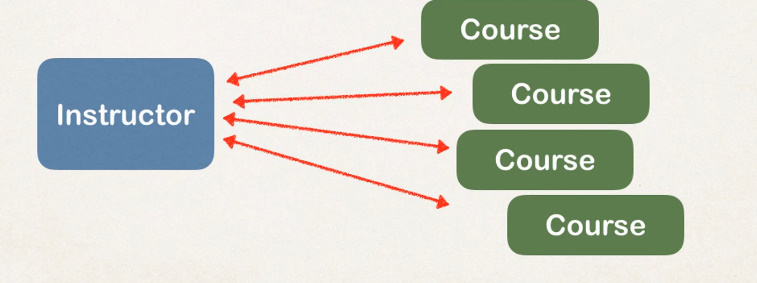
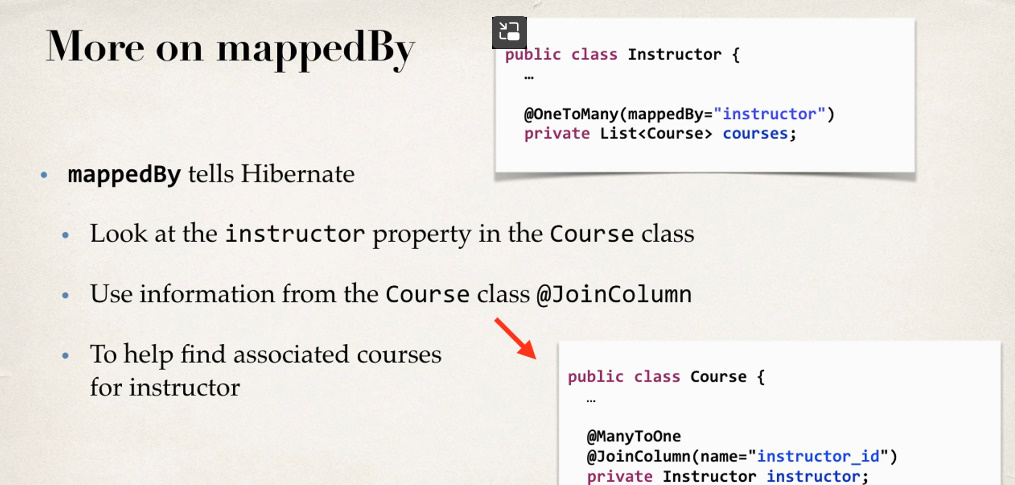
* OneToMany este bidirectionala



Adica putem incepe cu un curs si adauga un instructor, sau dauga un instructor si apoi prin el cursuri.

* Many to one e identic, adica mai multe cursuri pot avea un instructor, si presupunem ca fiecare curs e dus doar de un profesor, ca sa avem one to many
* Aici de ex nu trebuie sa aplicam cascading delete.

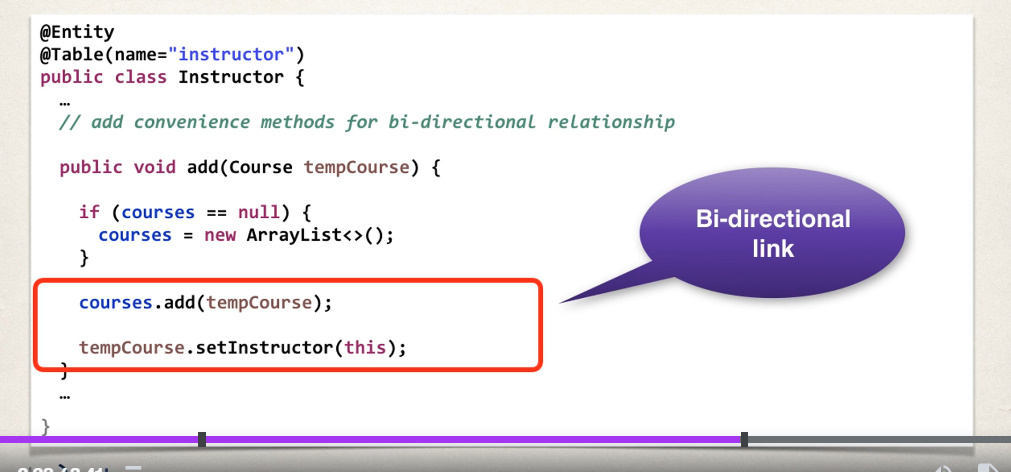
**Create**



Aici adaugam o conexiunea intre Instructor si Course, prin @OneToMany(mappedBy=”instructor”), si o conexiune intre Course si Instructor prin foreign key “instructor\_id”. Asa avem o conexiunea bidirectionala. In plus, Instructor se va uita la fiecare course si va accesa fieldul instructor pentru ca

