



## Compte Rendu JEE :

### Spring Data JPA

- **Réalisé par : BENNOUNA LORIDY MOHAMED**
- **GROUPE : 4IIR G2/ EMSI CENTRE**

Année universitaire 2023-2024

```
Etudiant.java x Tp1Application.java x pom.xml (TP1) x EtudiantRepository.java x application.properties x
25 <dependencies>
26 <dependency>
27 <groupId>org.springframework.boot</groupId>
28 <artifactId>spring-boot-starter-data-jpa</artifactId>
29 </dependency>
30 <dependency>
31 <groupId>org.springframework.boot</groupId>
32 <artifactId>spring-boot-starter-web</artifactId>
33 </dependency>
34 <dependency>
35 <groupId>org.springframework.boot</groupId>
36 <artifactId>spring-boot-devtools</artifactId>
37 <scope>runtime</scope>
38 <optional>true</optional>
39 </dependency>
40 <dependency>
41 <groupId>com.h2database</groupId>
42 <artifactId>h2</artifactId>
43 <scope>runtime</scope>
44 </dependency>
45 <dependency>
46 <groupId>org.projectlombok</groupId>
47 <artifactId>lombok</artifactId>
48 <optional>true</optional>
49 </dependency>
50 <dependency>
51 <groupId>org.springframework.boot</groupId>
52 <artifactId>spring-boot-starter-test</artifactId>
53 <scope>test</scope>
```

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.29</version>
</dependency>
```

```
application.properties x
1 #spring.datasource.url=jdbc:h2:mem:Etudiant-db
2 #spring.h2.console.enabled=true
3 spring.datasource.url=jdbc:mysql://localhost:3308/Cours-db?createDatabaseIfNotExist=true
4 spring.datasource.username=root
5 spring.datasource.password=Tasnim2009
6 server.port=8083
7 spring.jpa.show-sql=true
8 spring.jpa.hibernate.ddl-auto=create
9 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDBDialect
10
```

### 1) Creation de l'entité Jpa Cours :

Cette classe représente une entité JPA appelée Cours, avec les attributs suivants :

id : Un champ entier qui sert de clé primaire pour l'entité. Il est annoté avec @Id et @GeneratedValue pour indiquer qu'il s'agit d'une valeur générée automatiquement en utilisant une stratégie d'identité.

title : Un champ de chaîne de caractères qui représente le titre du cours. Il est annoté avec @Column pour spécifier une longueur de 25 caractères, et qu'il ne peut pas être nul et n'a pas besoin d'être unique.

description : Un champ de chaîne de caractères qui représente la description du cours.

timing : Un champ entier.

professeur : Un champ de relation OneToOne avec une autre entité Professeur, indiquant qu'un cours est associé à exactement un professeur. Il est annoté avec @OneToOne pour spécifier la relation.

seance : Un champ de relation OneToMany avec une autre entité Seance, indiquant qu'un cours peut avoir plusieurs séances. Il est annoté avec @OneToMany avec l'attribut mappedBy défini sur "cours" pour spécifier le côté inverse de la relation, et l'attribut fetch défini sur FetchType.EAGER pour indiquer que la collection doit être chargée de manière précoce.

La classe a également les annotations @Data, @AllArgsConstructor et @NoArgsConstructor, qui sont des annotations de Lombok qui génèrent des méthodes courantes telles que les getters, setters, constructeurs et toString() pour la classe.

