**Hibernate Advanced**

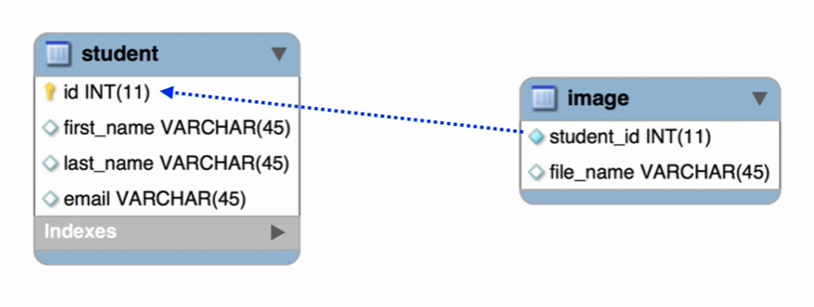
**Mapping Collections (Set and List)**

1. With **@ElementCollection**, We can map non-entities.

We can also use @OneToMany instead of @ElementCollection but then we need to create Separate Entities for that.

When we use @ElementCollection, JPA will create new Table in the database that will have a relationship with the parent table.

1. With @CollectionTable we can give the name to the non-entity collection and we can join it to the Parent Entity.

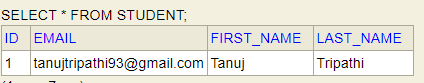


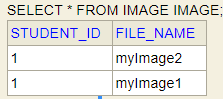
**Student.java**

@Data  
@Entity  
@Table(name = "student")  
public class Student {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
  
 @Column(name = "first\_name")  
 private String firstName;  
  
 @Column(name = "last\_name")  
 private String lastName;  
  
 @Column(name = "email")  
 private String email;  
  
 */\*\*  
 \* It will create the Table named image  
 \* `image` table will have the column named `student\_id`  
 \* `student\_id` will be joined to `id` of `student` table*

*\* Even if we don’t give joinColumns it will automatically use <parent-table>\_primaryColumnName  
 \*/* @ElementCollection // It is similar to @oneToMany,  
 @CollectionTable(name = "image", joinColumns = @JoinColumn(name = "student\_id"))  
 @Column(name = "file\_name")  
 private Set<String> images = new HashSet<>();  
  
 // Required constructors  
}

@RestController  
public class MyController {  
  
 @Autowired  
 private StudentRepository repository;  
  
 @GetMapping("/save")  
 public String saveInStudent() {  
 Student student = new Student("Tanuj", "Tripathi", "a@gmail.com");  
  
 Set<String> images = student.getImages();  
 images.add("myImage1");  
 images.add("myImage2");  
  
 repository.save(student);  
 return "Success";  
 }  
}





**List**

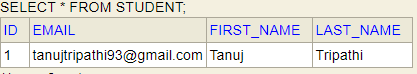
By using List we can maintain insertion order and can insert duplicate value

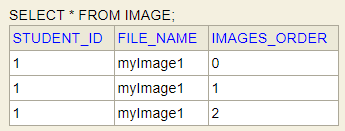
@OrderColumn: This annotation creates an extra column <entity-name>\_order and will give the order of values. The @**OrderColumn** annotation **can**'t be **used with Sets**, only with Lists. ... For **Sets**, the only option is to **use** the @**OrderBy** annotation using a field of the Entity for ordering, not the insertion order of the **Set**

@ElementCollection // It is similar to @oneToMany,  
@CollectionTable(name = "image")  
@OrderColumn  
@Column(name = "file\_name")  
private List<String> images = new ArrayList<>();

@GetMapping("/save")  
public String saveInStudent() {  
 Student student = new Student("Tanuj", "Tripathi", "tanujtripathi93@gmail.com");  
  
 List<String> images = student.getImages();  
 images.add("myImage1");  
 images.add("myImage1");  
 images.add("myImage1");  
  
 repository.save(student);  
 return "Success";  
}

Here we have not used Join Columns, It is to be noted that it will automatically join the column and give the default name <Parent-table-name>\_<primary-key-name>. Example our parent table name is student and primary key name is id, so the name of the joined column will be student\_id.





Maps