5장 연관관계 매핑 기초

엔티티들은 대부분 다른 엔티티와 연간관계가 있다.

그런데, 객체는 참조를 통해서 연관관계를 맺고, 테이블은 외래 키를 사용해서 관계를 맺는다. 이 둘은 완전히 다르며 ORM에서 가장 어려운 부분이 바로 객체 연관관계와 테이블 연관관계를 매핑하는 것이다.

이번 장에서는 객체의 참조와 테이블의 외래 키를 매핑하는 것이 목표다.

5.1 단방향 연관관계

연관관계 중에선 다대일(N:1) 단방향 관계를 가장 먼저 이해해야 한다.

우선 멤버와 팀이라는 다대일 관계가 있을 때 객체와 테이블의 연관관계가 어떻게 다른지 본다.

객체 연관관계 : 멤버 객체에 Member.team필드로 팀 객체와 연관관계를 맺는다.

멤버는 Member.team 필드를 통해서 팀을 알 수 있지만 Team객체는 Member를 알지 못하고 접근하지 못한다. 이를 단방향 관계라고 한다.

테이블 연관관계 : 멤버 테이블은 TEAM\_ID 외래 키로 팀 테이블과 연관관계를 맺는다. 이렇게 되면 외래 키로 JOIN을 할 수 있어 멤버, 팀 테이블 모든 방향으로 접근이 가능하다. 이를 양방향 관계라고 한다.

JOIN SQL

SELECT \* FROM MEMBER M JOIN TEAM T ON M.TEAM\_ID = T.ID

SELECT \* FROM TEAM T JOIN MEMBER M ON T.ID = M.TEAM\_ID

객체 연관관계와 테이블 연관관계의 가장 큰 차이

객체에서도 Team 객체에 Member를 필드로 가지면 접근이 가능하지만 이는 양방향이 아니라 서로 다른 단방향 관계가 2개인 것이 된다.

객체는 참조(Member.team)로 연관관계를 맺고, 참조를 통해 탐색한다. 이를 객체 그래프 탐색이라 한다.

테이블은 외래 키(TEAM\_ID)를 사용해서 연관관계를 탐색할 수 있는데 이것을 조인이라 한다.

5.1.3 객체 관계 매핑

객체 연관관계에서 사용하는 Member.team 필드와 테이블 연관관계에서 사용되는 외래 키 MEMBER.TEAM\_ID를 JPA를 사용해서 매핑 한다.

그 때 사용되는 어노테이션 @ManyToOne : 다대일 관계라는 매핑정보와

@JoinColumn : 외래 키를 매핑할 때 사용한다.

ex)

@ManyToOne

@JoinColumn(name = “TEAM\_ID”)

private Team team;

5.2 연관관계 사용

5.2.1 저장

Team과 Member 객체를 생성하고 JPA에 저장한다.

여기서 Member 객체에는 Team을 참조하도록 하여 저장시키면 어떻게 될까.

em.persist(team1);

member1.setTeam(team);

em.persist(member1);

JPA는 참조한 팀의 식별자를 외래 키로 사용해서 적절한 등록 쿼리를 생성한다.

5.2.2 조회

연관관계가 있는 엔티티를 조회하는 방법은 크게 2가지이다.

1. 객체 그래프 탐색
   1. member.getTeam()을 사용해서 member와 연관관계가 있는 team엔티티를 조회한다.
2. 객체지향 쿼리 사용JPQL
   1. String jpql = “select m from Member m join m.team t where “ + “t.name=:teamName”;

: 로 나타난 부분은 parameter로 설정할 수 있음.

em.createQuery(jpql, Member.class).setParameter(“teamName”, “팀1”).getResultList();

를 통해 팀1과 연관관계가 있는 member를 조회할 수 있다.

5.2.3 수정

em.persist(team2);

Member member = em.persist(Member.class, “member1”);

member.setTeam(team2);

JPA에 등록되어있는 영속 상태의 엔티티인 member1은 내용이 변경 되면 자동으로 수정 SQL이 쓰기 지연 목록에 저장되어 수정이 될 것이다.

