persistence.TemporalType.DATE) — **DATE** without time

@Column(columnDefinition="BOOLEAN") — **DATA TYPE**

@Table(name = "PERSON") + @Column(name = "NAME") - mapovanie triedy na tabulku z roznymy nazovami

@Basic(optional = false) — **NOT NULL** (only var, not in DB)

@Max(value=?) @Min(value=?)// range of your decimal fields | p23jpql

@NamedQueries({ @NamedQuery (name = “Person.findALL” , query = "SELECT p FROM Person p")})— JPQL dotazy

**//jpa3**

@ElementCollection — не делает масив сериализованным, а генерит отдельную таблицу

@+ @JoinColumn(name="VYDAVATEL\_FK", nullable=false) — definovať názov prepojovacieho stĺpca

@+ @JoinColumn(nullable=false) a @ManyToOne(optional=false) — **NOT NULL** ASOCIACIA pre povinný atribút, p33uniN1 cv3Зображення, що містить Шрифт, ряд, текст, знімок екрана

Автоматично згенерований опис

@JoinTable(name = "Book\_Author", joinColumns = @JoinColumn(name = "book\_id"), inverseJoinColumns = @JoinColumn(name = "author\_id")) – vymenit nazov prepojovacej tabulky a stlpcov

**//jpa4**

@OneToMany(mappedBy="vydavatel") - nahrádza @JoinColumn(name) používané v prípade jednosmerných asociácií

@OneToMany(fetch = FetchType.EAGER) — načítali údaje aj s asociovaným objektom stratégia EAGER, LAZY

**//jpa5**

@OneToMany(cascade=CascadeType.REMOVE, orphanRemoval=true) -- **Kompozícia**

— cascade=CascadeType.REMOVE - pri odstránení košíka z databázy budú s ním automaticky odstránené aj všetky položky

— orphaRemoval=true - ak sa položka odstráni z košíka bude nielen vynulovaná FK hodnota v tabuľke KOSIK, ale odstránený aj záznam z tabuľky POLOZKA

@OneToOne - p52bi11

@JoinColumn(nullable=false) -- not null

@JoinColumn(unique=true) -- unique-index pre FK, ktorí nedovolí vložiť viac poistení pre jedno auto

**//jpa6**

@MappedSuperclass -- **Mapovanie abstraktnej rodičovskej triedy**

@Inheritance(strategy = InheritanceType.SINGLE\_TABLE)

# JPA1

### Poznamky relačných diagramov ER (синие табл) + UML diagramov tried OO

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Зображення, що містить текст, Шрифт, знімок екрана, ряд

Автоматично згенерований опис

## (p1jpa)

### create EntityManager

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager(); //create Entity

### čítanie dat z databazy

Query q = em.createNativeQuery("select \* from kniha", Kniha.class);

List<Kniha> knihy = q.getResultList();

for (Kniha k : knihy) {

System.out.println("" + k);

}

### modifikacia dat v databaze

Query q = em.createNativeQuery("select \* from kniha", Kniha.class);

List<Kniha> knihy = q.getResultList();

em.getTransaction().begin();

for (Kniha k : knihy) {

k.setCena(k.getCena() \* 0.8);

}

em.getTransaction().commit();

### vyhľadávanie podľa kľúča

Kniha k = em.find(Kniha.class, "Pipi"); //Pipi je kľúč

### vytvorenie nového záznamu

Kniha k = new Kniha();

k.setNazov("Brave new world");

em.getTransaction().begin();

em.persist(k);

em.getTransaction().commit();

### 

# JPA2 - Entity a JPA anotácie, JPQL

### Generovanie entitnej triedy a entitnej triedy z tabuľky

Зображення, що містить текст, Шрифт, знімок екрана, документ

Автоматично згенерований опис

**Generovanie entitnej triedy z tabuľky (p22 or p21)**

Зображення, що містить текст, знімок екрана, Шрифт, документ

Автоматично згенерований опис

### (p23jpql) JPQL dotazy

#### Pomenovany dotaz:

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

TypedQuery<Person> q = em.createNamedQuery("Person.findAll", Person.class);

List<Person> pl = q.getResultList();

for (Person p : pl) {System.out.println(p.getName() + " narodeny=" + p.getBorn());}

#### Pomenovany dotaz s parametrami:

TypedQuery<Person> q1 = em.createNamedQuery("Person.findByName",Person.class);

q.setParameter("name", "Fero");

#### Dotaz na vyhľadanie osôb bez dátumu narodenia:

TypedQuery<Person> q2 = em.createQuery("select p from Person p where p.born is null", Person.class);

#### Dotazu na zistenie počtu osôb s platom menším ako 1000

TypedQuery<Long> q3 = em.createQuery("select count(p) from Person p where p.salary < 1000",Long.class);