Instructorul poate avea mai multe cursuri, deci nu putem afisa toate cursurile unui instructor in tabelul sau, de aceea nu cream nici-un atribut pentru cursuri, dar un curs poate avea doar un instructor, si deci poate avea un atribut pentru el. In schimb, Instructor class poate sa creeze o lista de cursuri, care sa ne ajute, desi in tabel asa ceva nu ar fi posibil. Asta si spune @OneToMany, un instructor va avea mai multe referinte la cursuri. **Deci, cand se va crea Instructor, Hibernate automat va extrage toate obiectele curs corespunzatoare si le va adauga in lista.**

Acum cream legaturaa bidirectionala, pentru a putea adauga cursuri manual.



Daca nu vom face asta se va intampla aceasta situatie neplacuta:

Sa zicem ca cream instructor

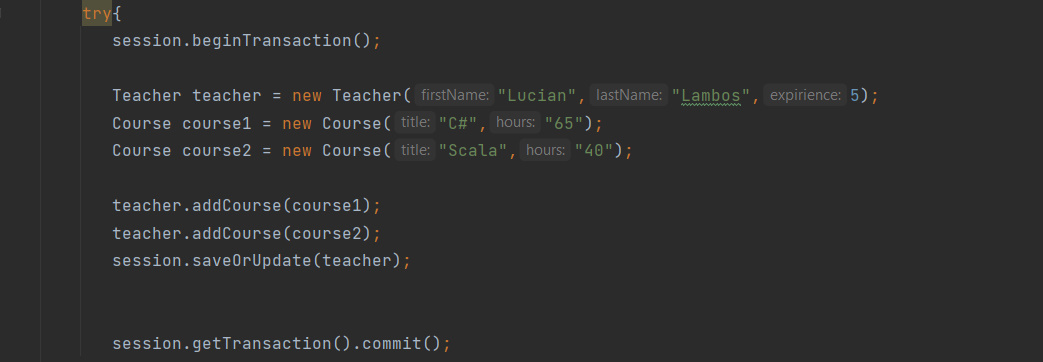
Acum cream course1 si course2

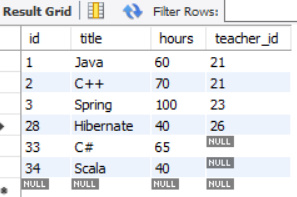
Lista de courses a lui instructor va fi cascade save, deci daca salvam un instructor, si lista sa de cursuri va fi salvata cu el

Acum, adaugam in lista de cursuri a lui instructor course1 si course2

Nu am adaugat acest instructor in course1 si nici in course2 cu course1/2.setInstructor(instructor)

Acum, daca salvam doar instructor, evident, datorita lui Cascade, se vor salva si cursurile, dar problema e ca cursurile vor avea la foreign key la instructor column NULL. Hibernate nu a salvat automat si id la instructor pentru cursuri, chiar daca cursurile au fost puse in lista lui instructor.





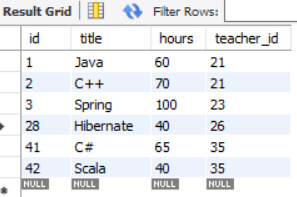


Situatia data neplacuta poate fi rezolvata si prin inlocuirea lui mappedBy cu @JoinColumn pentru a crea legatura bidirectionala. Asa cum in JoinColumn vom specifica coloana in care Teacher e foreign key in Course, in fiecare curs din lista de cursuri a lui teacher Hibernate va pune id a lui teacher:

Teacher Class:

