**Hibernate Advanced**

**Associations**

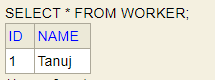
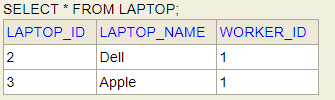
@Entity  
@Data  
public class Worker {  
  
 @Id  
 @GeneratedValue  
 private int id;  
 private String name;  
  
 @OneToMany(cascade = CascadeType.*PERSIST*)  
 private List<Laptop> laptops = new ArrayList<>();  
}

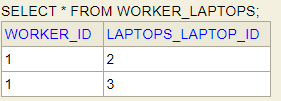
@Data  
@Entity  
public class Laptop {  
  
 @Id  
 @GeneratedValue  
 private int laptopId;  
 private String laptopName;  
  
 @ManyToOne  
 private Worker worker;  
  
}

@GetMapping("/asso")  
public void assosciation() {  
  
 Worker worker = new Worker();  
 worker.setName("Tanuj");  
  
 Laptop laptop1 = new Laptop();  
 laptop1.setLaptopName("Dell");  
 laptop1.setWorker(worker);  
  
 Laptop laptop2 = new Laptop();  
 laptop2.setLaptopName("Apple");  
 laptop2.setWorker(worker);  
  
 List<Laptop> laptops = new ArrayList<Laptop>() {{  
 add(laptop1);  
 add(laptop2);  
 }};  
  
 worker.setLaptops(laptops);  
  
 workerRepository.save(worker);  
}

Three tables will be formed

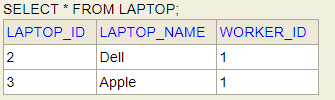
1. worker
2. laptop
3. worker\_laptop

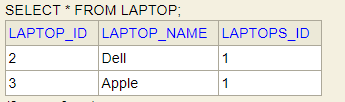


Now if we do not want three tables to be created, we can use mappedBy. The third table above won’t be created if we use mappedby. Similary, @JoinColumn we can use instead of mappedBy for Unidirectional Flow.

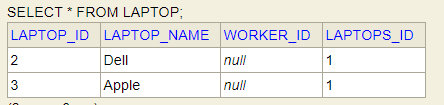
In case of mappedby (both @OneToMany and @ManyToOne is used i.e, bidirectional)



In case of JoinColumn : If @ManyToOne is not used i.e UNIDIRECTIONAL. In case if @JoinColumn is used with Bidirectional, one extra column will also come in the below table



In case join column is used with Bidirectional flow



LAPTOPS\_ID is created from the <field above which join column is used> \_<entity Id>

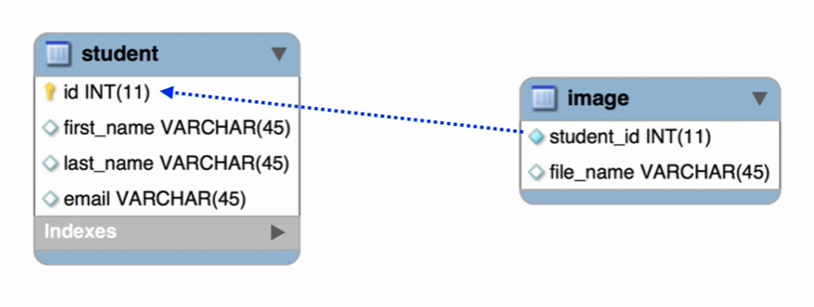
**Mapping Collections (Set and List)**

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.

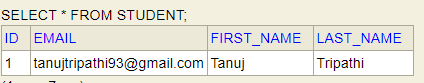


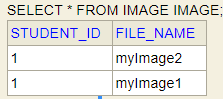
**Student.java**

@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**

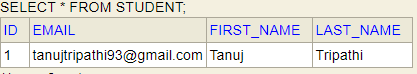
By using List we can maintain insertion order and can insert duplicate value

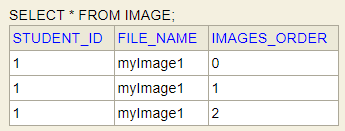
@OrderColumn: This annotation creates an extra column <entity-name>\_order and will give the order of values. The @**OrderColumn** annotation **can**'t be **used with Sets**, only with Lists. ... For **Sets**, the only option is to **use** the @**OrderBy** annotation using a field of the Entity for ordering,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

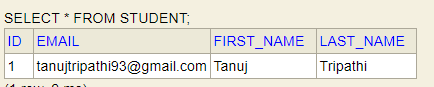


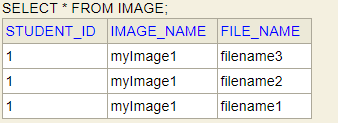


**Map**

@ElementCollection // It is similar to @oneToMany,  
@CollectionTable(name = "image")  
**@MapKeyColumn**(name = "file\_name")  
@Column(name = "image\_name")  
private Map<String, String> images = new HashMap();