```
Cours.java x
1 package ma.emsi.tpspringdata.entities;
2
3 import jakarta.persistence.*;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 import java.util.Collection;
9
10 10 usages
11 @Data @AllArgsConstructor @NoArgsConstructor
12 @Entity
13 public class Cours {
14     no usages
15     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private int id;
17     no usages
18     @Column(length = 25, nullable = false, unique = false)
19     private String title;
20     no usages
21     private String description;
22     no usages
23     private int timing;
24     no usages
25     @OneToOne
26     private Professeur professeur;
27     no usages
28     @OneToMany(mappedBy = "cours", fetch = FetchType.EAGER)
29     private Collection<Seance> seance;
30
31 }
```

## 2) Créer l'entité Jpa Etudiant:

Cette classe représente une entité JPA appelée Etudiant avec les annotations suivantes :

@Entity: Indique que cette classe est une entité persistante qui sera mappée à une table dans une base de données.

`@Table(name = "EMSI_STUDENTS")`: Spécifie le nom de la table dans la base de données où les objets de cette classe seront stockés.

`@Data`, `@NoArgsConstructor`, `@AllArgsConstructor`, `@ToString`: Ce sont des annotations de Lombok qui génèrent des méthodes courantes telles que les getters, setters, constructeurs sans argument, constructeurs avec tous les arguments, et la méthode `toString()` pour la classe.

`@Id`, `@GeneratedValue(strategy= GenerationType.IDENTITY)`: Indiquent que le champ `id` est la clé primaire de l'entité, et que sa valeur sera générée automatiquement à l'aide d'une stratégie d'identité.

`@Column(name = "REGISTRATION_N", unique = true)`: Spécifie que le champ `registrationNumber` sera mappé à une colonne dans la table avec le nom "REGISTRATION\_N", et qu'il doit être unique.

`@Column(name = "Name", length = 30, nullable = false)`: Spécifie que le champ `fullName` sera mappé à une colonne dans la table avec le nom "Name", avec une longueur maximale de 30 caractères, et qu'il ne peut pas être nul.

`@Temporal(TemporalType.DATE)`: Indique que le champ `birthay` sera mappé à une colonne de type DATE dans la base de données.

`private Boolean stillActive`: Un champ de type booléen.

`@Temporal(TemporalType.TIMESTAMP) @CreationTimestamp`: Indique que le champ `lastConnection` sera mappé à une colonne de type TIMESTAMP dans la base de données, et qu'il sera automatiquement mis à jour avec la date et l'heure actuelles lorsqu'un nouvel enregistrement d'étudiant est créé.

`@ManyToMany(mappedBy = "etudiants", fetch = FetchType.EAGER):`

Indique une relation `ManyToMany` avec l'entité `Seance`, où un étudiant peut être associé à plusieurs séances. L'attribut `mappedBy` est défini sur `"etudiants"` pour indiquer le côté inverse de la relation, et l'attribut `fetch` est défini sur `FetchType.EAGER` pour indiquer que la collection de séances associées doit être chargée de manière précoce.