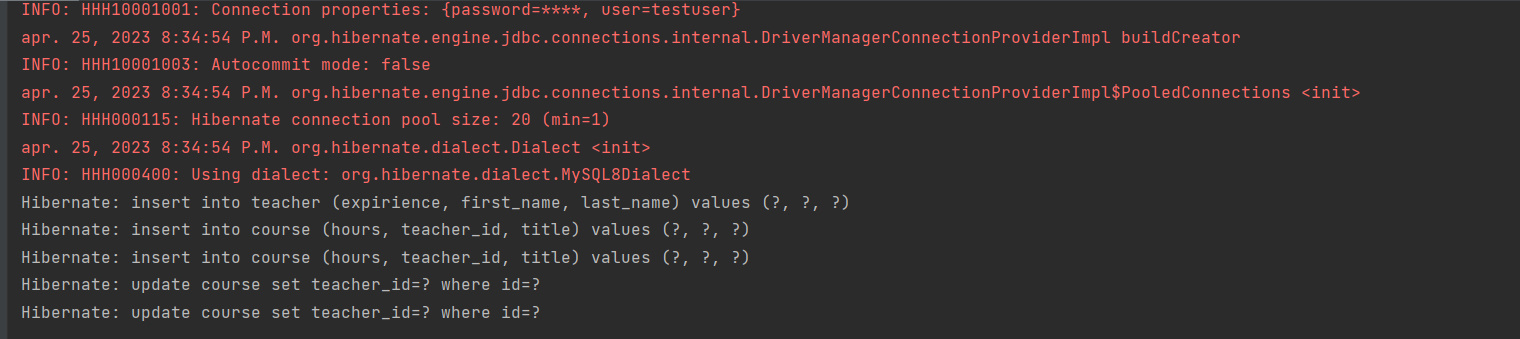
@OneToMany  
@Cascade(value = org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)  
@JoinColumn(name = "teacher\_id")  
private List<Course> courses;

tot codul de sus, dar vedem ceva interesant:



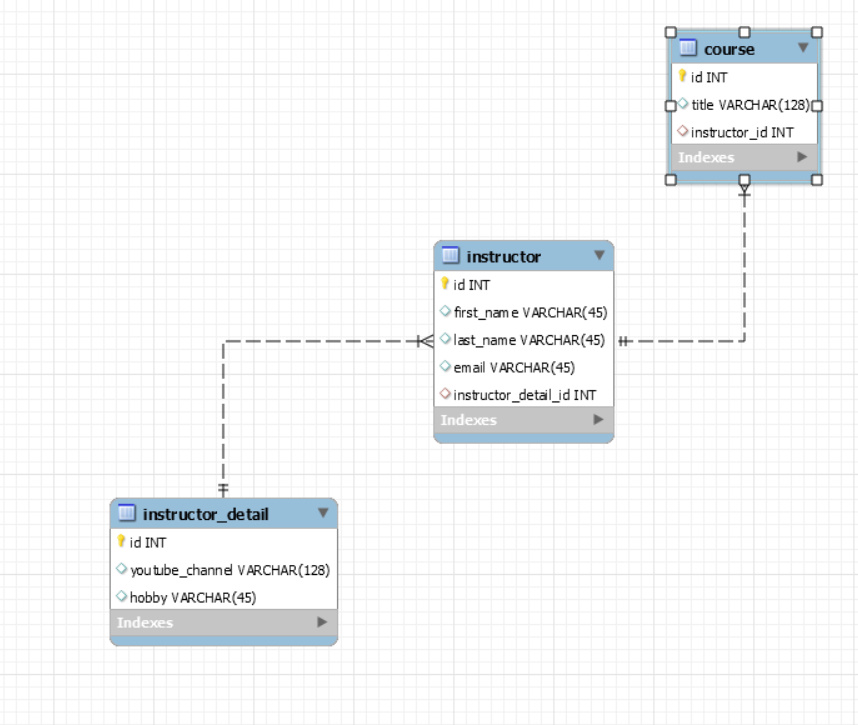


**Si vedem ca Hibernate a adaugat acest foreign key:**

****

Evident, asta va merge doar daca foreign key accepta null in curs, ca sa dea mai apoi update la forein key din null in altceva.

Schema va fi asa:



@Entity  
@Table(name="instructor\_detail")  
public class InstructorDetail {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 private int id;  
  
 @Column(name="youtube\_channel")  
 private String youtubeChannel;  
  
 @Column(name="hobby")  
 private String hobby;  
  
 @OneToOne(mappedBy = "instructorDetail", cascade = {CascadeType.*DETACH*, CascadeType.*MERGE*, CascadeType.*PERSIST*, CascadeType.*REFRESH*})  
 private Instructor instructor;

Clasele copii(fara foreign key) vor avea nevoie de mappedBy, pentru a sti la care foreign key sa se uite din tabelul parinte. Instructor\_detail are nevoie de mappedBy pentru a se uita la entity Instructor si sa caute fieldul “instructorDetail” ca sa stie ca el e foreign key si sa caute obiecte dupa el identic cu primary key a lui.

@Entity  
@Table(name = "course")  
public class Course {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 private int id;  
  
 @Column(name = "title")  
 private String title;  
  
 @ManyToOne(cascade = {CascadeType.*DETACH*,CascadeType.*MERGE*,CascadeType.*PERSIST*,CascadeType.*REFRESH*})  
 @JoinColumn(name="instructor\_id")  
 private Instructor instructor;

Nu are nevoie de mapped, asa cum stie bine ca el are foreign key si deci va trebuie sa o caute prin primary key la Instructor.

