ORM (Object-Relational Mapping) **là một kỹ thuật cho phép bạn truy vấn và thao tác dữ liệu từ cơ sở dữ liệu bằng cách sử dụng mô hình hướng đối tượng**. Trong Spring Data JPA, ORM được sử dụng để ánh xạ các lớp Java (các thực thể - entity) với các bảng trong cơ sở dữ liệu và thực hiện các thao tác với cơ sở dữ liệu thông qua các thực thể này.

Spring Data JPA cung cấp một triển khai mạnh mẽ của đặc tả JPA, giúp đơn giản hóa tương tác giữa các ứng dụng Java và các cơ sở dữ liệu quan hệ. Nó sử dụng các annotation để định nghĩa ánh xạ giữa các lớp Java và các bảng trong cơ sở dữ liệu, cũng như mối quan hệ giữa các thực thể khác nhau.

**1. @Entity**

**Lý Thuyết:**

Annotation @Entity được sử dụng để đánh dấu một lớp Java là một thực thể (entity) và nó sẽ được ánh xạ tới một bảng trong cơ sở dữ liệu.

**Ví dụ từ các file:**

@Entity  
@Table(name = "customer")  
@Getter  
@Setter  
public class CustomerEntity extends BaseEntity {  
  
 private String name;  
 private String email;  
 private String phoneNumber;  
 private String address;  
 private String bankNumber;  
 private String bankName;

Trong ví dụ trên, CustomerEntity là một thực thể và sẽ được ánh xạ tới một bảng trong cơ sở dữ liệu với tên mặc định là CustomerEntity.

**2. @Table**

**Lý Thuyết:**

Annotation @Table cho phép bạn định rõ tên bảng và các thuộc tính khác liên quan tới bảng mà thực thể sẽ ánh xạ tới.

**Ví dụ từ các file:**

@Entity  
@Table(name = "customer")  
@Getter  
@Setter  
public class CustomerEntity extends BaseEntity {  
  
 private String name;  
 private String email;  
 private String phoneNumber;  
 private String address;  
 private String bankNumber;  
 private String bankName;

Ở đây, CustomerEntity sẽ được ánh xạ tới bảng customer trong cơ sở dữ liệu thay vì tên mặc định là CustomerEntity.

**3. @Id**

**Lý Thuyết:**

Annotation @Id được sử dụng để xác định thuộc tính nào sẽ là khóa chính của thực thể.

**Ví dụ từ các file:**

public abstract class BaseEntity {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;

Trong ví dụ trên, thuộc tính id trong ClaimEntity là khóa chính của bảng claim.

**4. @GeneratedValue**

**Lý Thuyết:**

Annotation @GeneratedValue được sử dụng để định nghĩa chiến lược tạo giá trị tự động cho khóa chính.

**Ví dụ từ các file:**

@Data  
public abstract class BaseEntity {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;

Ở đây, id được tự động sinh với chiến lược IDENTITY, tức là giá trị sẽ được tự động tăng dựa trên cơ chế của cơ sở dữ liệu.

**5. @OneToOne**

**Lý Thuyết:**

Annotation @OneToOne định nghĩa mối quan hệ một - một giữa hai thực thể. Mối quan hệ này thường được sử dụng khi một thực thể có một tham chiếu duy nhất tới một thực thể khác.

**Ví dụ từ các file:**

@Entity  
@Table(name = "claim\_document")  
@Getter  
@Setter  
public class ClaimDocumentEntity extends BaseEntity {  
  
 @OneToOne  
 @JoinColumn(name = "claim\_id")  
 private ClaimEntity claimEntity;

Trong ví dụ này, ClaimDocumentEntity có mối quan hệ một - một với ClaimEntity qua cột claim\_id.

**6. @OneToMany**

**Lý Thuyết:**

Annotation @OneToMany định nghĩa mối quan hệ một - nhiều giữa hai thực thể. Một thực thể có thể có nhiều thực thể con.

**Ví dụ từ các file:**

@Entity

@Table(name = "customer")

public class CustomerEntity {

@OneToMany(mappedBy = "customer")

private List<ClaimEntity> claims;

// Các trường, getter, setter

}

Ở đây, CustomerEntity có thể có nhiều ClaimEntity, và mối quan hệ này được xác định thông qua thuộc tính customer trong ClaimEntity.

**7. @ManyToOne**

**Lý Thuyết:**

Annotation @ManyToOne định nghĩa mối quan hệ nhiều - một giữa hai thực thể. Nhiều thực thể có thể tham chiếu tới cùng một thực thể cha.

**Ví dụ từ các file:**

@Entity  
@Table(name = "claim")  
@Getter  
@Setter  
public class ClaimEntity extends BaseEntity {  
  
 private String code;  
  
 @ManyToOne  
 @JoinColumn(name = "customer\_id")  
 private CustomerEntity customerEntity;

Trong ví dụ này, nhiều ClaimEntity có thể tham chiếu tới một CustomerEntity qua cột customer\_id.

**8. @ManyToMany**

**Lý Thuyết:**

Annotation @ManyToMany định nghĩa mối quan hệ nhiều - nhiều giữa hai thực thể. Cả hai thực thể có thể có nhiều tham chiếu tới nhau.