```
Etudiant.java x
1 package ma.emsi.tpspringdata.entities;
2
3 import ...
  7 usages
13 @Entity @Table(name = "EMSI_STUDENTS")
14 @Data @NoArgsConstructor @AllArgsConstructor @ToString
15 public class Etudiant {
  no usages
16     @Id @GeneratedValue(strategy= GenerationType.IDENTITY)
17     private Integer id;
  no usages
18     @Column(name = "REGISTRATION_N", unique = true)
19     private String registrationNumber;
  no usages
20     @Column(name = "Name", length = 30, nullable = false)
21     private String fullName;
  no usages
22     @Temporal(TemporalType.DATE)
23     private Date birthay;
  no usages
24     private Boolean stillActive;
  no usages
25     @Temporal(TemporalType.TIMESTAMP) @CreationTimestamp
26     private Date lastConnection;
  no usages
27     @ManyToMany(mappedBy = "etudiants", fetch = FetchType.EAGER)
28     private Collection<Seance> seances=new ArrayList<>();
29 }
```

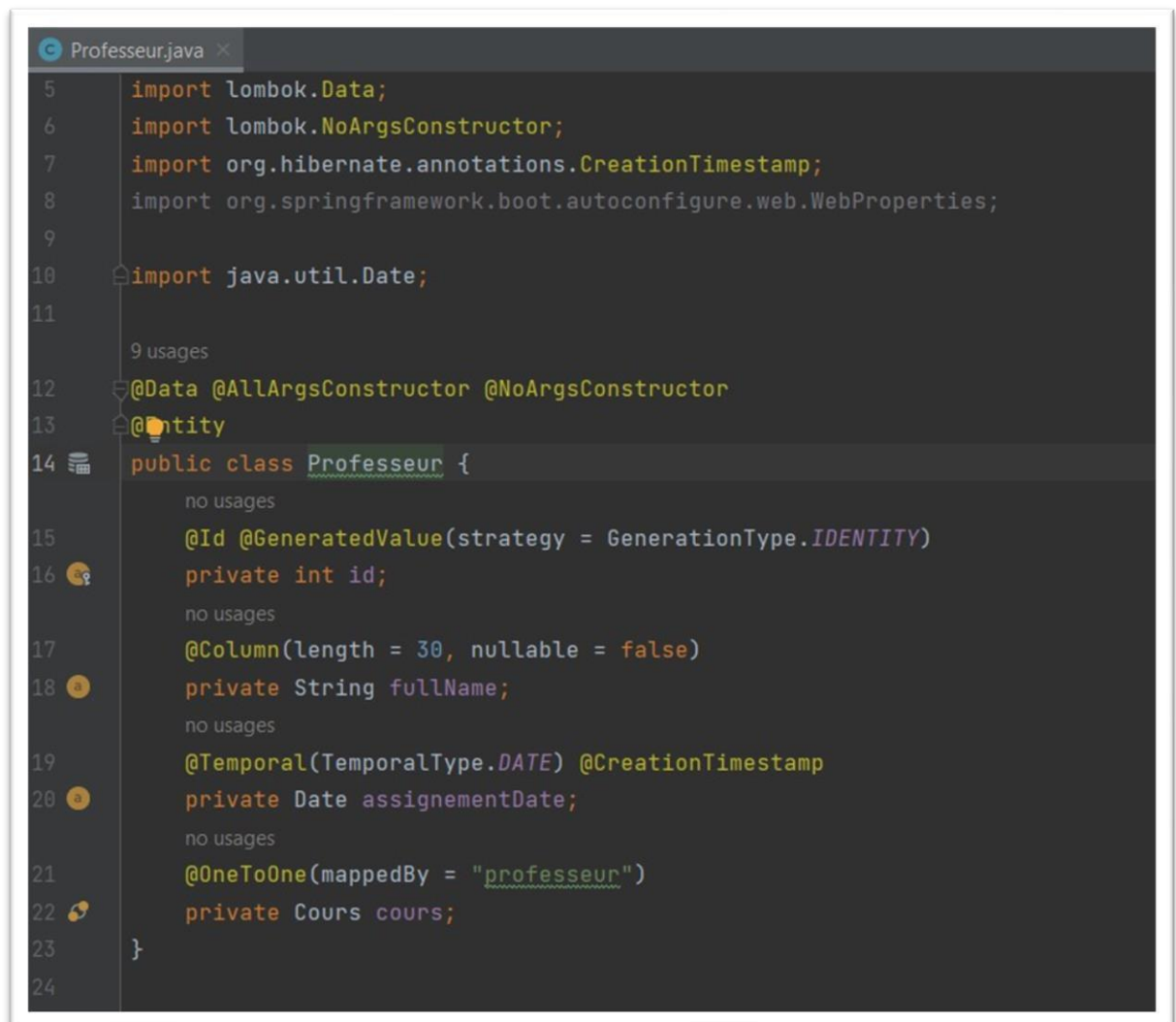
### 3) Créer l'entité Jpa Professeur:

Cette classe représente une entité JPA appelée Professeur avec les annotations suivantes :

@Column(length = 30, nullable = false): Spécifie que le champ fullName sera mappé à une colonne dans la table avec une longueur maximale de 30 caractères, et qu'il ne peut pas être nul.

@Temporal(TemporalType.DATE) @CreationTimestamp: Indique que le champ assignementDate sera mappé à une colonne de type DATE dans la base de données, et qu'il sera automatiquement mis à jour avec la date actuelle lorsqu'un nouvel enregistrement de professeur est créé.

@OneToOne(mappedBy = "professeur"): Indique une relation OneToOne avec l'entité Cours, où un professeur est associé à un seul cours. L'attribut mappedBy est défini sur "professeur" pour indiquer le côté inverse de la relation, où la classe Cours a une référence vers un objet Professeur.



```
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7 import org.hibernate.annotations.CreationTimestamp;
8 import org.springframework.boot.autoconfigure.web.WebProperties;
9
10 import java.util.Date;
11
12 @Data @AllArgsConstructor @NoArgsConstructor
13 @Entity
14 public class Professeur {
15     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private int id;
17     @Column(length = 30, nullable = false)
18     private String fullName;
19     @Temporal(TemporalType.DATE) @CreationTimestamp
20     private Date assignementDate;
21     @OneToOne(mappedBy = "professeur")
22     private Cours cours;
23 }
24
```

#### 4) Création de l'entité Jpa seance:

Cette classe représente une entité appelée **Seance** avec les annotations suivantes :



`@Temporal(TemporalType.DATE)`, `@Temporal(TemporalType.TIME)`:  
Indiquent que les champs `date`, `start_time` et `end_time` seront mappés à des colonnes de type `DATE` et `TIME` respectivement dans la base de données.

`@ManyToOne`: Indique une relation `ManyToOne` avec l'entité `Cours`, où plusieurs séances sont associées à un seul cours. Cela signifie qu'une séance a une référence vers un objet `Cours`.

`@ManyToMany(fetch = FetchType.EAGER)`: Indique une relation `ManyToMany` avec l'entité `Etudiant`, où plusieurs étudiants peuvent être associés à plusieurs séances. Cela signifie qu'une séance peut avoir plusieurs étudiants et qu'un étudiant peut participer à plusieurs séances. L'attribut `fetch` est défini sur `FetchType.EAGER` pour charger les étudiants en même temps que la séance pour éviter les problèmes de chargement paresseux.

`@ToString.Exclude`: Exclut la collection d'étudiants de la méthode `toString()` générée par Lombok pour éviter les boucles de références circulaires.

`@JsonProperty(access = JsonProperty.Access.WRITE_ONLY)`: Indique que la collection d'étudiants ne doit pas être sérialisée lors de la conversion de l'objet en JSON, afin d'éviter la récursivité infinie lors de la sérialisation.