@Entity  
@Table(name="instructor")  
public class Instructor {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 private int id;  
  
 @Column(name="first\_name")  
 private String firstName;  
  
 @Column(name = "last\_name")  
 private String lastName;  
  
 @Column(name = "email")  
 private String email;  
  
 @OneToOne(cascade = CascadeType.*ALL*)  
 @JoinColumn(name="instructor\_detail\_id")  
 private InstructorDetail instructorDetail;  
  
 @OneToMany(mappedBy = "instructor")  
 private List<Course> course;

Course List are nevoie de mapped, caci trebuie sa stie care e foreign key la care sa se uite cand va cauta cursuri cu id identic cu primary key al sau, si e List deoarece asa cum avem oneToMany, inseamna ca pot fi returnare mai multe cursuri,care au acelasi instructor, si deci e nevoie de o Lista.

Si la final, pentru a adauga manual cursuri

public void add(Course tempCourse){  
 if(courses == null){  
 courses = new ArrayList<Course>();  
 }  
 courses.add(tempCourse);  
 tempCourse.setInstructor(this);  
}

Apoi cream un Instructor si descrierea sa:

session.beginTransaction();  
  
Instructor instructor = new Instructor("Mititiuc","Eduard","ediutsu2002@mail.ru");  
InstructorDetail instructorDetail = new InstructorDetail("AlfaLight","Programming");  
  
instructor.setInstructorDetail(instructorDetail);  
  
session.save(instructor);  
  
session.getTransaction().commit();

Acum cream niste cursuri:

Instructor instructor = session.get(Instructor.class,6);  
 Course course1 = new Course("Spring");  
 Course course2 = new Course("Hibernate");  
  
 instructor.add(course1);  
 instructor.add(course2);  
  
 session.save(course1);  
 session.save(course2);

Sau putem face totul deodata:

Instructor instructor = new Instructor("Mititiuc","Eduard","ediutsu2002@mail.ru");  
InstructorDetail instructorDetail = new InstructorDetail("AlfaLight","Programming");  
 instructor.setInstructorDetail(instructorDetail);  
  
 Course course1 = new Course("Spring");  
Course course2 = new Course("Hibernate");  
  
instructor.add(course1);  
instructor.add(course2);  
  
session.save(instructor);  
session.save(course1);  
session.save(course2);

**Mare grija sa salvam in sesiune si instructorul daca nu e cascading pus!!!Si mare grija la ordinea de salvare daca nu e Cascading!**

**Pentru a obtine toate cursurile unui istructor:**

Instructor instructor = session.get(Instructor.class,7);  
  
System.*out*.println(instructor.getCourses());

**@Cascade**

Avem o problema cand lucram cu Hibernate. Daca de exemplu avem entitatile Course si Techer, si Course are drept foreign key primary key de la teacher, avem problema. Daca cream un teacher si un course, si apoi in course adaugam acel profesor, si desi avem la Course

@ManyToOne(cascade = {CascadeType.*MERGE*,CascadeType.*PERSIST*,CascadeType.*DETACH*,CascadeType.*REFRESH*}

daca salvam cu session.save(course), nu se va salva si teacher, si asa primim eroare. Chiar de mai punem si o lista de cursuri la profesor si folosim mappedBy, oricum se va salva profesorul fara cursuri. Problema o putem rezolva cu CascadeType.ALL, insa nu e bine ca se si sterg bidirectional.Unica solutie e aceasta:

@Cascade(org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)

Asta punem deasupra la obiectul care trebuie sa se salveze impreuna cu entitatea. Asa, se repara neajunsul:

@Cascade(org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)  
@OneToMany(mappedBy = "teacher",cascade = {CascadeType.*MERGE*,CascadeType.*PERSIST*,CascadeType.*DETACH*,CascadeType.*REFRESH*})  
private List<Course> courseList = new LinkedList<Course>();

sau

@Cascade(org.hibernate.annotations.CascadeType.*SAVE\_UPDATE*)  
@ManyToOne()  
@JoinColumn(name = "teacher\_id")  
private Teacher teacher;

**@JoinColumn alterntiva pentru mappedBy**

* mappedBy poate fi inlocuit cu @JoinColumn(name=”nume coloana”);
* Daca un teacher are mai multe cursuri, si un curs doar un teacher, e logic ca course va avea Foreign key, dar nu si teacher. In mod normal, mappedBy e folosi asa:

@OneToMany(fetch = FetchType.*LAZY*,mappedBy = "teacher",cascade = CascadeType.*DETACH*)  
private List<Course> courseList;