System.out.println("" + q3.getSingleResult());

### (p-- CV2-3) jpql dotazy

EntityManagerFactory emf = Persistence.createEntityManagerFactory("osobyPU");

EntityManager em = emf.createEntityManager();

try {

em.getTransaction().begin();

// JPQL dotaz pre načitanie všetkych osob

TypedQuery<Osoba> queryAll = em.createQuery("SELECT o FROM Osoba o", Osoba.class);

List<Osoba> osoby = queryAll.getResultList();

// Zvýšení platu všech osob o 100

for (Osoba osoba : osoby) {

osoba.setPlat(osoba.getPlat() + 100);

}

// JPQL dotaz pro získání jmen osob s platem nižším než 1000

TypedQuery<String> queryNames = em.createQuery(

"SELECT o.meno FROM Osoba o WHERE o.plat < 1000.0", String.class);

List<String> mena = queryNames.getResultList();

// JPQL dotaz pro získání celkové sumy platů všech osob

TypedQuery<Double> queryTotalSalary = em.createQuery(

"SELECT SUM(o.plat) FROM Osoba o", Double.class);

Double celkovaSumaPlatu = queryTotalSalary.getSingleResult();

em.getTransaction().commit();

} catch (Exception e) {

if (em.getTransaction().isActive()) {

em.getTransaction().rollback();

}

e.printStackTrace();

} finally {

em.close();

emf.close(); }

### (p21) equals, hash, toStraing, clear, detach, refresh, merge, remove

Poznámka: Pre main1 nastavte schema-generation create, pre main2 až main7 nastavte drop-and-create.

#### (main2) p != p2, p.equals(p2) //rozny entity manager

#### (main3) p == p2, p.equals(p2) //1 entity manager

#### (main4) em.clear(); or em.detach(p); — clear context entity manager or detach object // p != p2, p.equals(p2), 1 entity manager

em.clear(); // or em.detach(p);

System.out.println("p == p2 : " + (p == p2)); //false

System.out.println("p.equals(p2): " + p.equals(p2)); //true

#### (main5) em.refresh(p); — обновляет значения с базі данніх

p.setSalary(3000.0);

System.out.println("" + p.getName() + " " + p.getSalary()); //3000

em.refresh(p); // обновляет значения с базі данніх

System.out.println("" + p.getName() + " " + p.getSalary()); //0

#### (main6) em.merge(p); — скопирует р в р2, но обьекті разніе // p != p2, 1 entity manager

em.getTransaction().begin();

Person p2 = em.merge(p); //скопирует р в р2, но обьекті разніе

em.getTransaction().commit();

System.out.println("p == p2 : " + (p == p2)); //false

#### (main7) em.remove(p); — delete from db

em.getTransaction().begin();

em.remove(p); //

em.getTransaction().commit();

#### (P21 cv2-1) logovanie sql-príkazov

Do persistence.xml

<property name="eclipselink.logging.level.sql" value="FINE"/>

<property name="eclipselink.logging.parameters" value="true"/>

### (P22 CV2-2) @Temporal and columnDefinition=” BOOLEAN”

Modifikujte triedu Person a pozrite ako sa prejavia zmeny vo vygenerovanej tabuľke.

- členu married dajte anotáciu @Column(columnDefinition="BOOLEAN") // = bool kak i bez anotacii

- zmeňte typ člena born na LocalDate a odstráňte anotáciu @Temporal // error

- zmeňte typ člena born na LocalDateTime a odstráňte anotáciu @Temporal // error

# JPA3 - Jednosmerné asociácie (a kolekcie)

## @ElementCollection (p31kolekcia)

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований описЗображення, що містить текст, Шрифт, знімок екрана, ряд

Автоматично згенерований опис

## Зображення, що містить текст, Шрифт, знімок екрана, ряд Автоматично згенерований описЗображення, що містить текст, Шрифт, знімок екрана, ряд Автоматично згенерований описJednosmerná N:N asociácia (p32uniNN)

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, квитанція

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, квитанція, алгебра

Автоматично згенерований опис

## Зображення, що містить текст, Шрифт, знімок екрана, ряд Автоматично згенерований описЗображення, що містить текст, знімок екрана, Шрифт, ряд Автоматично згенерований описJednosmerná N:1 asociácia (p33uniN1)

Зображення, що містить текст, Шрифт, знімок екрана, ряд

Автоматично згенерований опис

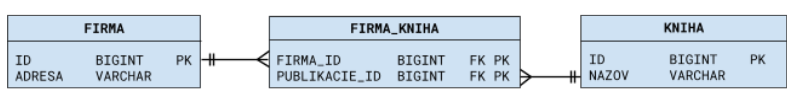
Зображення, що містить текст, квитанція, знімок екрана, Шрифт

Автоматично згенерований опис

## Зображення, що містить текст, Шрифт, знімок екрана, ряд Автоматично згенерований описJednosmerná 1:N asociácia (p34uni1N)

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис



Зображення, що містить текст, знімок екрана, Шрифт, квитанція

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Nutné je však definovať názov prepojovacieho stĺpca, ktorý bude v tabuľke asociovanej triedy.

