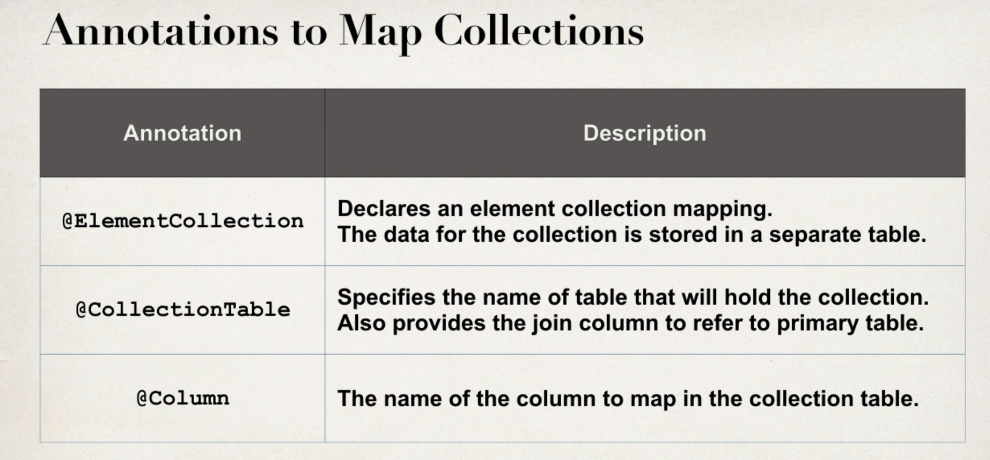
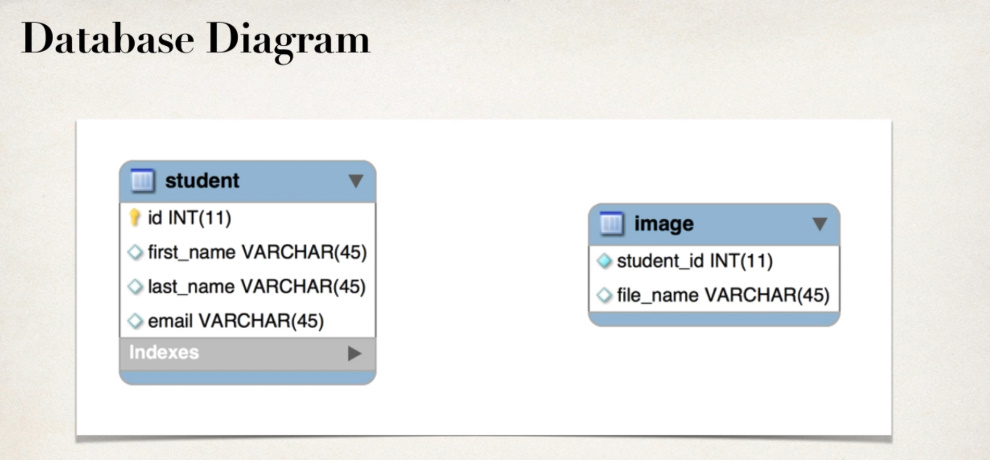
**Use case for Sets**

* Sets sunt bune cand nu ne intereseaza ordinea datelor si vrem ca ele sa fie unice
* Ele sunt bune si pentru a da raspunsuri Yes/No. De ex, este numele meu in lista de membri?