```

Seance.java x
1  package ma.emsi.tpspringdata.entities;
2
3  import ...
13
    8 usages
14  @Entity
15  @Data @AllArgsConstructor @NoArgsConstructor
16  public class Seance {
17      no usages
18      @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
19      private int id;
20      no usages
21      @Temporal(TemporalType.DATE)
22      private Date date;
23      no usages
24      @Temporal(TemporalType.TIME)
25      private Date start_time;
26      no usages
27      @Temporal(TemporalType.TIME)
28      private Date end_time;
29      no usages
30      @ManyToOne
31      private Cours cours;
32      no usages
33      @ManyToMany(fetch = FetchType.EAGER)
34      @ToString.Exclude
35      @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
36      private Collection<Etudiant> etudiants = new ArrayList<>();
37  }

```

```

1  package ma.emsi.tpspringdata.repositories;
2
3  import ma.emsi.tpspringdata.entities.Cours;
4  import org.springframework.data.jpa.repository.JpaRepository;
5
6  2 usages
7  public interface CoursRepository extends JpaRepository<Cours,Integer> {
8  }

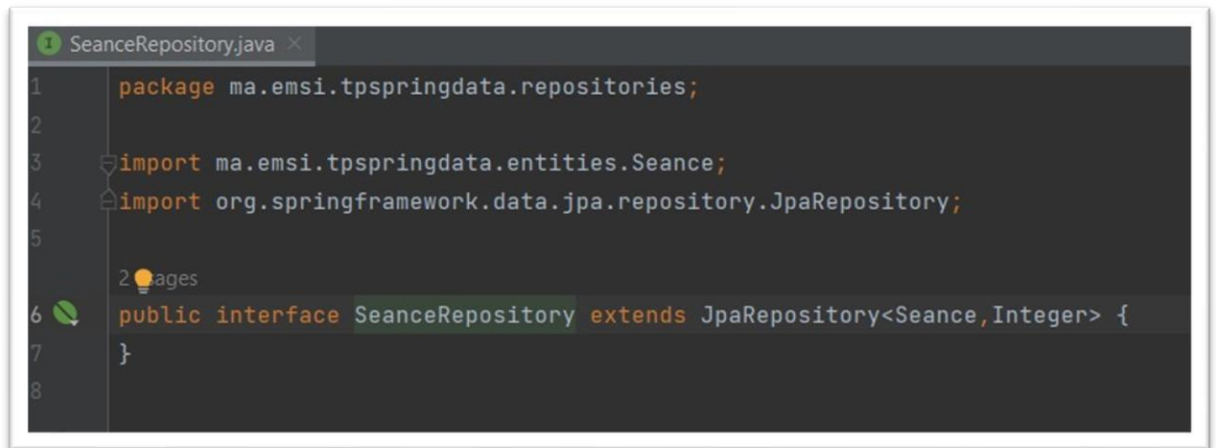
```

## 5) Création l'interface Etudiant Repository basé sur spring data.

EtudiantRepository est une interface qui hérite JpaRepository du framework Spring Data JPA. Elle est utilisée pour définir les opérations de persistance (CRUD) (Create, Read, Update, Delete) spécifiques à l'entité Etudiant dans la base de données. Dans le cas de EtudiantRepository, en héritant JpaRepository, On spécifie deux champs : Etudiant c'est le nom de la classe et Integer c'est le type de la clé primaire, les méthodes prédéfinies pour la persistance des données sont automatiquement générées pour l'entité Etudiant, avec les opérations courantes telles que l'ajout, la recherche, la mise à jour et la suppression d'étudiants dans la base de données

```
EtudiantRepository.java x
1 package ma.emsi.tpspringdata.repositories;
2
3 import ma.emsi.tpspringdata.entities.Etudiant;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 2 pages
7 public interface EtudiantRepository extends JpaRepository<Etudiant,Integer> {
8 }
9
```

```
ProfesseurRepository.java x
1 package ma.emsi.tpspringdata.repositories;
2
3
4 import ma.emsi.tpspringdata.entities.Professeur;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 2 pages
8 public interface ProfesseurRepository extends JpaRepository<Professeur,Integer> {
9 }
10
```