## (Cv1 - p31kolekcia) vyhľadá a načíta knihu podľa id (pomocou find) a vypíše jej autorov + vyhľadá knihy podľa autora

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

public static void findAndPrintAuthorsById(Long id) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

Kniha kniha = em.find(Kniha.class, id);

if (kniha != null) {

for (String autor : kniha.getAutori())

System.out.println(autor);

} else {

System.out.println("Book with id " + id + " not found.");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

em.close();

}

}

public static void findBooksByAuthor(String author) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

Query query = em.createQuery("SELECT k FROM Kniha k WHERE :author MEMBER OF k.autori");

//SELECT k FROM Kniha k WHERE :author IN (SELECT a FROM k.autori a) query.setParameter("author", author);

List<Kniha> books = query.getResultList();

if (!books.isEmpty()) {

System.out.println("Books by author " + author + ":");

for (Kniha kniha : books) {

System.out.println("- " + kniha.getNazov());

}

} else {

System.out.println("No books found by author " + author);

}

} catch (Exception e) {

e.printStackTrace();

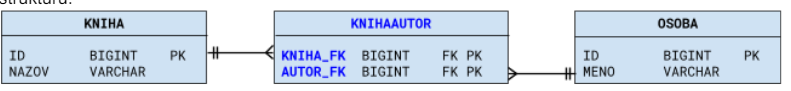
} finally {

em.close();

}

}

## (Cv2 - p32uniNN) upravte nazov pomocou @JoinTable



@ManyToMany

@JoinTable(name = "KNIHAAUTOR", joinColumns = @JoinColumn(name = "KNIHA\_FK"), inverseJoinColumns = @JoinColumn(name = "AUTHOR\_FK"))

private List<Osoba> autori;

## (Cv3 - p33uniN1) Modifikujte: not null stlpec + zmenit nazov stlpca

@ManyToOne(optional=false) + @JoinColumn(nullable=false)

JoinColumn(name="VYDAVATEL")

## (Cv4 - p32uniNN) nájdiAutorov, pocetKnih, nájdiKnihy + podmienky

1. List<Osoba> nájdiAutorov(Long idKnihy), ktorá dostane id knihy vyhľadá ju a vráti zoznam jej autorov. Ak kniha neexistuje, vráti null.

2. int pocetKnih(String menoAutora), ktorá dostane meno osoby a vráti počet kníh, ktorých je autorom. Ak osoba s daným menom neexistuje, vráti 0.

3. List<Kniha> nájdiKnihy(String menoAutora), ktorá dostane meno osoby, vyhľadá a vráti zoznam kníh, ktorých je autorom. Ak osoba s daným menom neexistuje, vráti null.

public static List<Osoba> nájdiAutorov(Long idKnihy){

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

Kniha kniha = em.find(Kniha.class, idKnihy);

if (kniha != null) {

return kniha.getAutori();

} else {

return null;

}

} catch (Exception e) {

e.printStackTrace();

} finally {

em.close();

}

return null;

}

public static int pocetKnih(String menoAutora){

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

Query query = em.createQuery("SELECT COUNT(k) FROM Kniha k JOIN k.autori a WHERE a.meno = :menoAutora");

query.setParameter("menoAutora", menoAutora);

Long count = (Long) query.getSingleResult();

return count.intValue();

} catch (Exception e) {

e.printStackTrace();

return 0;

} finally {

em.close();

}

}

public static List<Kniha> nájdiKnihy(String menoAutora) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

Query query = em.createQuery("SELECT k FROM Kniha k JOIN k.autori a WHERE a.meno = :menoAutora");

query.setParameter("menoAutora", menoAutora);

List<Kniha> knihy = query.getResultList();

return knihy;

} catch (Exception e) {

e.printStackTrace();

return null;

} finally {

em.close(); } }

## (Cv5 - p33uniN1 + p34uni1N) pridajKnihu + vela podmienok

void pridajKnihu(String nazov, String adresa) , ktorá dostane názov knihy a adresu firmy, ktorá je jej vydavateľom. Funkcia…

● najprv overí či kniha s daným názvom, resp. vydavateľstvo s danou adresou už v DB existujú

● ak vydavateľstvo s danou adresou v DB ešte neexistuje, vytvorí ho,

● ak kniha s daným názvom v DB ešte neexistuje, vytvorí ju a priradí jej zadané vydavateľstvo (zaradí ju medzi publikácie vydavateľstva).