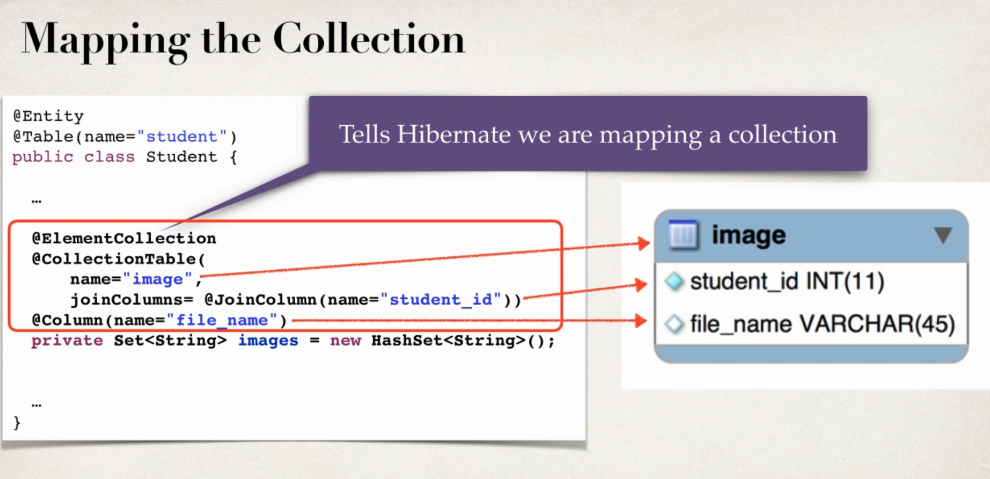
**Annotations to map Collections**



* Fie ca am un tabel Student si Image. Fiecare Image e pentru un student



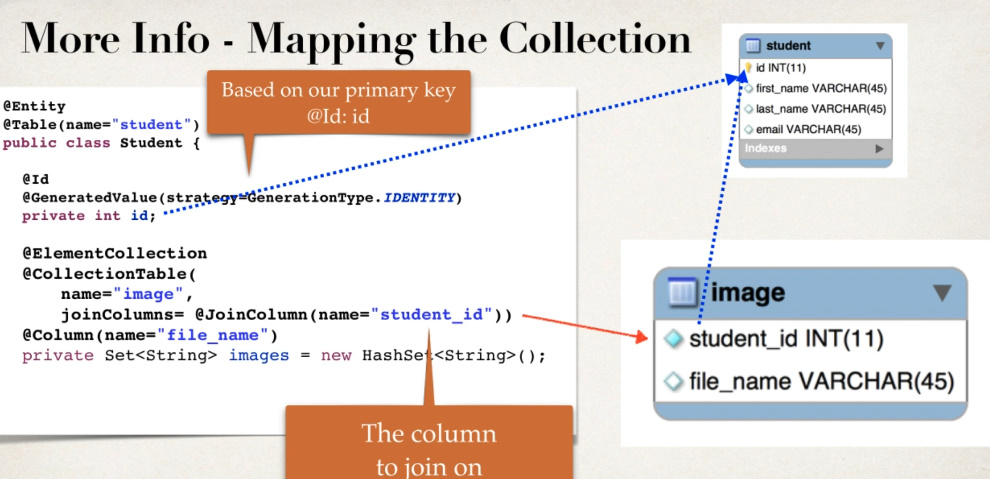
* Toate anotatiile de mai sus trebuie puse impreuna deasupra la un field



@ElementCollection – anume anotatia data i spune ca mapam o colectie si datele pentru aceasta colectie se afla in alt tabel, dar vrem sa folosim o Collection cu acele date in obiectul curent

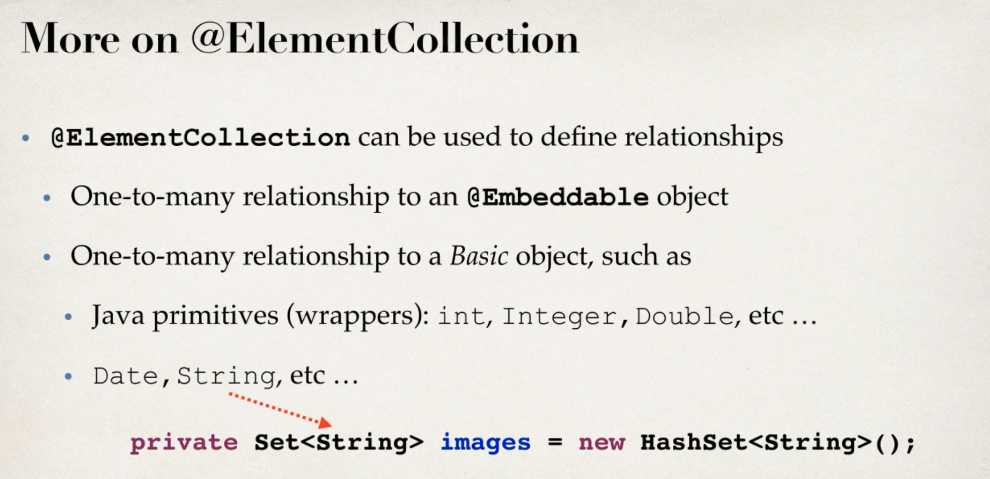
@CollectionTable(nam= “ “,joinColumns = @JoinColumn(name= “”)) – anume anotatia data specifica numele la tabel si coloana ce va fi luata ca datele ei sa se puna in Set<> pe baza la primary key a lui Student. @JoinColumn aici e pentru a face legatura bidirectionala intr-un fel, deci id la Student ca si cum e foreign key in Image. **Daca nu folosim joinColumns, automat se va lua numele la entity curent + \_id, gen student\_id, dar daca asa column nu exista, avem problema, de aceea mai bine scriem.**