```
1 package ma.emsi.tpspringdata.repositories;
2
3 import ma.emsi.tpspringdata.entities.Seance;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface SeanceRepository extends JpaRepository<Seance, Integer> {
7 }
8
```

#### 4) Teste de l'application avec des opération d'ajout de consultation et de modification .

Dans la méthode main, on a utilisé `SpringApplication.run()` pour exécuter votre application Spring Data.

Ensuite, on a créé une méthode `start()` annotée avec `@Bean` qui implémente l'interface `CommandLineRunner`. Cela signifie que cette méthode sera exécutée au démarrage de l'application .

Dans cette méthode `start()`, on a utilisé un `Stream.of()` pour itérer sur une liste de noms complets d'étudiants ("Akram Lkihal", "Ahmed Zidany", "Yasser Hmimou") et pour chaque nom, on a créé un objet `Etudiant`, défini ses propriétés et on a enregistré dans `etudiantRepository` en utilisant la méthode `save()`.

`etudiant.setSeances(null)`, ce qui affecte null à la collection de séances de l'étudiant.



```
1 package ma.emsi.tpspringdata;
2
3 import ...
4
18
19 @SpringBootApplication
20 public class TpSpringDataApplication {
21
22     public static void main(String[] args) {
23
24         SpringApplication.run(TpSpringDataApplication.class, args);
25     }
26
27     @Bean
28     CommandLineRunner start(CoursRepository coursRepository, ProfesseurRepository professeurRepository, SeanceRepository seanceRep
29         return args -> {
30             Stream.of( ...values: "Akram Lkhal", "Ahmed Zidany", "Yasser Hmimou")
31                 .forEach(fullName->{
32                     Etudiant etudiant= new Etudiant();
33                     etudiant.setRegistrationNumber(fullName );
34                     etudiant.setFullName(fullName);
35                     etudiant.setBirthay(new Date());
36                     etudiant.setLastConnection(new Date());
37                     etudiant.setStillActive(Math.random()>0.5?true:false);
38                     etudiant.setSeances(null);
39                     etudiantRepository.save(etudiant);
40
41         }
42     }
```

Dans le premier `Stream.of()`, on a itéré sur une liste de noms complets de professeurs ("Yassine Barami", "Ilyass Mouatassim", "Abdelmoughit Aitchihab") et pour chaque nom, on a créé un objet `Professeur`, défini ses propriétés et on l'a enregistré dans `professeurRepository` en utilisant la méthode `save()`.

`professeur.setCours(null)`, ce qui affecte `null` à la propriété `cours` du professeur.

Dans le deuxième `Stream.of()`, on a itéré sur une liste de chaînes de caractères ("1", "2", "3") qui semblent représenter des identifiants de séances. Pour chaque identifiant, on a créé un objet `Seance`, défini ses propriétés et on l'a enregistré dans `seanceRepository` en utilisant la méthode `save()`.

`seance.setCours(null)` et `seance.setEtudiants(null)`, ce qui affecte `null` aux propriétés `cours` et `etudiants` de la séance.

```

Stream.of("Yassine Barami", "Ilyass Mouatassim", "Abdelmoughit Aitchihab")
    .forEach(fullName->{
        Professeur professeur= new Professeur();
        professeur.setFullName(fullName);
        professeur.setAssignementDate(new Date());
        professeur.setCours(null);
        professeurRepository.save(professeur);

    });

Stream.of("1", "2", "3")
    .forEach(Seance->{
        Seance seance= new Seance();
        seance.setDate(new Date());
        seance.setStart_time(new Date());
        seance.setEnd_time(new Date());
        seance.setCours(null);
        seance.setEtudiants(null);
        seanceRepository.save(seance);
    });

```

Dans `Stream.of()`, on a itéré sur une liste de noms de cours ("math", "physique", "informatique") et pour chaque nom, on a créé un objet `Cours`, défini ses propriétés et on l'a enregistré dans `coursRepository` en utilisant la méthode `save()`.

`cours.setProfesseur(null)` et `cours.setSeance(null)`, ce qui affecte `null` aux propriétés `professeur` et `seances` du `cours`.

On a récupéré des objets `Cours` et `Professeur` des interfaces en utilisant leurs identifiants respectifs à l'aide de la méthode `findById()` et on a associé les objets `Cours` avec les objets `Professeur` en utilisant la méthode `setProfesseur()` sur les objets `Cours`. Enfin, on a enregistré les objets `Cours` mis à jour dans le `coursRepository` en utilisant à nouveau la méthode `save()`.