● ak kniha s daným názvom už v DB existovala neurobí nič.

public static void pridajKnihu(String nazov, String adresa) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

// Проверяем существует ли уже книга с указанным названием

Query knihaQuery = em.createQuery("SELECT k FROM Kniha k WHERE k.nazov = :nazov");

knihaQuery.setParameter("nazov", nazov);

List<Kniha> existingBooks = knihaQuery.getResultList();

if (existingBooks.isEmpty()) {

// Проверяем существует ли уже издательство с указанной адресой

Query firmaQuery = em.createQuery("SELECT f FROM Firma f WHERE f.adresa = :adresa");

firmaQuery.setParameter("adresa", adresa);

List<Firma> existingFirmas = firmaQuery.getResultList();

Firma vydavatel;

if (existingFirmas.isEmpty()) {

// Если издательства с таким адресом нет, создаем новое

vydavatel = new Firma();

vydavatel.setAdresa(adresa);

em.persist(vydavatel);

} else {

// Если издательство с таким адресом уже есть, используем его

vydavatel = existingFirmas.get(0);

}

// Создаем новую книгу и связываем ее с издательством

Kniha kniha = new Kniha();

kniha.setNazov(nazov);

kniha.setVydavatel(vydavatel);

em.persist(kniha);

em.getTransaction().commit();

System.out.println("Книга успешно добавлена.");

} else {

// Если книга уже существует, ничего не делаем

System.out.println("Книга с таким названием уже существует.");

}

} catch (Exception e) {

e.printStackTrace();

em.getTransaction().rollback();

} finally {

em.close();

}

}

public static void pridajKnihu(String nazov, String adresa) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

// Проверяем существует ли уже книга с указанным названием

Query knihaQuery = em.createQuery("SELECT k FROM Kniha k WHERE k.nazov = :nazov");

knihaQuery.setParameter("nazov", nazov);

List<Kniha> existingBooks = knihaQuery.getResultList();

if (existingBooks.isEmpty()) {

// Проверяем существует ли уже издательство с указанной адресой

Query firmaQuery = em.createQuery("SELECT f FROM Firma f WHERE f.adresa = :adresa");

firmaQuery.setParameter("adresa", adresa);

List<Firma> existingFirmas = firmaQuery.getResultList();

Firma vydavatel;

if (existingFirmas.isEmpty()) {

// Если издательства с таким адресом нет, создаем новое

vydavatel = new Firma();

vydavatel.setAdresa(adresa);

em.persist(vydavatel);

} else {

// Если издательство с таким адресом уже есть, используем его

vydavatel = existingFirmas.get(0);

}

// Создаем новую книгу

Kniha kniha = new Kniha();

kniha.setNazov(nazov);

// Связываем книгу с издательством

List<Kniha> publikacie = vydavatel.getPublikacie();

if (publikacie == null) {

publikacie = new ArrayList<>();

}

publikacie.add(kniha);

vydavatel.setPublikacie(publikacie);

// Сохраняем изменения в базе данных

em.persist(kniha);

em.getTransaction().commit();

System.out.println("Книга успешно добавлена.");

} else {

// Если книга уже существует, ничего не делаем

System.out.println("Книга с таким названием уже существует.");

}

} catch (Exception e) {

e.printStackTrace();

em.getTransaction().rollback();

} finally {

em.close();

}

}

# JPA4 - Obojsmerné asociácie

## Obojsmerná 1:N - N:1 (p41biN1)

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований описЗображення, що містить текст, Шрифт, знімок екрана, ряд

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, ряд, Шрифт

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований описJPA interpretuje referencie v triedach ako dve nezávislé jednosmerné asociácie. Obojsmerná asociácia je však tvorená dvoma jednosmernými, ktoré sú navzájom inverzné.

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

@OneToMany(mappedBy nahrádza @JoinColumn(name) používané v prípade jednosmerných asociácií

Dátový člen, ktorý je na FK stĺpec namapovaný, asociáciu vlastní, definuje a označuje sa owner. (V našom príklade je to vydavatel v triede Kniha.)

Zhrnutie: Parameter mappedBy prepája dátové členy na oboch stranách asociácie

● určuje, ktorý dátový člen referencovanej triedy je owner asociácie t.j. predstavuje inverznú asociáciu.

● používa sa len pri obojsmerných asociáciách a to na strane, ktorá nie je jej vlastníkom

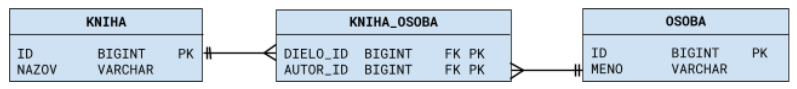
Poznámka: Všimnite si, že mappedBy udáva meno atribútu triedy, na rozdiel od anotácie @JoinColumn, ktorá udáva meno FK stĺpca.

