6장 다양한 연관관계 매핑

6.1 다대일

* 다대일 관계에서 외래 키는 항상 다쪽에 있다. 따라서 객체 양방향 관계에서 연관관계의 주인은 항상 다쪽이다.

6.1.1 다대일 단방향( 참조를 단방향밖에 못함)

* 회원과 팀 엔티티 예제에서 Member가 N이고, Team이 1의 관계를 맺으면

@Entity

public class Member{

@Id @GeneratedValue

@Column(name = “MEMBER\_ID”)

private Long id;

@ManyToOne

@JoinColumn(name =”TEAM\_ID”)

private Team team;

위와 같이 Member.team으로 팀 엔티티를 참조할 수 있지만 팀에는 회원을 참조하는 필드가 없으므로, 회원과 팀은 다대일 단방향 연관관계이다.

JoinColumn이라는 어노테이션을 활용해서 Member.team 필드로 회원 테이블의 TEAM\_ID 외래 키를 관리함.

6.1.2 다대일 양방향

* 양방향은 외래 키가 있는 쪽이 연관관계의 주인이다.
* 양방향 연관관계는 항상 서로를 참조해야 한다.

-> 연관관계의 주인이 아닌 쪽은 mappedBy로 참조한다.

* 항상 서로 참조하게 하기위해 편의 메소드를 작성하는 것이 좋다.

양쪽에 다 작성하게 되면 무한루프에 빠질 수 있으므로 조심해야 하며, 무한루프를 검사하는 로직이 필요.

Member class에 setTeam 메소드 추가

@Entity

public class Team{

@Id @GeneratedValue

@Column(name = “TEAM\_ID”)

private Long id;

@OneToMany(mappedBy=”team”)

private List<Member> members = new ArrayList<Member>();

public void addMember(Member member);

6.2 일대다

일대다 관계는 다대일의 반대로, 1인 측에서 엔티티를 하나 이상 참조할 수 있으므로 자바 컬렉션을 사용한다.

6.2.1 일대다 단방향

하나의 팀은 여러 회원을 참조할 수 있고, 회원이 팀을 참조하지 않으면 일대다 단방향이다.

일대다 단방향 연관관계는 특이하게 매핑한 테이블에 외래 키를 관리하는게 아니라

매핑당한 테이블에서 외래 키를 관리하는 특이한 관계가 맺어지며

Team에서 MEMBER 테이블의 TEAM\_ID를 JoinColumn으로 사용함.

-> 이는 여러 단점을 불러 일으킨다.

* 본인 테이블에 외래 키가 있으면 INSERT 한번으로 처리할 수 있지만 다른 테이블에 외래 키가 있으면 UPDATE SQL을 추가로 실행해야함.

->Member 엔티티는 Team 엔티티를 모르기 때문에, Member 엔티티를 저장할 때는 MEMBER테이블의 TEAM\_ID에 아무 값도 저장되지 않고, Team 엔티티를 저장할 때 team.members를 참조해서 TEAM\_ID를 업데이트함

이러한 추가적인 관리가 부담스럽고 성능도 저하되므로 일대다 단방향 매핑보다는 다대일 양방향 매핑을 사용하자!

6.2.2 일대다 양방향

일대다 양방향 매핑은 존재하지 않고, 다대일 양방향과 같은말이다. (그냥 연관관계의 주인이 왼쪽에 있고, 오른쪽에 있고 차이)

이 말은, OneToMany는 양방향 매핑에서 연관관계의 주인이 될 수 없기 때문.

일대다 양방향 매핑이 완전히 불가능한 것은 아니다.

일대다 단방향 매핑 반대편(Member)에 다대일 단방향 매핑을 읽기 전용으로 하나 추가해주면 됨.

Member class에

@JoinColumn(name = “TEAM\_ID”, insertable=false, updatable = false)

Private Team team; 을 추가한다.

이것은 일대다 양방향처럼 보이도록 하는 방법이기 때문에 일대다 단방향 매핑이 가지는 단점을 그대로 가지고 있음. 따라서 다대일 양방향 매핑을 사용하는 것이 좋음.

6.3 일대일

일대일 관계는 양쪽이 서로 하나의 관계만 가지므로 주 테이블과 대상 테이블로 나누어 생각한다.

어떤 테이블이 외래 키를 가질지 선택한다.

주 테이블에 외래 키

* 주 객체가 대상 객체를 참조하는 것처럼 주 테이블에 외래 키를 두고 대상 테이블을 참조한다.