```

Stream.of( ...values: "math", "physique", "informatique")
    .forEach(cour->{
        Cours cours = new Cours();
        cours.setTitle(cour);
        cours.setDescription(Math.random()>0.5?"intéressant":"long");
        cours.setTiming(5);
        cours.setProfesseur(null);
        cours.setSeance(null);
        coursRepository.save(cours);
    });

Cours cours1= coursRepository.findById(1).orElse( other: null);
Professeur professeur1= professeurRepository.findById(1).orElse( other: null);
cours1.setProfesseur(professeur1);
coursRepository.save(cours1);

Cours cours2= coursRepository.findById(2).orElse( other: null);
Professeur professeur3= professeurRepository.findById(3).orElse( other: null);
cours2.setProfesseur(professeur3);
coursRepository.save(cours2);

Cours cours3= coursRepository.findById(3).orElse( other: null);
Professeur professeur2= professeurRepository.findById(2).orElse( other: null);
cours3.setProfesseur(professeur2);
coursRepository.save(cours3);

```

On a continué à associer les objets Seance et Etudiant en utilisant les identifiants et les références récupérés.



Ecole Marocaine  
des Sciences de l'Ingénieur

On a récupéré un objet Seance en utilisant son identifiant avec la méthode findById() et avez associé cet objet Seance avec un objet Cours en utilisant la méthode setCours() sur l'objet Seance. Ensuite, on a enregistré l'objet Seance mis à jour dans seanceRepository en utilisant la méthode save().



Ensuite, on a récupéré un objet Etudiant en utilisant son identifiant avec la méthode findById() et on a vérifié si la liste seances de l'objet Etudiant n'était pas nulle. Si la liste seances n'était pas nulle, on a ajouté l'objet Seance associé précédemment à la liste seances de l'objet Etudiant en utilisant la méthode add() sur la liste. On a ajouté l'objet Etudiant à la liste etudiants de l'objet Seance en utilisant la méthode add() sur la liste. Enfin, on a enregistré les objets Etudiant et Seance mis à jour dans etudiantRepository et seanceRepository respectivement en utilisant à nouveau la méthode save().

Enfin, on a affiché les séances des étudiants.

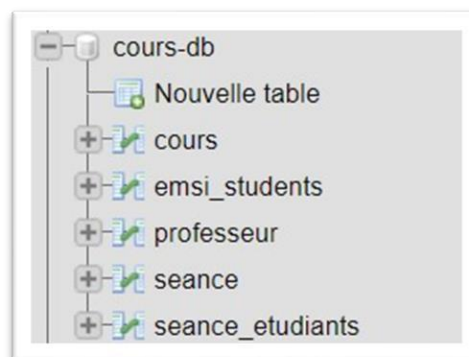
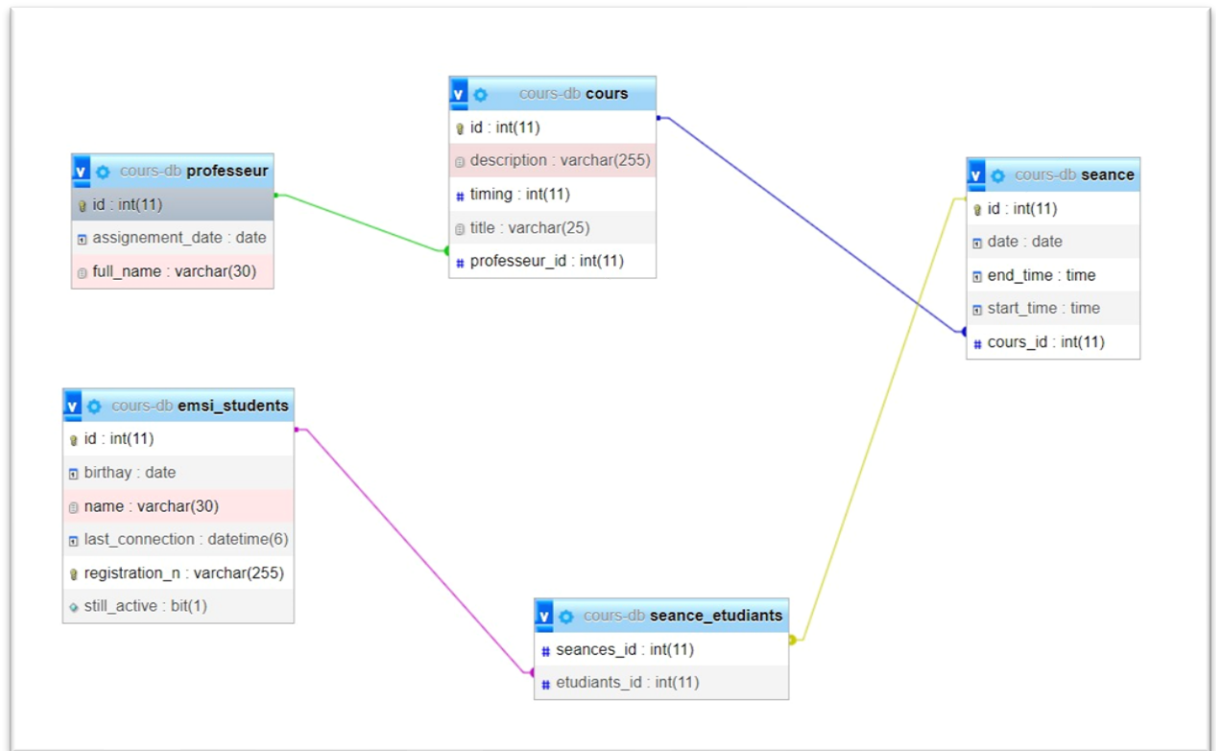
```
Seance seance1= seanceRepository.findById(1).orElse( other: null);
seance1.setCours(cours1);
seanceRepository.save(seance1);