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Poznámka: Pri vytváraní asociácie sme zadali referencie na oboch stranách asociácie. Pre správne vytvorenie hodnoty FK v databáze však stačí zadať len owner stranu t.j. setVydavatel.

## Зображення, що містить текст, Шрифт, знімок екрана, ряд Автоматично згенерований опис2. Obojsmerná N:N - N:N (p42biNN)



Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

## Stratégia dotazovania - Fetch type (EAGER, LAZY)

Vsetko naraz:

Зображення, що містить текст, знімок екрана, Шрифт, інформація

Автоматично згенерований опис

Dotaz na asociovanú entitu t.j. tabuľku KNIHA vyšle až neskôr volanie metódy getPublikacie():

Зображення, що містить текст, знімок екрана, Шрифт, алгебра

Автоматично згенерований опис

Зображення, що містить текст, Шрифт, знімок екрана, білий

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, схема, Паралель

Автоматично згенерований опис

## (Cv4-4 – 4cv4) 2xObojsmerne + read(String meno) + release(Osoba o)

Зображення, що містить текст, знімок екрана, Шрифт, лист

Автоматично згенерований опис

public static void create() {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("triedyPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

// Создаем предметы

Predmet predmet1 = new Predmet();

predmet1.setNazov("Predmet 1");

predmet1.setRocnik("1");

predmet1.setKredity(3);

Predmet predmet2 = new Predmet();

predmet2.setNazov("Predmet 2");

predmet2.setRocnik("2");

predmet2.setKredity(5);

// Создаем людей

Osoba osoba1 = new Osoba();

osoba1.setMeno("Osoba 1");

osoba1.setTitul(Titul.Doc);

Osoba osoba2 = new Osoba();

osoba2.setMeno("Osoba 2");

osoba2.setTitul(Titul.Prof);

Osoba osoba3 = new Osoba();

osoba3.setMeno("Osoba 3");

osoba3.setTitul(Titul.Prof);

// Устанавливаем связи

predmet1.setPrednasajuci(osoba1);

predmet2.setPrednasajuci(osoba2);

predmet2.setAsistent(new ArrayList<>());

predmet2.getAsistent().add(osoba2);

predmet2.getAsistent().add(osoba3);

em.persist(predmet1);

em.persist(predmet2);

em.persist(osoba1);

em.persist(osoba2);

em.persist(osoba3);

em.getTransaction().commit();

} catch (Exception e) {

if (em.getTransaction().isActive()) {

em.getTransaction().rollback();

}

e.printStackTrace();

} finally {

em.close();

}

}

public static void read(String meno) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("triedyPU");

EntityManager em = emf.createEntityManager();

try {

TypedQuery<Osoba> q = em.createQuery("SELECT o FROM Osoba o WHERE o.meno = :meno", Osoba.class).setParameter("meno", meno);

Osoba osoba = q.getSingleResult();

List<Predmet> predmety = osoba.getCvicenie(); //predmety.isEmpty()

osoba.getCvicenie().size();

System.out.println(osoba.getCvicenie());

for (Predmet predmet : predmety) {

System.out.println("Predmet: " + predmet.getNazov() + ", Prednasajuci: " + predmet.getPrednasajuci().getMeno());

}

} catch (Exception e) {

e.printStackTrace();

} finally {

em.close();

}

}

public static void release(Osoba o) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("triedyPU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

try {

Osoba osoba = em.merge(o);

em.remove(osoba);

em.getTransaction().commit();

} catch (Exception e) {

if (em.getTransaction().isActive()) {

em.getTransaction().rollback();

}

e.printStackTrace();

} finally {

em.close();

}

}

## (Cv4-5 – 4cv5) Vela tabuliek + find by meno +

Зображення, що містить текст, Шрифт, ряд, схема

Автоматично згенерований опис

#### ● Implementujte funkciu, ktorá vyhľadá knihu podľa isbn.

public static void findByISBN(int isbn) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("triedyPU");

EntityManager em = emf.createEntityManager();

try {

TypedQuery<Kniha> query = em.createQuery("SELECT k FROM Kniha k WHERE k.isbn = :isbn", Kniha.class);

query.setParameter("isbn", isbn);

Kniha book = query.getSingleResult();

System.out.println("Book with ISBN " + isbn + ": " + book.getNazov());

} catch (Exception e) {

e.printStackTrace();

} finally {

em.close();

}

}

#### ● Implementujte funkciu, ktorá vyhľadá autora podľa mena.

public static Autor findAuthorByName(String name) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("triedyPU");

EntityManager em = emf.createEntityManager();

try {

TypedQuery<Autor> query = em.createQuery("SELECT a FROM Autor a WHERE a.meno = :meno", Autor.class);

query.setParameter("meno", name);

return query.getSingleResult();