객체지향 개발자 선호

* 장점 - 주 테이블이 외래 키를 가지고 있으므로 주 테이블만 확인해도 대상 테이블과 연관관계가 있는지 알수 있다.

대상 테이블에 외래 키

* 전통적인 데이터베이스 개발자 선호
* 장점 - 관계를 일대일에서 일대다로 변경할 때 테이블 구조를 그대로 유지할 수 있음.

6.3.1 주 테이블에 외래 키

* 단방향 - 회원과 사물함 관계
  + 주 테이블의 엔티티인 Member에 외래 키를 넣어 단방향으로 매핑함

@Entity

public class Member {

@Id @GeneratedValue

@Column(name = "MEMBER\_ID")

private Long id;

private String username;

@OneToOne

@JoinColumn(name = "LOCKER\_ID");

private Locker locker;

@Entity

public class Locker {

@Id @GeneratedValue

@Column(name = "LOCKER\_ID")

private Long id;

private String name;

* 양방향
  + 반대 방향을 추가해서 일대일 양방향 관계로 만듦.

기존 코드에서 Locker에 Member를 추가하고, mappedBy를 통해 연관관계의 주인이 아님을 명시함

@Entity

public class Locker {

@Id @GeneratedValue

@Column(name = "LOCKER\_ID")

private Long id;

private String name;

@OnetoOne(mappedBy=”locker”)

private Member member;

6.3.2 대상 테이블에 외래 키

* 단방향
  + 일대일 관계 중 대상 테이블에 외래 키가 있는 단방향 관계는 JPA에서 지원하지 않는다.
  + Member 엔티티에 Locker를 가지고 있고, 테이블은 Locker에 MEMBER\_ID를 외래 키로 가지고 있으니 매핑이 이루어지지 않는다.

방향을 바꾸던지, 양방향 관계로 만들고 Locker를 연관관계의 주인으로 설정해야함.

* 양방향
  + 일대일 매핑에서 대상 테이블에 외래 키를 두고 싶다면 양방향으로 매핑하고, 연관관계의 주인을 주 엔티티가 아닌 대상 엔티티로 만들어, 대상 테이블의 외래 키를 관리하도록 한다.

-> Member 엔티티에서 관리하던 외래 키를 Locker 엔티티에서 관리하도록 함.

연관관계의 주인이 연관관계 매핑하여 외래 키를 관리!

@Entity

public class Member {

@Id @GeneratedValue

@Column(name = "MEMBER\_ID")

private Long id;

private String username;

@OneToOne(mappedBy=”member”) 난 연관관계의 주인이 아님

private Locker locker;

@Entity

public class Locker {

@Id @GeneratedValue

@Column(name = "LOCKER\_ID")

private Long id;

private String name;

@JoinColumn(name = "MEMBER\_ID");

private Member member;

6.4 다대다

다대다 관계는 정규화된 테이블 2개로 표현할 수 없기 때문에 일대다, 다대일 관계로 풀어내는 연결 테이블을 사용한다.

회원들은 상품을 주문한다는 다대다 관계를 예로 들어보자.

다대다 관계에서는 각 테이블의 PK 두개를 참조해야 하기 때문에 외래키 하나로 해결하지 못한다. 따라서 각 테이블의 PK를 외래키로 가지는 연결 테이블을 하나 추가해서 일대다, 다대일 관계를 맺으면 다대다 관계를 풀어낼 수 있다.

그런데, 객체는 테이블과 다르게 객체 2개로 다대다 관계를 만들 수 있다.

회원 객체는 컬렉션을 사용해서 상품들을 참조하면 되고 반대로 상품들도 컬렉션을 사용해서 회원들을 참조하면 된다. -> ManyToMany 어노테이션 활용

6.4.1 다대다 단방향

@Entity

public class Member {

@Id @GeneratedValue

@Column(name = "MEMBER\_ID")

private Long id;

private String username;

@ManyToMany

@JoinTable(name=”MEMBER\_PRODUCT”, joinColumns=

@JoinColumn(name=”MEMBER\_ID”, inverseJoinColumn=

@JoinColumn(name=”PRODUCT\_ID”))

private List<Product>products = new ArrayList<Product>();

@Entity

public class Product{

@Id @Column(name=”PRODUCT\_ID”)

private String id;

private String name;

회원 엔티티에 @ManyToMany로 상품 엔티티를 매핑했고, JoinTable이란 어노테이션을 활용해서 연결 테이블을 바로 매핑했다. @JoinTable의 속성을 살펴보면

* @JoinTable.name : 연결 테이블을 지정 다대다 관계를 일대다, 다대일 관계로 풀어주는 연결 테이블로 설정했다.
* @JoinTable.joinColumns : 현재 방향인 회원과 매핑할 조인 컬럼 정보를 지정한다.
* @JoinTable.inverseJoinColumn : 반대 방향인 상품과 매핑할 조인 컬럼 정보를 지정한다.

