SPRING DATA JPA MAPPING

Many-to-One Uni-directional

The child will have a reference to the parent entity.

// First, let's create the University class :

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "uni")
public class University {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	private String name;

}
```

// Second, let's create the Student class :

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "student")
public class Student {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;
	private String name;

	@ManyToOne
	@JoinColumn(name = "uni_id")
	private University university;

}
```

Now, we have a unidirectional many-to-one mapping between the student and university class. Let's discuss some of the defined annotations :

@ManyToOne : this annotation defines a one to many relationship between the student entity and the university entity. This annotation means that many instances of this entity are mapped to one instance of another entity- Many students are in one university.
@JoinColumn : this annotation will create a column in the student table to store the primary key of the university table.

By default, @ManyToOne annotation will eagerly fetch the associated parent entity when the child entity is loaded from the database. That's why it is recommended to use lazy fetching with it to enhance the performance.

One-To-Many Uni-directional

```
// Author.java
import jakarta.persistence.*;
import lombok.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "authors")
@Data
public class Author {

   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   private Long authorId;

   private String name;

   @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
   @JoinColumn(name = "author_id", referencedColumnName = "authorId")
   private Set<Book> books = new HashSet<>();

   // Uni-directional One to Many mapping -> One Author can have Many Books
   // Here, extra column 'author_id' will be created on the many side of the relationship i.e. in the Books table
}

// Book.java

import jakarta.persistence.*;
import lombok.*;

@Entity
@Table(name = "books")
@Data
public class Book {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long bookId;

   private String title;
}
```

In this example:

The Author entity has a one-to-many relationship with the Book entity, and it's unidirectional (from Author to Book).
The Author entity has a set of Book objects, representing the books written by the author.
The @OneToMany annotation is used to define the one-to-many relationship. The cascade attribute is set to CascadeType.ALL to cascade all operations (e.g., save, update, delete) to the associated Book entities.
The @JoinColumn annotation is used to specify the foreign key column (author_id) in the books table that establishes the relationship. Thus, it specifies the foreign key column in the child entity's table that refers to the parent entity.
The 'mappedBy' attribute is not used in a unidirectional relationship as it's specific to bidirectional relationships.

```
// MainApplication Class ---

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
@Slf4j
public class SpringDataJpaApplication
{

 public static void main(String[] args)
 {
  ApplicationContext context = SpringApplication.run(SpringDataJpaApplication.class, args);

  AuthorRepository authorRepository = context.getBean(AuthorRepository.class);

  BookRepository bookRepository = context.getBean(BookRepository.class);


  // Create authors
  Author author1 = new Author();
  author1.setName("Deepak Kumar");

  Author author2 = new Author();
  author2.setName("Katty Janes");

  // Create books
  Book book1 = new Book();
  book1.setTitle("Welcome to CSS");

  Book book2 = new Book();
  book2.setTitle("Javascript Programming");

  // Associate books with authors
  author1.getBooks().add(book1);
  author2.getBooks().add(book2);

  // Save authors (and cascade to save associated books)
  authorRepository.save(author1);
  authorRepository.save(author2);

  // Retrieve and print saved authors (optional)
  System.out.println("Saved Authors:");
  authorRepository.findAll().forEach(System.out::println);

 }

}
```

LINKS ------

1)
---goodddd---
https://github.com/LuisSalas94/Spring-Data-JPA-Bookstore
https://blog.stackademic.com/the-complete-guide-to-spring-data-jpa-building-a-bookstore-application-from-scratch-part-i-3f8ba9d13b10
https://blog.stackademic.com/the-complete-guide-to-spring-data-jpa-building-a-bookstore-application-from-scratch-part-ii-28f70149fa9
---gooddd---
https://blog.stackademic.com/the-complete-guide-to-spring-data-jpa-building-a-bookstore-application-from-scratch-part-iii-8a1de3bc9949

---test---
https://medium.com/@srajas02/running-a-single-test-on-spring-boot-with-maven-commands-6816149b160c

2)
Spring Data JPA Relationship Mapping

---one-to-one---
https://medium.com/@mohamedyousife3/one-to-one-mapping-in-spring-jpa-82b3028ab365

---one-to-many---gooddd----
https://medium.com/@mohamedyousife3/one-to-many-mapping-in-spring-jpa-117d7717929b
---One-To-Many Uni-directional---
https://medium.com/@bectorhimanshu/spring-data-jpa-one-to-many-unidirectional-relationship-mapping-2a18ed985a5d

---many-to-many---
https://medium.com/@mohamedyousife3/many-to-many-in-spring-jpa-a9fa7bdb639f

3)
Spring Data JPA Relationship Mapping

--- Many-Many Uni-directional ---

---simple---
https://medium.com/@bectorhimanshu/spring-data-jpa-many-to-many-unidirectional-relationship-mapping-538649df20f6

---many-many Bi-direction---

---simple---
https://blog.tericcabrel.com/many-to-many-relationship-hibernate-part-1/
https://lorenzomiscoli.com/mapping-a-many-to-many-relationship-with-jpa-and-hibernate/
https://www.geeksforgeeks.org/jpa-many-to-many-mapping/

---relationship between two entities with additional information---
https://blog.tericcabrel.com/many-to-many-relationship-hibernate-part-2/
https://teddysmith.io/many-to-many-relationships-in-spring-boot-explained-simply/

---SBoot---API---
https://blog.tericcabrel.com/build-rest-api-spring-boot-mysql/
---CI/CD---
https://blog.tericcabrel.com/build-cicd-github-actions-deploy-nodejs/