در Spring Boot و با استفاده از JPA (Java Persistence API)، می‌توان روابط مختلف بین موجودیت‌ها (Entities) را تعریف کرد. این روابط شامل **یک به یک (One-to-One)**، **یک به چند (One-to-Many)**، و **چند به چند (Many-to-Many)** می‌باشد. در زیر، توضیحات مربوط به هر نوع رابطه و نحوه پیاده‌سازی آن در Entity ها آورده شده است.

1. رابطه یک به یک (One-to-One)

در این رابطه، هر رکورد در یک جدول با یک رکورد در جدول دیگر مرتبط است. برای مثال، فرض کنید که هر کاربر یک پروفایل دارد.

مثال: User و Profile

```java
import javax.persistence.*;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL)
    private Profile profile;

    //Getters and Setters
}

@Entity
public class Profile {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String bio;

    @OneToOne
    @JoinColumn(name = "user_id")
    private User user;

    //Getters and Setters
}
```

توضیحات:

- در کلاس `User`، `@OneToOne(mappedBy = "user", cascade = CascadeType.ALL)` نشان می‌دهد که رابطه از طرف `User` کنترل می‌شود و اگر `User` حذف شود، `Profile` نیز حذف خواهد شد.
- در کلاس `Profile`، `@JoinColumn(name = "user_id")` مشخص می‌کند که کلید خارجی در جدول `Profile` به کدام ستون در `User` اشاره می‌کند.

2. رابطه یک به چند (One-to-Many)

در این رابطه، یک رکورد در یک جدول می‌تواند به چند رکورد در جدول دیگر مرتبط شود. به عنوان مثال، هر دسته‌بندی می‌تواند چندین محصول داشته باشد.

مثال: Category و Product

```java
import javax.persistence.*;
import java.util.Set;

@Entity
public class Category {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "category", cascade = CascadeType.ALL)
    private Set<Product> products;

    //Getters and Setters
}

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToOne
    @JoinColumn(name = "category_id")
    private Category category;

    //Getters and Setters
}
```

توضیحات:

- در کلاس `Category`، `@OneToMany(mappedBy = "category", cascade = CascadeType.ALL)` نشان می‌دهد که یک دسته‌بندی می‌تواند چندین محصول داشته باشد و اگر یک دسته‌بندی حذف شود، محصولات آن نیز حذف خواهند شد.
- در کلاس `Product`، `@ManyToOne` مشخص می‌کند که چندین محصول می‌توانند به یک دسته‌بندی تعلق داشته باشند.

3. رابطه چند به چند (Many-to-Many)

در این رابطه، هر رکورد در یک جدول می‌تواند با چندین رکورد در جدول دیگر مرتبط شود و برعکس. به عنوان مثال، دانش‌آموزان و کلاس‌ها.

مثال: Student و Course

```java
import javax.persistence.*;
import java.util.Set;

@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(
     name = "student_course",
     joinColumns = @JoinColumn(name = "student_id"),
     inverseJoinColumns = @JoinColumn(name = "course_id")
(
    private Set<Course> courses;

    //Getters and Setters
}

@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @ManyToMany(mappedBy = "courses")
    private Set<Student> students;
```

```
    //Getters and Setters
}
```

توضیحات:

- در کلاس `Student`، `@ManyToMany` نشان می‌دهد که هر دانش‌آموز می‌تواند در چندین کلاس ثبت‌نام کند و `@JoinTable` مشخص می‌کند که جدول میانی (در اینجا `student_course`) برای ذخیره‌سازی کلیدهای خارجی کدام است.
- در کلاس `Course`، `mappedBy = "courses"` مشخص می‌کند که این طرف رابطه، سمت دیگر را کنترل می‌کند.

4. جمع‌بندی

- **رابطه یک به یک (One-to-One)**: هر رکورد در یک جدول با یک رکورد در جدول دیگر مرتبط است.
- **رابطه یک به چند (One-to-Many)**: یک رکورد در یک جدول می‌تواند به چند رکورد در جدول دیگر مرتبط باشد.
- **رابطه چند به چند (Many-to-Many)**: هر رکورد در یک جدول می‌تواند با چندین رکورد در جدول دیگر مرتبط شود و برعکس.

نکات مهم

- **Cascade**: از `cascade` برای تعیین رفتارهای خاص هنگام حذف یا به‌روزرسانی رکوردها استفاده می‌شود. برای مثال، اگر `CascadeType.ALL` را تنظیم کنید، هر تغییر در رکوردهای والد بر رکوردهای فرزند نیز تأثیر خواهد گذاشت.
- **Fetch Type**: می‌توانید نوع بارگذاری داده‌ها (`FetchType.LAZY` یا `FetchType.EAGER`) را برای بهینه‌سازی عملکرد تعیین کنید.

با استفاده از این روابط، می‌توانید ساختار داده‌های خود را به طور موثری طراحی کنید و از قدرت JPA برای مدیریت داده‌ها بهره ببرید.