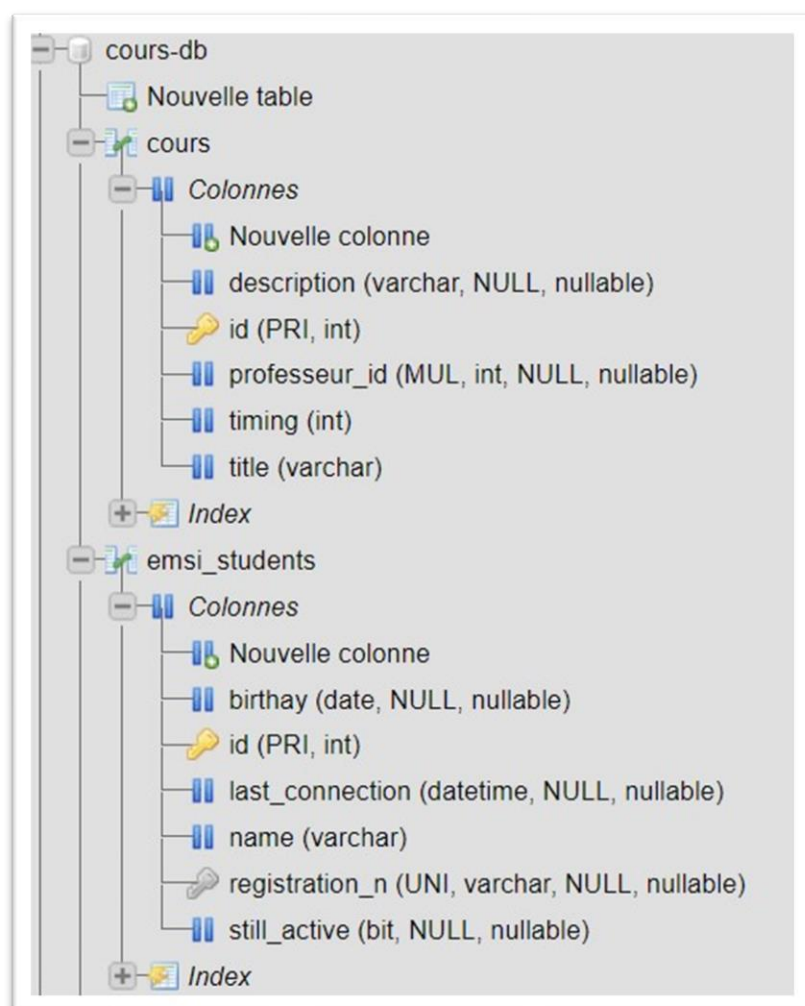
Etudiant etudiant1= etudiantRepository.findById(1).orElse( other: null);
if(etudiant1.getSeances()!=null) {
    etudiant1.getSeances().add(seance1);
    seance1.getEtudiants().add(etudiant1);
    etudiantRepository.save(etudiant1);
    seanceRepository.save(seance1);
}

