1. Create a new Spring Boot project with Spring Data JPA set up.
2. Define at least three JPA entities with different types of relationships. For example, you could define **`Author`**, **`Book`**, and **`Category`** entities, with the following relationships:
    - An **`Author`** can write many **`Book`**s`, and a **`Book`** can be written by one **`Author`**. Use `**@ManyToOne`** and `**@OneToMany`** annotations to define this relationship, and use `**@JoinColumn`** to specify the foreign key column name.
    - A **`Book`** can belong to many categories, and a **`Category`** can have many books. Use `**@ManyToMany`** and `**@JoinTable`** annotations to define this relationship, and use `**mappedBy`** to specify the inverse relationship in the **`Book`** entity.
3. Use `**FetchType.EAGER`** or `**FetchType.LAZY`** to specify the fetching strategy for each relationship. For example, you might want to eagerly fetch the **`Author`** entity whenever you retrieve a **`Book`**, but lazily fetch the **`Category`** entities. Use the **`fetch`** attribute of the relationship annotations to specify the fetching strategy.
4. Implement a simple CRUD interface for each entity using Spring Data JPA's built-in repository methods, and test the entity relationships by creating, reading, updating, and deleting instances of each entity.
5. Use JPA's entity manager to write some custom queries that join two or more entities, and test those queries to ensure that they return the expected results.

Here's an example implementation for the **`Book`** entity, with `**@ManyToOne`** and `**@OneToMany`** relationships:

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

    private String title;

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

    @OneToMany(mappedBy = "book")
    private List<Review> reviews;

    // constructors, getters, and setters
}
```

This code defines a **`Book`** entity with a `**@ManyToOne`** relationship to an **`Author`** entity, and a `**@OneToMany`** relationship to a **`Review`** entity. The `**@JoinColumn`** annotation specifies the name of the foreign key column in the **`Book`** table that points to the **`Author`** table. The **`mappedBy`** attribute in the `**@OneToMany`** annotation specifies the name of the inverse relationship in the **`Review`** entity.

You can implement the other entities and relationships similarly, using the appropriate annotations and fetching strategies.