Totusi, aceasta e echivalent cu:

@OneToMany(fetch = FetchType.*LAZY*,cascade = CascadeType.*DETACH*)  
@JoinColumn(name = "teacher\_id")  
private List<Course> courseList;

**Atentie@! “teacher\_id” e numele coloanei din tabelul course, nu din clasa Course care e Entity!**

* **Teacher nu are asa coloana teacher\_id, dar @JoinColumn poate fi folosit si pentru a crea legaturi bidirectionale, nu doar legatura cu foreign key. Desi teacher nu are asa coloana, Hibernate va intelege ca noi vrem sa cream o legatura bidirectionala, si cand va obtine cursurile din baza de date, va crea un obiect pentru fiecare curs din lista, bazandu-se pe entitatea Course, insa va folosi coloana “teacher\_id” din course table pentru a o compara cu id al Teacher, caci el stie ca foreign key din “teacher\_id” poate fi doar primary key din Teacher. Nu are importanta ca folosim mappedBy sau @JoinColumn.La mapped by scriem numele la obiectul care e foreign key din entity si hibernate se uita la numele coloanei lui, iar la @JoinColumn noi direct scriem numele. Hibernate daca vede ca asa coloana nu exista in Entity Teacher, va intelege ca vrem sa facem conexiunea bidirectionala.**
* E preferabil sa folosim totusi mappedBy

**NoForeignKey(nerecomandat)**

* Daca folosim JoinColumn in loc de mappedBy, e interesant ca e posibil sa nu facem legatura la foreign key. De ex, daca avem un tabel course si altul review, tabelul review va avea ca foreign key id al cursului. Dar, nu e neaparat necesar sa facem legatura cu foreign key.

@Entity  
@Table(name="review")  
public class Review {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name="id")  
 private int id;  
  
 @Column(name = "comment")  
 private String comment;



@Entity  
@Table(name="course")  
public class Course {  
 @Id  
 @GeneratedValue(strategy=GenerationType.*IDENTITY*)  
 @Column(name="id")  
 private int id;  
   
 @Column(name="title")  
 private String title;  
   
 @ManyToOne(cascade= {CascadeType.*PERSIST*, CascadeType.*MERGE*,  
 CascadeType.*DETACH*, CascadeType.*REFRESH*})  
 @JoinColumn(name="instructor\_id")  
 private Instructor instructor;  
  
 @OneToMany(fetch = FetchType.*LAZY*,cascade = CascadeType.*ALL*)  
 @JoinColumn(name="course\_id")  
 private List<Review> reviewList;

codul va merge:

// start a transaction  
 session.beginTransaction();  
   
 // get the instructor from db  
 Course tempCourse = new Course("Hibernate");  
  
 tempCourse.addReview(new Review("Great course .. loved it"));  
 tempCourse.addReview(new Review("The course was oke"));  
 tempCourse.addReview(new Review("Not bad at all"));  
  
session.save(tempCourse);  
  
  
 // commit transaction  
 session.getTransaction().commit();

Chiar daca review are un foreign key, noi nu l-am specificat in Entity, dar asa cum Cascade e all, fiecare review adaugat in lista de review list din course va fi salvat impreuna cu cursul in baza de date, si deoarece deja am specificat in @JoinColumn care coloana corespunde la foreign key pentru review, Hibernate va adauga automat id in foreign key.

**Atentie! Putem sa evitam setarea lui foreign key in entity doar daca campul cu foreign key accepta valori nule! Asta e din cauza ca Hibernate intai insereaza toate obiectele in baza de date, inclusiv review-urile, si abea apoi dupa ce le amplaseaza seteaza foreign key, dar initial ea va fi null**

**mappedBy vs JoinColumn**

Exista totusi si diferente intre mappedBy si JoinColumn:

1. JoinColumn e mai restrictiv. Deci, daca avem un teacher, si in el o List<> cu cursuri, daca salvam doar teacher, fara cursurile din lista, vom primi eroare. Deci, elementele din List<> trebuie mereu salvate, de nu primim eroare. Insa, daca vom folosi mappedBy, putem si sa nu salvam elementele din List<>, nu va fi nicio problema, el e mai putin restrictiv.
2. Presupunem ca avem cascade save.JoinColumn() se va asigura ca la salvarea lui Teacher, daca fiecare course din List<> de cursuri nu a adaugat o referinta la teacher object cu .setTeacher(), reiese ca foreign key va fi null, dar JoinColumn se va asigura sa adauge id a lui teacher ca foreign key pentru fiecare course, insa cu mappedBy, foreign key pentru fiecare course va ramane null.