@ManyToMany로 매핑한 덕분에 다대다 관계를 사용할 때는 연결 테이블을 신경쓰지 않아도 된다.

단방향이기 때문에 Product를 em에 등록하고, 상품을 구매한 Member에 add해주기만 하면 된다.

6.4.2 다대다 양방향

다대다 매핑이므로 역방향의 엔티티에도 @ManyToMany를 추가해주고, mappedBy를 통해 연관관계의 주인을 설정한다.

@Entity

public class Product{

@Id @Column(name=”PRODUCT\_ID”)

private String id;

@ManyeToMany(mappedBy=”products”)

private List<Member> members;

이렇게 되면 역방향으로도 탐색이 가능해진다.

이전 단방향에서 Member에만 add했던 부분을 Product에도 member를 add해줘야 하며 이부분은 앞에서 했듯이 연관관계 편의 메소드를 통해 오류의 위험을 없애자.

6.4.3 다대다: 매핑의 한계와 극복, 연결 엔티티 사용

실무에서는 연결 테이블에 단순히 주, 대상 테이블의 기본키만 담고 끝나지 않는다. 상품을 주문하면 주문 수량, 주문한 날짜 같은 컬럼이 더 필요하다.

이렇게 컬럼을 추가하면 더는 @ManyToMany를 사용할 수 없다.

주문 엔티티나 상품 엔티티에는 추가한 컬럼들을 매핑할 수 없기 때문이다.

따라서, 연결 테이블을 회원상품(MemberProduct)엔티티를 추가하여 일대다, 다대일관계로 풀어야 한다.

@Entity

@IdClass(MemberProductId.class)

public class MemberProduct{

@Id

@ManyToOne

@JoinColumn(name = "MEMBER\_ID")

private Member member;

@Id

@ManyToOne

@JoinColumn(name = "PRODUCT\_ID")

private Product product

private int orderAmount;

…

}

IdClass인 MemberProductId 클래스도 정의한다.

public class MemberProductId implements Serializable {

private String member;

private String product;

…

}

MemberProduct엔티티는 @Id 와 @JoinColumn을 동시에 사용해서 기본 키 + 외래 키를 한번에 매핑했고 @IdClass를 사용해서 복합 기본 키를 매핑했다.

복합 기본 키

JPA에서 복합 키를 사용하려면 별도의 식별자 클래스를 만들어야한다. 그리고 엔티티에 @IdClass를 사용해서 식별자 클래스를 지정하면 된다.

* 복합 키는 별도의 식별자 클래스로 만들어야 한다.
* Serializable을 구현해야 한다.
* equals와 hashCode 메소드를 구현해야 한다.
* 기본 생성자가 있어야 한다.
* 식별자 클래스는 public
* @IdClass를 사용하는 방법 외에 @EmbeddedId를 사용하는 방법도 있다.

복합 키를 사용하는 방법은 식별자 클래스를 만들어야하기 때문에 복잡하다.

6.4.4 다대다: 새로운 기본 키 사용

데이터베이스에서 자동으로 생성해주는 대리 키를 Long 값으로 사용하여 새로운 기본 키를 만들면, 복합키를 만들지 않아도 되므로 간단히 매핑이 가능해진다.

Member - Order 1: N 양방향, Order - Product - N : 1 단방향

@Entity

public class Order{

@Id @GeneratedValue

@Column(name = “ORDER\_ID”)

private Long id;

@Id

@ManyToOne

@JoinColumn(name = "MEMBER\_ID")

private Member member;

@Id

@ManyToOne

@JoinColumn(name = "PRODUCT\_ID")

private Product product

private int orderAmount;

…

}

Member와 Product는 수정할 필요가 없음.

6.4.5 다대다 연관관계 정리

다대다 관계를 일대다 다대일 관계로 풀어내기 위해 연결 테이블을 만들 때 식별자를 어떻게 구성할지 선택해야 한다.

* 식별 관계 : 받아온 식별자를 기본 키 + 외래 키로 사용한다.
* 비식별 관계 : 받아온 식별자는 외래 키로만 사용하고 새로운 식별자를 추가한다.

객체 관점에서는 비식별 관계를 사용하는 것이 식별자 클래스를 만들 필요가 없어 추천됨.