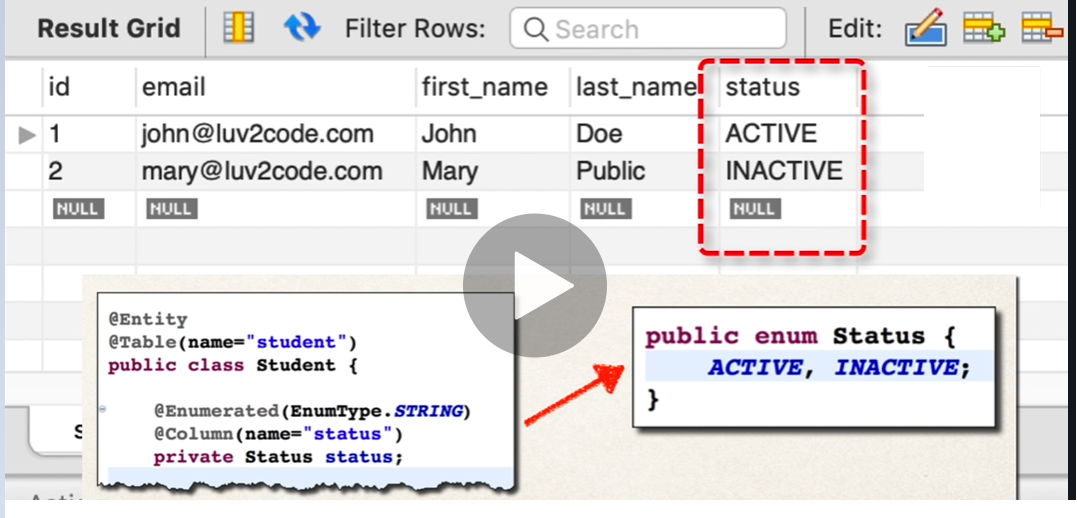
Here, There will be two columns , 1 for map key and another for map value





Enums

We use **@Enumerated** for Enums.



**Inheritance**

**Strategies:**

1. Single table
2. Table per class
3. JOINED
4. Mapped Superclass

Single Table Strategy

Annotations used in Inheritance

1. @Inheritance : SINGLE\_TABLE, TABLE\_PER\_CLASS, JOINED
2. @DiscriminatorColumn: Name of the column that holds the discriminator values
3. @DiscriminatorValue: A unique value that describes the given subclass.

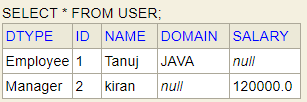
@Data  
@Entity  
@Table(name = "user")  
@Inheritance(strategy = InheritanceType.*SINGLE\_TABLE*)  
public abstract class User {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private int id;  
  
 @Column(name = "name")  
 private String name;  
  
 public User(String name) {  
 this.name = name;  
 }  
}

@Data  
@Entity  
@Table(name = "employee")  
public class Employee extends User {  
  
 @Column(name = "domain")  
 private String domain;  
  
 public Employee(String name, String domain) {  
 super(name);  
 this.domain = domain;  
 }  
}

@Data  
@Entity  
@Table(name = "manager")  
public class Manager extends User {  
  
 @Column(name = "salary")  
 private Double salary;  
  
 public Manager(String name, Double salary) {  
 super(name);  
 this.salary = salary;  
 }  
}

public interface UserRepository extends CrudRepository<User, Integer> {  
}

@GetMapping("/saveInheritance")  
public void saveInheritnce() {  
 Employee employee = new Employee("Tanuj", "JAVA");  
 Manager manager = new Manager("kiran", 120000.0);  
 userRepository.save(employee);  
 userRepository.save(manager);  
}



By default the discriminator value is **DTYPE** and **Discriminator columns are the name of child classes**. We can override the Discriminator value and Column be used @DiscriminatorValue and @DiscriminatorColumn annotations

@Data  
@Entity  
@Table(name = "user")  
@Inheritance(strategy = InheritanceType.*SINGLE\_TABLE*)  
**@DiscriminatorColumn**(name = "USER\_TYPE")  
public abstract class User {)

And in child tables we can use **@DiscriminatorValue**.

@Data  
@Entity  
@Table(name = "employee")  
**@DiscriminatorValue**(value = "EMPLOYEE")  
public class Employee extends User {}

@Data  
@Entity  
@Table(name = "employee")  
**@DiscriminatorValue**(value = "MANAGER")  
public class Manger extends User {}

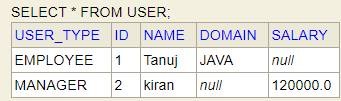


Table per class Strategy

It will generate separate table for each child table. It is to be noted that the GenerationType should be TABLE for this strategy. We can remove @DiscriminatorColumn and @DiscriminatorValue since we don’t need as all the table will be separate.

Tables that will be created

1. HIBERNATE\_SEQUENCES : it will derive primary key from this sequence.
2. STUDENT
3. MANAGER

@Data  
@Entity  
@Table(name = "user")  
@Inheritance(strategy = InheritanceType.*TABLE\_PER\_CLASS*)  
public abstract class User {  
  
 @Id  
 @GeneratedValue(**strategy = GenerationType.*TABLE***)  
 private int id;

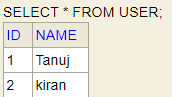
**JOINED TABLE STRATEGY**

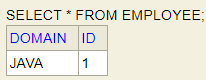
It will create primary key for USER table and will join child tables to it with the help of foreign key.

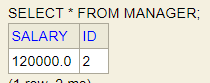
@Data  
@Entity  
@Table(name = "user")  
@Inheritance(strategy = InheritanceType.*JOINED*)  
public abstract class User {  
  
 @Id  
 @GeneratedValue(**strategy = GenerationType.*IDENTITY***)  
 private int id;

It will create 3 tables.

1. USER
2. EMPLOYEE
3. MANAGER







**Mapped Super class**

In this we use annotation @MappedSuperClass. In this strategy only the child classes are created and they have all the fields of parent class. Hence, we do not need any annotations in Parent class such as @Entity @Table etc since this class won’t be created.

@Data  
**@MappedSuperclass**  
public abstract class User {}