} catch (Exception e) {

e.printStackTrace();

} finally {

em.close();

}

return null;

}

#### ● Implementujte funkciu, ktorá načíta z databázy všetky obchody a pre každý obchod vypíše všetky údaje o knihách (aj autoroch) v ponuke.

public static void listBooksInStores() {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("triedyPU");

EntityManager em = emf.createEntityManager();

try {

TypedQuery<Obchod> query = em.createQuery("SELECT o FROM Obchod o", Obchod.class);

List<Obchod> stores = query.getResultList();

for (Obchod store : stores) {

System.out.println("Store: " + store.getNazov());

List<Polozka> items = store.getPolozky();

for (Polozka item : items) {

Kniha book = item.getKniha();

System.out.println("Book: " + book.getNazov());

System.out.println("Authors: ");

for (Autor author : book.getAutori()) {

System.out.println("- " + author.getMeno());

}

System.out.println("Price: " + item.getCena());

}

System.out.println();

}

} catch (Exception e) {

e.printStackTrace();

} finally {

em.close();

}

}

#### ● Implementujte funkciu, ktorá dostane ako argument objekt triedy Autor a vráti zoznam všetkých obchodov (objektov triedy Obchod), ktoré ponúkajú knihu od daného autora (t.j. je jedným z jej autorov).Funkcia môže predpokladať, že objekt, ktorý dostane ako argument je manažovaný, t.j. bol načítaný z databázy.

public static List<Obchod> findStoresByAuthor(Autor author) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("triedyPU");

EntityManager em = emf.createEntityManager();

try {

TypedQuery<Obchod> query = em.createQuery("SELECT o FROM Obchod o JOIN o.polozky p WHERE p.kniha IN (SELECT k FROM Kniha k JOIN k.autori a WHERE a = :author)", Obchod.class);

query.setParameter("author", author);

List<Obchod> stores = query.getResultList();

//

// List<Obchod> stores = new ArrayList<>();

// for (Polozka item : items) {

// stores.add(item.getObchod());

// }

return stores;

} catch (Exception e) {

e.printStackTrace();

} finally {

em.close();

}

return null;

}

#### ● Implementujte program, ktorý otestuje funkčnosť oboch funkcií. Pre naplnenie databázy testovacími údajmi implementujte funkciu, ktorá importuje do DB dáta zo súboru data4cv45.csv.

// public static void main(String[] args) {

// importDataFromCSV("data4cv45.csv");

// testFunctionality();

// }

//

// public static void importDataFromCSV(String filename) {

// EntityManager em = // получите EntityManager

// EntityTransaction tx = em.getTransaction();

//

// try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

// String line;

// while ((line = br.readLine()) != null) {

// String[] data = line.split(","); // предполагаем, что данные разделены запятыми

// // создайте объекты и сохраните их в базу данных

// // например:

// // Kniha book = new Kniha();

// // book.setIsbn(Integer.parseInt(data[0]));

// // book.setNazov(data[1]);

// // ...

// // tx.begin();

// // em.persist(book);

// // tx.commit();

// }

// } catch (Exception e) {

// e.printStackTrace();

// } finally {

// em.close();

// }

// }

//public void importDataFromCSV(String filename) {

// try (BufferedReader br = new BufferedReader(new FileReader(filename))) {

// String line;

// while ((line = br.readLine()) != null) {

// String[] data = line.split(",");

// if (data.length >= 6) {

// Kniha kniha = new Kniha();

// kniha.setIsbn(Integer.parseInt(data[0]));

// kniha.setNazov(data[1]);

//

// Autor autor = new Autor();

// autor.setMeno(data[2]);

//

// Vydavatel vydavatel = new Vydavatel();

// vydavatel.setNazov(data[3]);

//

// Polozka polozka = new Polozka();

// polozka.setCena(Double.parseDouble(data[4]));

// polozka.setKniha(kniha);

//

// Obchod obchod = new Obchod();

// obchod.setNazov(data[5]);

// obchod.setPolozky(Arrays.asList(polozka)); // Преобразуем одиночный элемент в список

//

// // Сохраняем объекты в базу данных

// EntityManager em = entityManagerFactory.createEntityManager();

// try {

// em.getTransaction().begin();

// em.persist(kniha);

// em.persist(autor);

// em.persist(vydavatel);

// em.persist(polozka);

// em.persist(obchod);

// em.getTransaction().commit();

// } finally {

// em.close();

// }

// }

// }

// } catch (IOException e) {

// e.printStackTrace();

// }

//}

# JPA5 - Špecifické situácie

## Kompozícia (OneToMany, p51composit)

Зображення, що містить текст, Шрифт, знімок екрана, ряд

Автоматично згенерований опис

Зображення, що містить текст, квитанція, Шрифт, знімок екрана

Автоматично згенерований опис

Зображення, що містить текст, Шрифт, квитанція, ряд

Автоматично згенерований опис

Parameter cascade=CascadeType.REMOVE zabezpečí, že pri odstránení košíka z databázy budú s ním automaticky z databázy odstránené aj všetky položky.