* Putem face acum orice, sterge imagini, adauga, get etc.
* Atentie! In Set se vor pune anume datele coloanei “file\_name”, dar nu se vor pune toate. Nu asta e rolul lui @ElementCollection. El nu ia pur si simplu toate itemele si le pune in Set. Se vor lua anume acele iteme din tabelul image care au la coloana student\_id un id identic cu cel al primary key la Student
* Deci, daca Student creat are id 1, se vor lua toate denumirile imaginilor din tabelul image, din coloana “file\_name”, a caror student\_id value este 1, ca la student



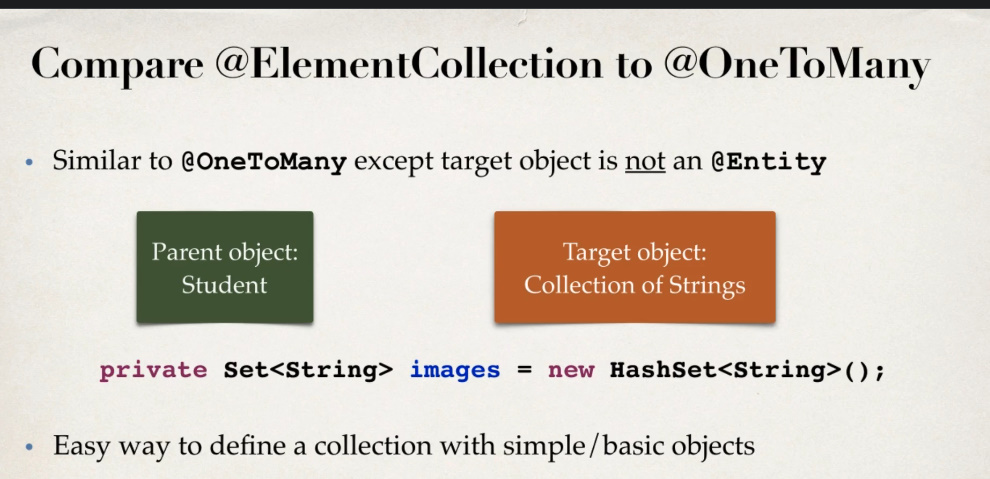
* Column la care se face @JoinColumn nu trebuie sa fi foreign key. student\_id din Image table nu e foreign key catre Student, si asta nu conteaza.

**More about @ElementCollection**

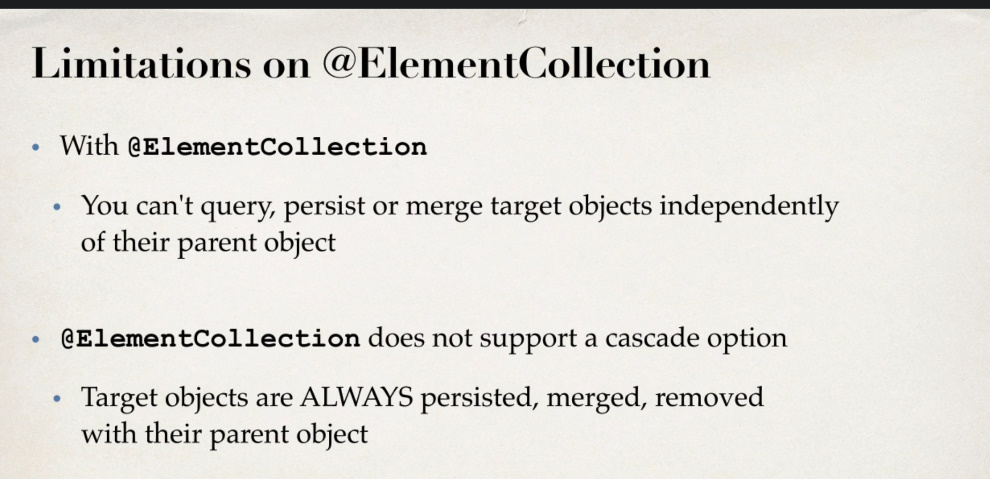


**@ElementCollection vs @OneToMany**

1. @OneToMany mereu face legatura cu un alt @Entity, dar @ElementCollection nu. El poate face legatura cu orice tabel, chiar daca el nu are o clasa @Entity creata



* @OneToMany suporta @Cascade, dar @ElementCollectio nu. Orice Collection anotata cu @ElementCollection are automat pus cascade all.
* Collection de la @ElementCollection nu pot fi deci manipulate independent de obiectul parinte. Adica, Set al nostru nu va putea fi salvat, sters sau updatat direct, ci doar va depinde mereu de actiunile luate asupra lui Student



Daca vom incerca sa facem ceva de genul:

