

스프링 스타-디

레이블 관계 매핑 + 배포



테이블 관계 매핑

테이블 관계란?

하나의 테이블이 다른 테이블의 데이터를
참조하거나 연결되어 있는 구조

종류는 크게 3가지 정도

- 1:1
- 1:N
- N:M



테이블 관계 매핑

스프링에서 관계 설정

스프링에서 관계 어노테이션 종류

1. `OneToOne`
2. `OneToMany`
3. `ManyToOne`
4. `ManyToMany`

큰 틀은 비슷한데, 조금씩 사용법이 다름



테이블 관계 매핑

우선 만들 테이블들

1. UserEntity
2. UserDetailsEntity
3. BookEntity
4. CategoryEntity

UserEntity

```
public class UserEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @Column  
    private String email;  
  
    @Column  
    private Long age;  
}
```

테이블 관계 매핑

우선 만들 테이블들

1. UserEntity
2. UserDetailEntity
3. BookEntity
4. CategoryEntity

UserDetailEntity

```
public class UserDetailEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String phone;  
  
    @Column  
    private String email;  
}
```

테이블 관계 매핑

우선 만들 테이블들

1. UserEntity
2. UserDetailsEntity
3. BookEntity
4. CategoryEntity

BookEntity

```
public class BookEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String title;  
  
    @Column  
    private String author;  
  
    @Column  
    private String publisher;  
}
```

테이블 관계 매핑

우선 만들 테이블들

1. UserEntity
2. UserDetailsEntity
3. BookEntity
4. CategoryEntity

CategoryEntity

```
public class CategoryEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
}
```

테이블 관계 매핑

OneToOne 어노테이션

사용할 때 **두 가지 방법**이 있음

1. 단방향 관계 설정
2. 양방향 관계 설정

개인적으로 단방향이 좀 편하긴 함

아래는 단방향 연결 예시

```
public class UserEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @Column  
    private String email;  
  
    @Column  
    private Long age;  
  
    @OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)  
    @JoinColumn(name = "detail_id", referencedColumnName = "id")  
    private UserDetailEntity userDetail;  
}
```


테이블 관계 매핑

OneToOne 어노테이션

단방향 매핑 방법

1. 1:1로 연결할 객체 중 **관계의 주인**으로 쓸 객체 열기
2. 좌측과 같이 어노테이션을 설정
3. 해당 어노테이션에 사용할 수 있는 속성들은 따로 설명
4. 필드는 Entity로 설정

아래는 단방향 연결 예시

```
public class UserEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @Column  
    private String email;  
  
    @Column  
    private Long age;  
  
    @OneToOne  
    @JoinColumn(name = "detail_id", referencedColumnName = "id")  
    private UserDetailEntity userDetail;  
}
```

테이블 관계 매핑

OneToOne 어노테이션

OneToOne 어노테이션의 속성

1. **Cascade** 설정하여 수정 삭제 전파 가능
2. **orphanRemoval**을 설정하여 부모 객체 삭제 시 자식 객체 자동 삭제 가능

Cascade 예시

```
@OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)  
@JoinColumn(name = "detail_id", referencedColumnName = "id")  
private UserDetailsEntity userDetails;
```

테이블 관계 매핑

OneToOne 어노테이션

JoinColumn 어노테이션의 속성

1. 이것이 선언된 필드는 **외래키**로 작동
2. **name**을 이용해 필드명 설정 (별명 느낌)
3. **referencedColumnName**으로 참조할 id
를 지정 (안쓰면 **@Id**인 필드로 자동 설정)
4. nullable도 설정 가능

JoinColumn 예시

```
@OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)
@JoinColumn(name = "detail_id", referencedColumnName = "id")
private UserDetailEntity userDetail;
```

테이블 관계 매핑

OneToOne 어노테이션

양방향 매핑 방법

1. 1:1로 연결할 객체 중 **관계의 주인**
2. 좌측과 같이 어노테이션을 설정
3. 상대 객체에도 좌측과 같이 설정
4. 필드는 Entity로 설정

아래는 양방향 연결 예시

1, 2

```
public class UserEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @Column  
    private String email;  
  
    @Column  
    private Long age;  
  
    @OneToOne(mappedBy = "user")  
    private UserDetailsEntity userDetails;  
}
```

테이블 관계 매핑

OneToOne 어노테이션

양방향 매핑 방법

1. 1:1로 연결할 객체 중 **관계의 주인**
2. 좌측과 같이 어노테이션을 설정
3. 상대 객체에도 좌측과 같이 설정
4. 필드는 Entity로 설정