Parameter orphaRemoval=true zabezpečí, že ak sa položka odstráni z košíka (zo zoznamu polozky) bude nielen vynulovaná FK hodnota v tabuľke KOSIK, ale odstránený aj záznam z tabuľky POLOZKA. (samotné CascadeType.REMOVE nestačí)(cv2 p51composit) + pridajTovar(Kosik k, Tovar t) + odoberTovar(Kosik k, Tovar t)

Зображення, що містить текст, знімок екрана, ряд, Шрифт

Автоматично згенерований опис

Implementujte dve funkcie:

● pridajTovar(Kosik k, Tovar t)

#### Pridá tovar do košíka: Ak tovar v košíku ešte nie je, pridá novú položku. Ak už v košíku tovar bol, zvýši jeho množstvo.

public static void pridajTovar(Kosik ka, Tovar t){

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

Kosik k = em.find(Kosik.class, ka.getId());

k.setSuma(ka.getSuma());

List<Polozka> polozky= k.getPolozka();

//Polozka p=polozky.forEach(p ->if(p.getTovar().equals(t)){break;return p;})

Polozka p2=null;

for(Polozka p:polozky){

if(p.getTovar().equals(t)){

p2=p;

break;

}

}

em.getTransaction().begin();

if (p2!=null){

p2.setMnozstvo(p2.getMnozstvo()+1);

}

else{

p2=new Polozka();

p2.setMnozstvo(1);

p2.setTovar(t);

p2.setKos(k);

k.getPolozka().add(p2);

em.persist(p2);

em.persist(t);

}

k.setSuma(k.getSuma()+t.getCena());

em.getTransaction().commit();

}

● odoberTovar(Kosik k, Tovar t)

#### Odoberie tovar z košíka, t.j zníži množstvo tovaru o 1. Ak klesne množstvo na 0, odstráni položku. Ak už košík nebude obsahovať žiadne položky, odstráni ho z databázy

public static void odoberTovar(Kosik ka, Tovar t){

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

Kosik k = em.find(Kosik.class, ka.getId());

List<Polozka> polozky= k.getPolozka();

//Polozka p=polozky.forEach(p ->if(p.getTovar().equals(t)){break;return p;})

Polozka p2=null;

for(Polozka p:polozky){

if(p.getTovar().equals(t)){

p2=p;

break;

}

}

em.getTransaction().begin();

if (p2!=null){

if(p2.getMnozstvo()==1){

if(k.getPolozka().size()==1){

em.remove(k);

}

else{

k.getPolozka().remove(p2);

}

em.remove(p2);

}

else{

p2.setMnozstvo(p2.getMnozstvo()-1);

}

k.setSuma(ka.getSuma()-t.getCena());

}

em.getTransaction().commit();

}

}

## Зображення, що містить текст, електроніка, знімок екрана, Шрифт Автоматично згенерований описAsociácia 1:1 (OneToOne, p52bi11)

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Parameter nullable=false v anotáciu referencie na vozidlo zabezpečuje, že poistenie musí mať zadané vozidlo (v databáze), a ko vyžaduje v UML kardinalita 1 na strane vozidla

@JoinColumn(unique=true)

## Mapovanie dvoch tried do jednej tabuľky (kompozicia 1:1, p53embed)

Зображення, що містить текст, Шрифт, знімок екрана, число

Автоматично згенерований описЗображення, що містить текст, Шрифт, знімок екрана, ряд

Автоматично згенерований опис

@Embedded @Embeddable

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

### (Cv5 p53embed) funkcia List findByYear(int year),ktorá pomocou JPQL-dotazu vyhľadá všetky poistné zmluvy pre autá so zadaným rokom výroby (похожее)

static List<Poistenie> readFromYear(){

EntityManagerFactory emf = Persistence.createEntityManagerFactory("vsaPU");

EntityManager em = emf.createEntityManager();

TypedQuery<Poistenie> q = em.createQuery("select p from Poistenie p where p.vozidlo.rokVyroby=1999",Poistenie.class);

return q.getResultList();

}

## Mapovanie jednej triedy do viacerých tabuliek (p54secondary)

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований описЗображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований описЗображення, що містить текст, Шрифт, знімок екрана, число

Автоматично згенерований опис

## Zložené kľúče (p55compkey)

Зображення, що містить текст, Шрифт, знімок екрана, число

Автоматично згенерований описЗображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

