(스터디)7장 고급 매핑

7.1 상속 관계 매핑 p.244

- 상속관계 매핑 : 객체의 상속 구조와 데이터베이스의 슈퍼타입 서브타입 관계를 매핑하는 것

7.1.1 조인 전략(Joined Strategy) p.245

- 엔티티 각각을 모두 테이블로 만듦.
- 자식 테이블이 부모 테이블의 기본 키를 받아서 기본 키 + 외래 키로 사용하는 전략
- 테이블은 타입의 개념이 없기 때문에 타입을 구분하는 컬럼을 추가해야 한다.

```
@Inheritance(strategy = InheritanceType.XXX)
```

- 상속 매핑은 부모 클래스에 `@Inheritance` 를 상용해야 한다.
- JOINED : 조인 전략
- SINGLE_TABLE : 단일 테이블 전략
- TABLE_PER_CLASS : 구현 클래스마다 테이블 전략

```
@DiscriminatorColumn(name = "DTYPE")
```

- 부모 클래스에 구분 컬럼을 지정한다.
- 이 컬럼으로 저장된 자식 테이블을 구할 수 있다.
- 기본값이 `"DYPTE"` 이므로 `@DiscriminatorColumn` 로 줄여 사용해도 된다.

```
@DiscriminatorValue("M")
```

- 엔티티를 저장할 때 구분 컬럼에 입력할 값을 저장한다.

  ex) 영화 엔티티를 저장하면 구분 컬럼인 `DTYPE` 에 값이 M 이 저장된다.

@PrimaryKeyJoinColumn

- 기본값으로 자식 테이블은 부모 테이블의 ID 컬럼명을 그대로 사용한다.
- 만약 자식 테이블의 기본 키 컬럼명을 변경하고 싶을 때 이 어노테이션을 사용하면 된다.

조인 전략

- 장점
  - 테이블이 정규화된다.
  - 외래 키 참조 무결성 제약조건을 활용할 수 있다(외래키는 참조할 수 없는 값을 가질 수 없다는 규칙).
  - 저장공간을 효율적으로 사용한다.
- 단점
  - 조회할 때 조인이 많이 사용되므로 성능이 저하될 수 있다.
  - 조회 쿼리가 복잡하다.
  - 데이터를 등록할 INSERT SQL을 두 번 실행한다.

7.1.2 단일 테이블 전략 p.248

- 테이블을 하나만 사용해서 통합한다.
- 구분 컬럼(DTYPE)으로 어떤 자식 데이터가 저장되는지 구분한다.
- 자식 엔티티가 매핑한 컬럼은 모두 null 을 허용해야 한다.

- 장점
  - 조인이 필요 없으므로 일반적으로 조회 성능이 빠름
  - 조회 쿼리가 단순함
- 단점

    - 단일 테이블에 모든것을 저장하므로 테이블이 커질수있다.
    - 상황에 따라서 조회 성능이 오히려 느려질 수 있다.
- 특징
    - 구분 컬럼(@DiscriminatorColumn)을 필수로 사용해야 한다.
    - `@DIscriminatorValue` 를 지정하지 않으면 기본으로 엔티티 이름을 사용한다. ex) Movie, Album, Book

7.1.3 구현 클래스마다 테이블 전략

- **이 전략은 데이터베이스 설계자와 ORM 전문가 둘 다 추천하지 않는 전략이다.**
- 장점
    - 서브 타입을 명확하게 구분해서 처리할 때 효과적
    - not null 제약조건 사용 가능
- 단점
    - 여러 자식 테이블을 함께 조회할 때 성능이 느림(UNION SQL 필요)
    - 자식 테이블을 통합해서 쿼리하기 어려움

7.2 @MappedSuperclass p.251

- **테이블과 매핑할 필요가 없다.**
- **자식 클래스(엔티티)에게 공통으로 사용되는 매핑 정보만 제공한다**.
- 이 어노테이션을 지정한 클래스는 **엔티티가 아니므로** `em.find()` 나 `JPQL` 에서 사용할 수 없다. ⇒ 조회, 검색 불가
- 직접 생성해서 사용할 일은 거의 없으므로 **추상 클래스** 로 만드는 것을 권장한다.

- 테이블과 관계가 없고, 단순히 엔티티가 공통으로 사용하는 매핑정보를 모아주는 역할을 한다.

- 등록일자, 수정일자, 등록자, 수정자 같은 여러 엔티티에서 공통으로 사용하는 정보(속성)을 모을 때 사용한다(효과적으로 관리할 수 있다.)
- 참고 : `@Entity` 클래스는 `엔티티`나 `@MappedSuperclass` 로 지정한 클래스만 상속 가능하다.

7.3 복합 키와 식별 관계 매핑

7.3.1 식별관계 vs 비식별 관계

- 테이블베이스 테이블 사이에 관계는 외래 키가 기본 키에 포함되는지 여부에 따라 식별관계와 비식별 관계로 구분한다.
- 최근에는 비식별 관계를 주로 사용하고 꼭 필요한 곳에만 식별 관계를 사용하는 추세

식별 관계 p.254

- 부모 테이블의 기본 키를 내려받아서 자식 테이블의 기본 키 + 외래 키로 사용하는 관계

비식별 관계 p.255

- 부모 테이블의 기본 키를 받아서 자식 테이블의 외래 키로만 사용하는 관계

- 필수적 비식별 관계(Mandatory)
  - 외래 키에 NULL을 허용하지 않는다.
  - 연관관계를 필수적으로 맺어야 한다.
- 선택적 비식별 관계(Optional)
  - 외래 키에 NULL을 허용한다.
  - 연관관계를 맺을지 말지 선택할 수 있다.