**Ví dụ từ các file:**

@Entity  
@Table(name = "users")  
@Getter  
@Setter  
public class UserEntity extends BaseEntity {  
  
 private String username;  
 private String password;  
 private String code;  
 private String email;  
 private String firstName;  
 private String lastName;  
 private String phone;  
 private String address;  
  
 @ManyToMany(fetch = FetchType.*EAGER*, cascade = CascadeType.*ALL*)  
 @JoinTable(  
 name = "user\_roles",  
 joinColumns = @JoinColumn(name = "user\_id"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id")  
 )  
 private Set<RoleEntity> roles = new HashSet<>();

Ở đây, UserEntity có mối quan hệ nhiều - nhiều với RoleEntity, được thể hiện qua bảng nối user\_role với các cột user\_id và role\_id.

**9. @JoinColumn**

**Lý Thuyết:**

Annotation @JoinColumn được sử dụng để định nghĩa cột trong bảng con, thông qua đó mối quan hệ giữa các thực thể được xác định.

**Ví dụ từ các file:**

@Entity  
@Table(name = "claim\_document")  
@Getter  
@Setter  
public class ClaimDocumentEntity extends BaseEntity {  
  
 @ManyToOne  
 @JoinColumn(name = "claim\_id")  
 private ClaimEntity claimEntity;

Trong ví dụ này, JoinColumn xác định claim\_id là cột được sử dụng để ánh xạ mối quan hệ một - một giữa ClaimDocumentEntity và ClaimEntity.

**10. @JoinTable**

**Lý Thuyết:**

Annotation @JoinTable được sử dụng trong mối quan hệ nhiều - nhiều để xác định bảng nối và các cột của nó.

**Ví dụ từ các file:**

@Entity  
@Table(name = "users")  
@Getter  
@Setter  
public class UserEntity extends BaseEntity {  
  
 private String username;  
 private String password;  
 private String code;  
 private String email;  
 private String firstName;  
 private String lastName;  
 private String phone;  
 private String address;  
  
 @ManyToMany(fetch = FetchType.*EAGER*, cascade = CascadeType.*ALL*)  
 @JoinTable(  
 name = "user\_roles",  
 joinColumns = @JoinColumn(name = "user\_id"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id")  
 )  
 private Set<RoleEntity> roles = new HashSet<>();

Trong ví dụ trên, bảng nối user\_role với cột user\_id và role\_id được sử dụng để thiết lập mối quan hệ nhiều - nhiều giữa UserEntity và RoleEntity.

**Tóm tắt các Annotation với các Entity**

Dưới đây là cách các annotation được sử dụng trong các class bạn đã cung cấp:

**1. BaseEntity.java**

@MappedSuperclass

public abstract class BaseEntity {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

// Thông thường có thể thêm các trường khác như createdDate, modifiedDate, v.v.

}

**2. ClaimEntity.java**

@Entity

@Table(name = "claim")

public class ClaimEntity extends BaseEntity {

@ManyToOne

@JoinColumn(name = "customer\_id")

private CustomerEntity customer;

@OneToMany(mappedBy = "claim")

private List<ClaimDocumentEntity> claimDocuments;

@ManyToOne

@JoinColumn(name = "status\_id")

private ClaimStatusEntity status;

// Các trường khác, getter, setter

}

**3. ClaimDocumentEntity.java**

@Entity

@Table(name = "claim\_document")

public class ClaimDocumentEntity extends BaseEntity {

@OneToOne

@JoinColumn(name = "claim\_id")

private ClaimEntity claim;

// Các trường khác, getter, setter

}

**4. ClaimStatusEntity.java**

@Entity

@Table(name = "claim\_status")

public class ClaimStatusEntity extends BaseEntity {

@OneToMany(mappedBy = "status")

private List<ClaimEntity> claims;

// Các trường khác, getter, setter

}

**5. CustomerEntity.java**

@Entity

@Table(name = "customer")

public class CustomerEntity extends BaseEntity {

@OneToMany(mappedBy = "customer")

private List<ClaimEntity> claims;

// Các trường khác, getter, setter

}

**6. InsuranceProductEntity.java**

@Entity

@Table(name = "insurance\_product")

public class InsuranceProductEntity extends BaseEntity {

// Ví dụ về quan hệ một - nhiều có thể thêm vào đây

// Các trường khác, getter, setter

}

**7. RoleEntity.java**

@Entity

@Table(name = "role")

public class RoleEntity extends BaseEntity {

@ManyToMany(mappedBy = "roles")

private List<UserEntity> users;

// Các trường khác, getter, setter

}

**8. UserEntity.java**

@Entity

@Table(name = "user")

public class UserEntity extends BaseEntity {

@ManyToMany

@JoinTable(

name = "user\_role",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "role\_id")

)

private List<RoleEntity> roles;

// Các trường khác, getter, setter

}

Trên đây là các ví dụ và giải thích chi tiết về cách sử dụng các annotation trong Spring Data JPA, dựa trên các file bạn đã cung cấp. Các mối quan hệ như @OneToOne, @OneToMany, @ManyToOne, và @ManyToMany đã được thể hiện qua các thực thể trong ví dụ.