Session: Spring Data JPA with Hibernate Part 3

Assignment

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

```java
@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.

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

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

    @Embedded
    Address address;
```

- Introduce a List of subjects for author.

```java
@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;
```

```java
@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;
```

- Persist 3 subjects for each author.

```java
@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;
```

Result Grid | Filter Rows: | Edit:

author_id	location	state	street_number	author_name
1	India	Mumbai	18	Dev
NULL	NULL	NULL	NULL	NULL

```
4 •  select * from subject;
```

Result Grid | Filter Rows:

subject_id	subject_name	author_id
1	Linux	1
2	JVM	1
3	Spring Boot	1
NULL	NULL	NULL

- Create an Entity book with an instance variable bookName.

```
@Entity
public class BookOneToOne {

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

- Implement One to One mapping between Author and Book.

```
4 •  select * from author_one_to_one;
```

Result Grid | Filter Rows: | Edit:

author_id	location	state	street_number	author_name
1	Mumbai	Maharashtra	20	R.D.Sharma
NULL	NULL	NULL	NULL	NULL

```
4 •  select * from book_one_to_one;
```

Result Grid | Filter Rows:

book_id	book_name	author_id
1	Trigonometry	1
NULL	NULL	NULL

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

  Unidirectional:

```
4 •  select * from unidirectional_author;
```

Result Grid | Filter Rows: | Edit:

author_id	author_name
1	James
2	Charlie
NULL	NULL

```
4 •  select * from unidirectional_book;
```

Result Grid | Filter Rows: | Edit:

book_id	book_name	author_id
1	Introduction to JPA with Hibernate	1
2	Introduction to Operating System	1
3	Introduction to Linux	1
4	Introduction to Java	2
5	Computer Networks	2
6	Introduction to Linux	2
NULL	NULL	NULL

  Bidirectional:

```
4 •  select * from author_bidirectional;
```

Result Grid | Filter Rows: | Edit:

author_id	location	state	street_number	author_name
1	Jaipur	Rajasthan	22	Dev
NULL	NULL	NULL	NULL	NULL

```
4 •  select * from book_bidirectional;
```

Result Grid | Filter Rows: | Edit:

book_id	book_name	author_id
1	Advance Java	1
2	Computer Networks	1
3	C++	1
NULL	NULL	NULL

Without Additional Table:

```
4 •  select * from author_without_additional_table;
```

Result Grid | Filter Rows: | Edit:

author_id	author_name
1	James
2	Charlie
NULL	NULL

```
4 •  select * from book_without_additional_table;
```

Result Grid | Filter Rows: | Edit:

book_id	book_name	author_id
1	Introduction to JPA with Hibernate	1
2	Introduction to Linux	1
3	Introduction to Operating System	1
4	Introduction to Java	2
5	Computer Networks	2
6	Introduction to Linux	2
NULL	NULL	NULL

- Implement Many to Many Mapping between Author and Book.

```
4 •  select * from author_many_to_many;
```

Result Grid | Filter Rows: | Edit:

id	name
1	Clark
NULL	NULL

```
4 •  select * from book_many_to_many;
```

Result Grid | Filter Rows: | Edit:

id	name
1	Java Programming
2	Spring Boot
NULL	NULL

```
4 •  select * from author_book;
```

Result Grid | Filter Rows: | Edit:

author_id	book_id
1	1
1	2
NULL	NULL

- 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.

  E.g.,

```
@Transactional
        public void transfer(int amount)
        {
                BankAccount firstAccount = new BankAccount();
                firstAccount.setBalance(firstAccount.getBalance() - amount);
                repository.save(firstAccount);
                if(true){
                        throw new RuntimeException();
                }
                BankAccount secondAccount = new BankAccount();
                secondAccount.setBalance(secondAccount.getBalance() + amount);
                repository.save(secondAccount);
}
```

  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.