@Embeddable

### (cv6 p55compkey) Vytvor zlozeny kluc a kompoziciu

Зображення, що містить текст, знімок екрана, Шрифт, схема

Автоматично згенерований опис

### (cv7) Práca s historizovanou tabuľkou, zlozeny kluc ID + TIMESTAMP

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

# JPA6 - Mapovanie hierarchie tried (p61)

@MappedSuperclass -- **Mapovanie abstraktnej rodičovskej triedy**

@Inheritance(strategy = InheritanceType.SINGLE\_TABLE)

* SINGLE TABLE - je default
* JOINED
* TABLE PER CLASS

Зображення, що містить текст, Шрифт, ряд, почерк

Автоматично згенерований описЗображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований описЗображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований описЗображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

## Mapovanie abstraktnej rodičovskej triedy

Ak je niektorá trieda v hierarchii abstraktná trieda, nie je možné vytvárať jej inštancie a teda nie je ani potrebné vytvárať pre ňu tabuľku. V prípade stratégie TABLE\_PER\_CLASS sa ani nevytvorí

Зображення, що містить текст, Шрифт, знімок екрана

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

**Výber stratégie**

Z hľadiska použitia je asi najjednoduchšia stratégia SINGLE\_TABLE. Nevýhoda je, že v prípade veľkého počtu atribútov špecifických pre podtriedy, vznikne tabuľka s veľkým počtom stĺpcov, ktoré sú však zväčša prázdne.

Pre rozhodovanie medzi stratégiami JOINED a TABLE\_PER\_CLASS je rozhodujúci charakter dotazov, aké budú prevažovať pri vyhľadávaní a čítaní údajov z databázy.

Ilustrujú to nasledujúce jednoduché volania metód entity managera, kde v logu môžeme sledovať vygenerované SQL dotazy.

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

# REST2

@Path("generic")

@RequestScoped

public class GenericResource {

@Context

private UriInfo context;

public GenericResource() {

}

// Skuste v browseri: http://localhost:8080/rs1param/resources/generic/1995

@GET

@Path("{id}")

@Produces("text/plain")

public String getName(@PathParam("id") int id) {

//TODO return proper representation object

return "ID = " + (id);

}

// Skuste v browseri: http://localhost:8080/rs1param/resources/generic/par?meno=Jano

@GET

@Path("par")

@Produces("text/plain")

public String getParName(@QueryParam("meno") String name) {

//TODO return proper representation object

return "Hello " + name;

}

// Skuste v browseri: http://localhost:8080/rs1param/resources/generic/num?cislo=333

@GET

@Path("/num")

@Produces("text/plain")

public String getText(@DefaultValue("1111")

@QueryParam("cislo") int num) {

//TODO return proper representation object

return "poslali ste cislo " + num;

}

// Skuste v browseri: http://localhost:8080/rs1param/resources/generic/hlavicka

@GET

@Path("hlavicka")

@Produces("text/html")

public String getHeader(@Context HttpHeaders hh) {

MultivaluedMap<String, String> headerParams = hh.getRequestHeaders();

//Map<String, Cookie> pathParams = hh.getCookies();

String resp = "";

for (String key : headerParams.keySet()) {

resp = resp + key + " = " + headerParams.getFirst(key) + "<br/>";

}

return resp;

}

// Skuste v browseri: http://localhost:8080/rs1param/resources/generic

@GET

@Produces(MediaType.TEXT\_PLAIN)

public String getText(@Context ServletContext ctx) {

return "" + ctx.getAttribute("sprava");

}

@PUT

@Consumes(MediaType.TEXT\_PLAIN)

public void putText(@Context ServletContext ctx, String msg) {

ctx.setAttribute("sprava", msg);

}

}

### Jedalen Post:

private List<String> jedla; //Set<String> для уникальных, Map<Integer, String> для ключ-значение

private Set<String> jedlaSet;

private Map<Integer, String> jedlaMap;

public JedalenResource() {

jedla = new ArrayList<>();

jedla.add("Vyprazany syr");

**jedlaMap = new HashMap<>();**

**jedlaMap.put(1, "Vyprazany syr");**

}

@PUT

@Path("{cislo}")

@Consumes(MediaType.TEXT\_PLAIN)

@Produces(MediaType.TEXT\_PLAIN)

public String putJedlo(@PathParam("cislo") int n, String name) {

jedlaMap.put(n, name);

return "Success";

}

@DELETE

@Path("{cislo}")

public void deleteJedlo(@PathParam("cislo") int n) {

jedlaMap.remove(n);

}

@GET

@Path("menu")

@Produces(MediaType.TEXT\_PLAIN)

public String getMenu() {

StringBuilder sb = new StringBuilder();

for (Map.Entry<Integer, String> entry : jedlaMap.entrySet()) {

sb.append(entry.getKey()).append(": ").append(entry.getValue()).append("\n");

}

return sb.toString();

}