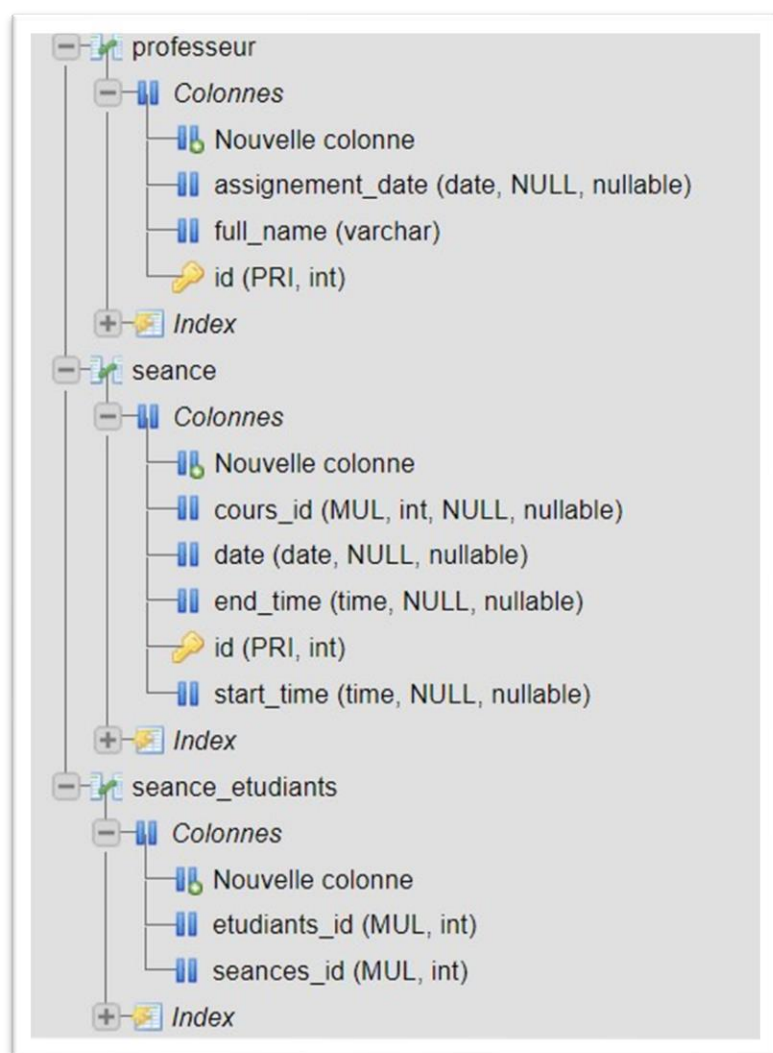
etudiant1.getSeances().forEach(s->{
    System.out.println("SEANCE=>" + s.toString());
});
```









```
hibernate: alter table cours drop foreign key FK3vnmktsmivc7wzqjso0u0v05u
Hibernate: alter table seance drop foreign key FKrrc1k4hpxsmh2hav4gl5wjgod
Hibernate: alter table seance_etudiants drop foreign key FKl9vLugsleaoui4y889lkdc3t
Hibernate: alter table seance_etudiants drop foreign key FKgs3nftvelfvhhx4grl040bbi
Hibernate: drop table if exists cours
Hibernate: drop table if exists emsi_students
Hibernate: drop table if exists professeur
Hibernate: drop table if exists seance
Hibernate: drop table if exists seance_etudiants
Hibernate: create table cours (id integer not null auto_increment, description varchar(255), timing integer not null, title varchar(25) not null, professeur_id integer, primary
Hibernate: create table emsi_students (id integer not null auto_increment, birthay date, name varchar(30) not null, last_connection datetime(6), registration_n varchar(255), sti
Hibernate: create table professeur (id integer not null auto_increment, assignement_date date, full_name varchar(30) not null, primary key (id)) engine=InnoDB
Hibernate: create table seance (id integer not null auto_increment, date date, end_time time, start_time time, cours_id integer, primary key (id)) engine=InnoDB
Hibernate: create table seance_etudiants (seances_id integer not null, etudiants_id integer not null) engine=InnoDB
Hibernate: alter table emsi_students add constraint UK_a3vjtvlbahy45i3mbr04y7hsu unique (registration_n)
```









| <div>←T→</div>           |   |        |   |        | id  | description | timing | title       | professeur_id |              |   |
|--------------------------|---|--------|---|--------|---|-------------|--------|-------------|---------------|--------------|---|
| <input type="checkbox"/> |  | Éditer |  | Copier |  | Supprimer   | 1      | intéressant | 5             | math         | 1 |
| <input type="checkbox"/> |  | Éditer |  | Copier |  | Supprimer   | 2      | intéressant | 5             | physique     | 3 |
| <input type="checkbox"/> |  | Éditer |  | Copier |  | Supprimer   | 3      | long        | 5             | informatique | 2 |