아래는 양방향 연결 예시

3

```
public class UserDetailsEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String phone;  
  
    @Column  
    private String email;  
  
    @OneToOne  
    @JoinColumn(name = "user_id", referencedColumnName = "id")  
    private UserEntity user;  
}
```

테이블 관계 매핑

OneToOne 어노테이션

OneToOne 어노테이션의 속성 - 2

1. mappedBy를 이용하여 상대 쪽에서 읽기
전용으로 주인 객체 참조
2. 이 경우엔 Cascade 사용 불가
3. 주인 객체의 경우는 전과 동일

mappedBy 예시

```
@OneToOne(mappedBy = "user")  
private UserDetailsEntity userDetails;
```

테이블 관계 매핑

OneToOne 어노테이션

관계의 주인이 뭔데요?

1. 도메인의 부모/자식과는 좀 다름
2. 외래키가 있는 쪽이 연관관계의 주인
3. 즉, 외래키를 관리하는 엔티티
4. 외래키가 있다는건 어떻게 보나?
 - a. **JoinColumn**이 달린 필드가 외래키
5. 설계할 때 부모쪽에서 관리하도록 해주는 게 편할겁니다

mappedBy 예시

```
@OneToOne(mappedBy = "user")  
private UserDetailsEntity userDetails;
```

테이블 관계 매핑

OneToMany /ManyToOne

사용할 때 두 가지 방법이 있음

1. 단방향 관계 설정
2. 양방향 관계 설정

개인적으로 이건

ManyToOne 단방향이 좀 편하긴 함

하지만 Cascade를 쓸 수 있다는 점에서

OneToMany 단방향도 괜찮은 선택지

아래는 OneToMany 단방향 연결 예시

```
public class CategoryEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)  
    @JoinColumn(name = "category_id")  
    private List<BookEntity> books = new ArrayList<>();  
}
```


테이블 관계 매핑

OneToMany /ManyToOne

OneToMany 단방향 매핑 방법

1. 1:N 으로 연결할 객체 중 **관계의 주인**으로
쓸 객체 열기
2. 좌측과 같이 어노테이션을 설정
3. 해당 어노테이션에 사용할 수 있는 속성들
은 **OneToOne**과 동일
4. 필드는 필요에 따라 List 또는 Map, Set

아래는 **OneToMany** 단방향 연결 예시

```
public class CategoryEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)  
    @JoinColumn(name = "category_id")  
    private List<BookEntity> books = new ArrayList<>();  
}
```

테이블 관계 매핑

OneToMany /ManyToOne

왜 `JoinColumn`에 `category_id` 인가요?

1. 일단 관계의 주인은 `Category` 테이블임
2. 하지만, 생성된 외래키를 가지는건 `Book`
3. 즉 관계의 주인과 외래키를 갖는곳이 다름
4. **단방향 설계**를 하면 이 점을 주의

아래는 `OneToMany` 단방향 연결 예시

```
public class CategoryEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)  
    @JoinColumn(name = "category_id")  
    private List<BookEntity> books = new ArrayList<>();  
}
```

테이블 관계 매핑

OneToMany /ManyToOne

ManyToOne 단방향 매핑 방법

1. N:1 로 연결할 객체 중 **관계의 주인**으로
쓸 객체 열기
2. 좌측과 같이 어노테이션을 설정
3. **OneToOne**과 거의 동일함
4. 필드는 Entity로

아래는 **ManyToOne** 단방향 연결 예시

```
public class BookEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String title;  
  
    @Column  
    private String author;  
  
    @Column  
    private String publisher;  
  
    @ManyToOne  
    @JoinColumn(name = "category_id", referencedColumnName = "id")  
    private CategoryEntity category;  
}
```

테이블 관계 매핑

OneToMany /ManyToOne

1:N, N:1 양방향 매핑 방법

1. 1:N 으로 연결할 **관계의 주인**에 좌측과 같이 설정
2. N인 테이블엔 **ManyToOne**으로 설정
3. 끝입니다.

아래는 양방향 연결 예시

```
public class CategoryEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @OneToMany(mappedBy = "category")  
    private List<BookEntity> books = new ArrayList<>();  
}
```

BookEntity의 컬럼

```
@ManyToOne  
@JoinColumn(name = "category_id", referencedColumnName = "id")  
private CategoryEntity category;
```

테이블 관계 매핑

ManyToMany

사용할 때 **두 가지 방법**이 있음

1. **ManyToMany**를 직접 사용
2. 중간 테이블을 만들어 **1:N, N:1**로 연결

실무에선 **2번**을 많이 사용한다고 합니다.