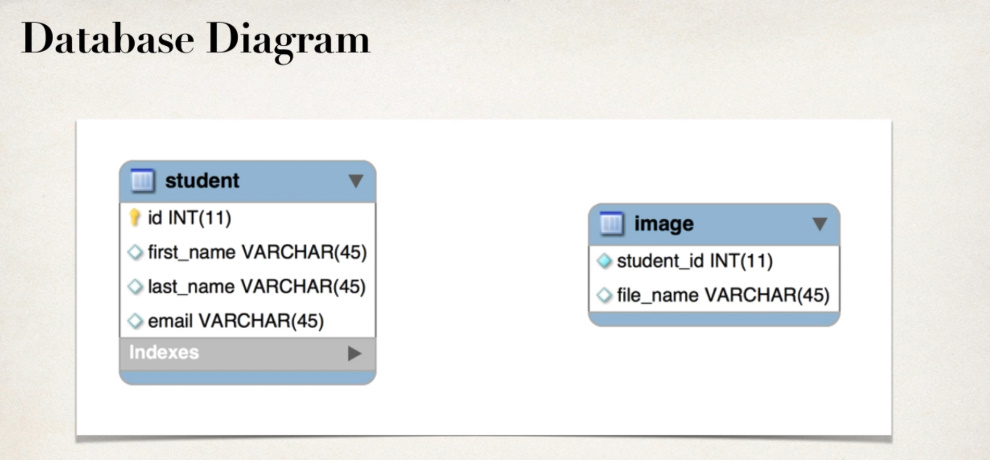
@PostMapping  
@Transactional  
public Student addStudent(@RequestBody Student student){  
 entityManager.persist(student.getImages());  
  
 return student;  
}

Vom primi eroare.



**Exemplu**

Fie baza de date



@Entity  
@Table(name = "student")  
@Getter  
@Setter  
@ToString  
@NoArgsConstructor  
public class Student {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 @Column(name = "id")  
 private Integer id;  
 @Column(name = "first\_name")  
 private String firstName;  
 @Column(name = "last\_name")  
 private String lastName;  
 @Column(name = "email")  
 private String email;  
  
 @ElementCollection  
 @CollectionTable(name = "image",  
 joinColumns = @JoinColumn(name = "student\_id"))  
 @Column(name = "file\_name")  
 private Set<String> images = new HashSet<>();  
  
 @Builder  
 public Student(String firstName, String lastName, String email) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.email = email;  
 }  
}

Si fie un asa RestController:

@RestController  
public class MyRestController {  
 @PersistenceContext  
 private EntityManager entityManager;  
  
 @GetMapping  
 @Transactional  
 public List<Student> getAll(){  
 TypedQuery<Student> typedQuery = entityManager.createQuery("FROM Student",Student.class);  
  
 return typedQuery.getResultList();  
 }  
 @PostMapping  
 @Transactional  
 public Student addStudent(@RequestBody Student student){  
 entityManager.persist(student);  
  
 return student;  
 }  
}

Acum, cand fom face Post cu un asa payload:

{

    "firstName" : "Mititiuc",

    "lastName" : "Eduard",

    "email" : "test@mail.ru",

    "images" : [

        "image1.png",

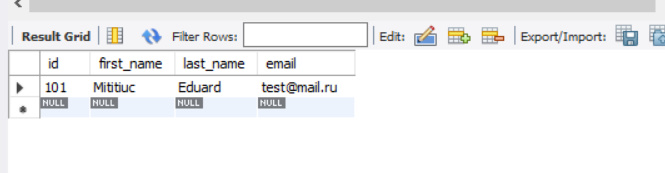
        "image2.png",

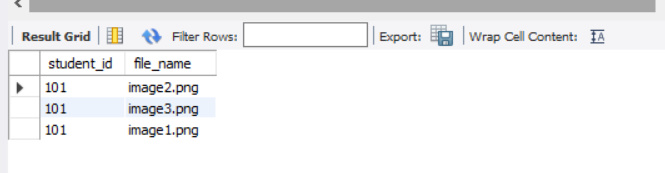
        "image3.png"

        ]

}

In baza de date primi:





Vedem ca Hibernate a avut grija sa insereseze la coloana de la @JoinColumn, pe baza la care se face conexiune bilaterala, id atribuit de baza de date la student. In plus, vedem cu Cascade si asa e automat

* Acum, pentru delete:
* @DeleteMapping("/{id}")  
  @Transactional  
  public String deleteStudent(@PathVariable int id){  
   entityManager.remove(id);  
    
   return "Student with id "+id+" was deleted.";  
  }

Se vor sterge si student si images ale lui din Image

Pentru a sterge o imagine a unui Student:

@DeleteMapping("/{id}")  
@Transactional  
public String deleteStudent(@PathVariable int id){  
 Student student = entityManager.find(Student.class,id);  
 student.getImages().remove("image1.png");  
  
 return "Image image1.png of with id "+id+" was deleted.";  
}