7.3.2 복합 키 : 비식별 관계 매핑 p.256

- 식별자 필드가 2개 이상이면 별도의 식별자 클래스를 만들고 그곳에 `equals` 와 `hashCode` 를 구현해야 한다.
- JPA는 복합 키를 지원하기 위해 `@IdClass` (관계형 데이터베이스에 가까운 방법)와 `@EmbeddedId` (객체지향에 좀 더 가까운 방법)을 제공한다.

@IdClass

- **식별자 클래스의 속성명과 엔티티에서 사용하는 식별자의 속성명이 같아야 한다.**

  ex) 예제의 `Parent.id1` 과 `ParentId.id1`, `Parent.id2` 와 `ParentId.id2` 가 같다.
- `Serializable` 인터페이스를 구현해야 한다.
- `equals`, `hashCode` 를 구현해야 한다.
- 기본 생성자가 있어야 한다.
- 식별자 클래스는 public 이어야 한다.

@EmbeddedId

- `@Embeddeble` 어노테이션을 붙여주어야 한다.
- `Serializable` 인터페이스를 구현해야 한다.
- `equals`, `hashCode` 를 구현해야 한다.
- 기본 생성자가 있어야 한다.
- 식별자 클래스는 public 이어야 한다.

복합 키와 equals(), hashCode() p.261

@IdClass va @EmbeddedId

- @EmbeddedId 가 @IdClass 와 비교해서 더 객체지향적이고 중복도 없어서 좋아보이지만 특정 상황에 JPQL이 조금 길어질 수 있다.

7.3.4 비식별 관계로 구현 p.266

- 복합 키가 없으므로 복합 키 클래스를 만들지 않아도 된다.
- 식별 관계의 복합 키를 사용한 코드와 비교하면 매핑도 쉽고 코드도 단순하다.

7.3.5 일대일 식별 관계 p.268

- 일대일 식별 관계는 자식 테이블의 기본 키 값으로 부모 테이블의 기본 키 값만 사용한다.
- 그래서 부모 케이블의 기본 키가 복합 키가 아니면 자식 테이블의 기본 키는 복합 키로 구성하지 않아도 된다.

7.3.6 식별, 비식별 관계의 장단점 p.269

- 식별 관계보다는 비식별 관계를 선호한다.

정리

- 될 수 있으면 비식별 관계를 사용
- 기본 키는 Long 타입의 대리 키를 사용
- JPA 는 @GenerateValue 를 통해 간편하게 대리 키를 생성할 수 있다.
- 선택적 비식별 관계보다는 필수적 비식별 관계를 사용하는 것이 좋다.
    - 선택적 비식별 관계는 NULL을 허용하므로 조인할 때에 외부 조인을 사용해야 한다.
    - 필수적 비식별 관계는 NOT NULL 로 항상 관계가 있다는 것을 보장하므로 내부 조인만 사용해도 된다.

데이터베이스 테이블의 연관관계를 설계하는 방법

1. 조인 컬럼 사용(외래 키)

- 테이블 간에 관계는 주로 조인 컬럼이라 부르는 외래 키 컬럼을 사용해서 관리한다.

2. 조인 테이블 사용(테이블 사용)

- 조인 컬럼을 사용하는 대신에 조인 테이블을 사용해서 연관관계를 관리
- 조인 테이블이라는 별도의 테이블을 사용해서 연관관계를 관리한다.

⇒ 기본은 조인 컬럼을 사용하고 필요하다고 판단되면 조인 테이블을 사용

조인 테이블

- 객체와 테이블을 매핑할 때 조인 컬럼은 `@JoinColumn` 으로 매핑하고 조인 테이블은 `@JoinTable` 로 매핑한다.
- 조인 테이블은 주로 다대다 관계를 일대다, 다대일 관계로 풀어내기 위해 사용한다(그렇지만 일대일, 일대다, 다대일 관계에서도 사용한다.)

7.4.1 일대일 조인 테이블 p.273

- 일대일 관계를 만들려면 조인 테이블의 외래 키 컬럼 각각에 총 2개의 유니크 제약조건을 걸어야 한다.

7.4.2 일대다 조인 테이블 p.275

- 일대다 관계를 만들려면 조인 테이블의 컬럼 중 다(N)와 관련된 컬럼인 CHILD_ID 에 유니크 제약조건을 걸어야 한다

7.4.3 다대일 조인 테이블 p.276

- 일대다에서 방향만 반대

7.4.4 다대다 조인 테이블 p.277

- 다대다 관계를 만들려면 조인 테이블의 두 컬럼을 합해서 하나의 복합 유니크 제약조건을 걸어야 한다

7.5 엔티티 하나에 여러 테이블 매핑 p.278

- 잘 사용하지 X
- `@SecondaryTable` 을 사용하면 한 엔티티에 여러 테이블을 매핑할 수 있다.

```
@Entity
@Table(name = "BOARD") // BOARD 테이블과 매핑
@SecondaryTable(name = "BOARD_DETAIL", // BOARD_DETAIL 테이블을 추가로 매핑
    pkJoinColumns = @PrimaryKeyJoinColumn(name = "BOARD_DETAIL_ID"))
public class Board {
    //
    @Id
    @GeneratedValue
    @Column(name = "BOARD_ID")
    private String id;

    private String title; // 테이블을 지정하지 않으면 기본 케이블인 BOARD 에 매핑된다.

    @Column(table = "BOARD_DETAIL") // BOARD_DETAIL 테이블의 컬럼에 매핑
    private String content;
}
```