ManyToMany를 사용할 일이 많진 않습니다

그리고 **ManyToMany**가 짝셉니다

일단 해봅시다.

아래는 **ManyToMany** 연결 예시

```
public class UserEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @Column  
    private String email;  
  
    @Column  
    private Long age;  
  
    @OneToOne(mappedBy = "user")  
    private UserDetailEntity userDetail;  
  
    @ManyToMany  
    @JoinTable(  
        name = "user_friends",  
        joinColumns = @JoinColumn(name = "user_id"),  
        inverseJoinColumns = @JoinColumn(name = "friend_id")  
    )  
    private List<UserEntity> friends;  
}
```

테이블 관계 매핑

ManyToMany

ManyToMany 매핑 방법

1. N:M 으로 연결할 객체 중 **관계의 주인**으로 쓸 객체 열기
2. 좌측과 같이 어노테이션을 설정
3. 애는 **JoinTable**로 상대 테이블을 조인
4. **JoinTable**은 다음 페이지에서 설명
5. 자료형은 List, Map, Set 중 자유

아래는 ManyToMany 연결 예시

```
public class UserEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @Column  
    private String email;  
  
    @Column  
    private Long age;  
  
    @OneToOne(mappedBy = "user")  
    private UserDetailEntity userDetail;  
  
    @ManyToMany  
    @JoinTable(  
        name = "user_friends",  
        joinColumns = @JoinColumn(name = "user_id"),  
        inverseJoinColumns = @JoinColumn(name = "friend_id")  
    )  
    private List<UserEntity> friends;
```


테이블 관계 매핑

ManyToMany

JoinTable 저건 어떻게 쓰나요?

1. `name`에 생성될 테이블 이름 설정
2. `joinColumns`에 나의 테이블 컬럼
3. `inverseJoinColumns`엔 상대 테이블
 - a. 상대 테이블은 **아래 필드 엔티티임**
4. 자가참조나 다른 테이블 참조나 동일함

아래는 **ManyToMany** 연결 예시

```
public class UserEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column  
    private String name;  
  
    @Column  
    private String email;  
  
    @Column  
    private Long age;  
  
    @OneToOne(mappedBy = "user")  
    private UserDetailsEntity userDetails;  
  
    @ManyToMany  
    @JoinTable(  
        name = "user_friends",  
        joinColumns = @JoinColumn(name = "user_id"),  
        inverseJoinColumns = @JoinColumn(name = "friend_id")  
    )  
    private List<UserEntity> friends;  
}
```

테이블 관계 매핑

ManyToMany

중간 테이블 분리로 사용하는 법

1. 정보를 저장할 엔티티 생성
2. N:M 중 N쪽과 1:N
3. M쪽과는 N:1
4. 기타 필드는 필요한 대로 설정

1. 정보를 저장할 엔티티 생성

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


테이블 관계 매핑

ManyToMany

중간 테이블 분리로 사용하는 법

1. 정보를 저장할 엔티티 생성
2. N:M 중 N쪽과 N:1, M쪽도 N:1
3. 기타 필드는 필요한 대로 설정

2. 1:N, N:1 매핑

```
public class FriendEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @ManyToOne  
    @JoinColumn(name = "user_id", referencedColumnName = "id")  
    private UserEntity user;  
  
    @ManyToOne  
    @JoinColumn(name = "friend_id", referencedColumnName = "id")  
    private UserEntity friend;  
}
```

테이블 관계 매핑

ManyToMany

중간 테이블 분리로 사용하는 법

1. 정보를 저장할 엔티티 생성
2. N:M 중 N쪽과 N:1, M쪽도 N:1
3. 기타 필드는 필요한 대로 설정

3. 예시로 만든 필드들

```
public class FriendEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @ManyToOne  
    @JoinColumn(name = "user_id", referencedColumnName = "id")  
    private UserEntity user;  
  
    @ManyToOne  
    @JoinColumn(name = "friend_id", referencedColumnName = "id")  
    private UserEntity friend;  
  
    @Column  
    private LocalDateTime createdAt;  
}
```

실습 과제 나갑니다

도서 대출 시스템 만들기

요구사항 및 기능들은 파일로 올려두겠습니다.

추가기능, 추추가기능 달성 시 해커톤 팀 구성에 **가산점** 들어갈 예정입니다.
열심히 해주십시오.

이정도 만드실 수 있으면 어느정도 스프링 기초는 댄 수준입니다

감사합니다

스프링 스타-디