Session: Spring Data JPA with Hibernate Part 3 <u>Assignment</u>

o Create a class Address for Author with instance variables streetNumber, location, State.

```
@Embeddable
public class Address {
   private int streetNumber;
   private String location;
   private String state;
```

 Create instance variable of Address class inside Author class and save it as embedded object.

```
@Entity
@Table(name = "author")
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int authorId;
    private String authorName;

@Embedded
Address address;
```

o Introduce a List of subjects for author.

```
@Entity
@Table(name = "author")
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int authorId;
    private String authorName;

@Embedded
Address address;

@OneToMany(mappedBy = "author", cascade = CascadeType.ALL)
    private Set<Subject> subjects;
```

```
@Entity
@Table(name = "subject")
public class Subject {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int subjectId;
    private String subjectName;

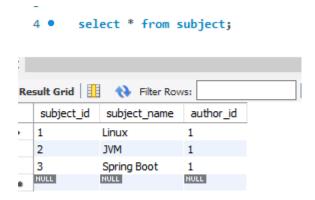
@ManyToOne
@JoinColumn(name = "author_id")
    private Author author;
```

o Persist 3 subjects for each author.

```
@Service
public class AuthorDaoService {
    @Autowired
    AuthorRepository authorRepository;
    public void addAuthorData() {
        Author author = new Author();
        author.setAuthorName("Dev");
        Address authorAddress = new Address();
        authorAddress.setStreetNumber(18);
        authorAddress.setLocation("India");
        authorAddress.setState("Mumbai");
        author.setAddress(authorAddress);
        Subject subject1 = new Subject();
        Subject subject2 = new Subject();
        Subject subject3 = new Subject();
        subject1.setSubjectName("Linux");
        subject2.setSubjectName("JVM");
        subject3.setSubjectName("Spring Boot");
        author.addSubject(subject1);
        author.addSubject(subject2);
        author.addSubject(subject3);
        System.out.println(author);
        authorRepository.save(author);
```

4 • select * from author;



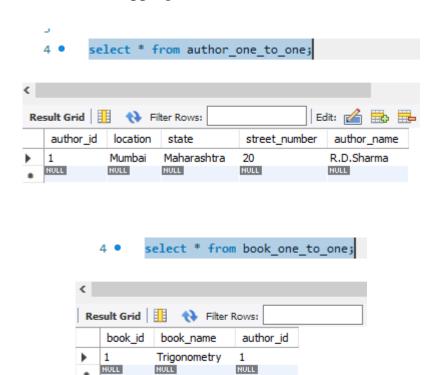


o Create an Entity book with an instance variable bookName.

```
@Entity
public class BookOneToOne {

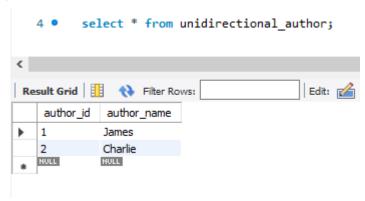
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int bookId;
    private String bookName;
```

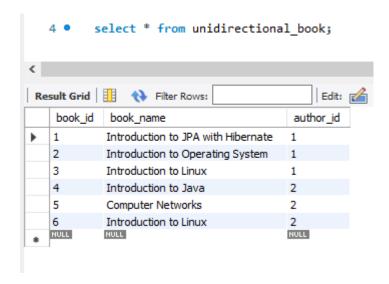
o Implement One to One mapping between Author and Book.



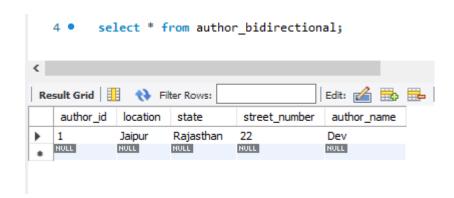
o Implement One to Many Mapping between Author and Book(Unidirectional, BiDirectional and without additional table) and implement cascade save.

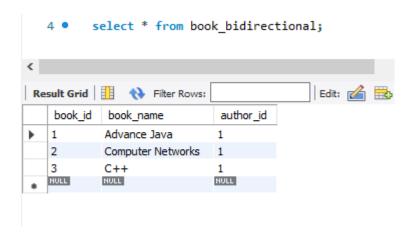
Unidirectional:



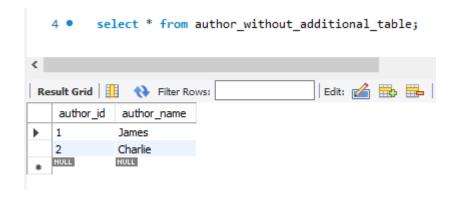


Bidirectional:

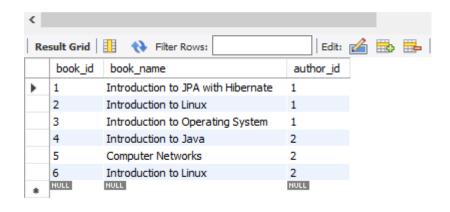




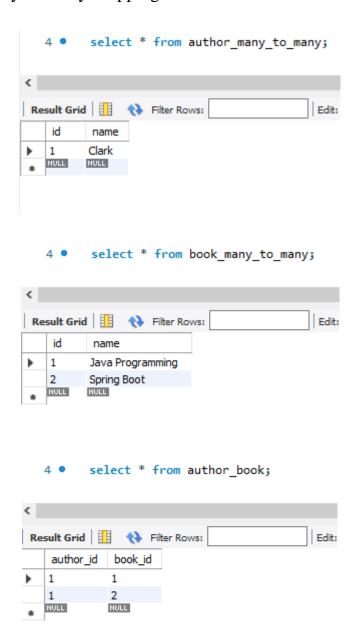
Without Additional Table:



4 • select * from book_without_additional_table;



o Implement Many to Many Mapping between Author and Book.



- Which method on the session object can be used to remove an object from the cache?
 - evict() method is used to remove an object from the cache.
 - ♣ After detaching the object from the session, any change to object will not be persisted.

• What does @transactional annotation do?

@Transactional annotation is used when we want the certain method/class to be executed in a transaction. When we call the method annotated with @Transactional, all or none of the writes on the database is executed.

In the case of read operations it is not useful and so it is in case of a single atomic write. If we are using it in a single read (select) the @Transactional annotation has no impact.

In the above example, if we do not annotate the transfer method with @Transactional annotation, it will update the firstAccount with and no updation on second Account. @Transactional annotation will rollback the transaction after exception occurred.