5.2.4 연관관계 제거

member1.setTeam(null);

null을 넣으면 연관관계가 제거 되며 마찬가지로 TEAM\_ID를 null로 바꾸는 SQL문이 저장되고 관계가 제거된다.

5.3 양방향 연관관계

지금까지 객체는 Member에서 Team으로만 연관관계를 가지는 단뱡향 연관관계였고, Team에서 Member로 연관관계를 짓고 이를 양방향 연관관계로 만들어본다. Team은 Member와 일대다 관계이므로 List로 team.members로 관리한다.

테이블은 외래 키 하나로 양방향 조회가 가능하기 때문에 수정할 것이 없다.

5.3.1 양방향 연관관계 매핑

우리는 앞서 Member객체의 필드에서 연관관계 매핑을 위한 작업을 했었다.

class Member

@ManyToOne

@JoinColumn(name =”TEAM\_ID”)

private Team team;

매핑한 팀 엔티티

class Team

@OneToMany(mappedBy=”team”)

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

..

을 통해서 객체에서 양방향 매핑을 해주었다. mappedBy에는 매핑한 객체의 필드명을 적어준다.

5.3.2 일대다 컬렉션 조회

이번엔 Team 객체를 가지고 Member객체를 조회한다. Team과 Member는 일대다 관계이므로 컬렉션으로 조회한다.

List<Member> members = team.getMembers(); // 팀->회원 객체 그래프 탐색

5.4 연관관계의 주인

앞선 장에서 객체는 양방향 매핑이 불가능하다고 했었다. 그런데, 양 객체에 단방향 연관관계를 넣어주면서 양방향 연관관계인 것 처럼 구현을 했다.

이렇게 되면 테이블과 객체간의 차이점이 발생한다. 테이블은 외래 키 1개만 관리하면 되는데, 객체는 참조한 필드를 각 1개씩 총 2개를 관리한다.

이렇게 되면 테이블 입장에서는 어떤 필드를 외래 키로 사용할 지 문제가 생기는데

외래 키를 결정하기위해 연관관계의 주인을 정하는 것이다.

5.4.1 양방향 매핑의 규칙: 연관관계의 주인

1. 연관관계의 주인만이 연관관계와 매핑되고 외래키를 관리
2. 주인이 아닌 쪽은 읽기만 가능.
3. 주인은 mappedBy 속성을 사용하지 않음.
4. 주인이 아니면 mappedBy 속성을 사용해서 주인을 지정해야함

5.4.2 연관관계의 주인은 외래 키가 있는 곳

연관관계의 주인은 테이블에 외래 키가 있는 곳으로 정해야 한다.

여기서는 MEMBER 테이블이 외래 키를 가지고 있으므로 Member.team이 주인이 된다.

Team.members에는 mappedBy=”team” 속성을 사용해서 주인이 아님을 설정하고, 주인이 누구인지 설정한다.

5.5 양방향 연관관계 저장

연관관계의 주인은 Member 객체가 되었다.

member1과 member2를 team1과 연관관계를 맺고 JPA에 저장하면 MEMBER 테이블에 TEAM\_ID 외래 키에는 팀의 기본 키 값이 저장되어 있다.

그 후 team.getMembers().add(member1); 과 같은 설정을 추가해야 될 것 같지만

연관관계의 주인이 team이 아니면 아무런 일도 일어나지 않는다.

외래 키에 영향을 주지 않음. 저장할 때 무시된다.

5.6 양방향 연관관계의 주의점

연관관계의 주인이 아니라면 외래 키에 아무런 영향을 못 끼친다.

따라서, 데이터베이스에 외래 키 값이 정상적으로 저장되지 않는다면 주인이 아닌 곳에만 값을 입력한 것을 가장 먼저 의심해봐야 한다.

5.6.1 순수한 객체까지 고려한 양방향 연관관계

연관관계의 주인에만 값을 저장하고 주인이 아닌 쪽에는 값을 저장하지 않아도 될까?

객체의 관점에선 양쪽 방향에 모두 값을 입력해주는 것이 가장 안전하다.

위에서 봤던 team.getMembers().add(member1);와 같은 코드는 입력을 해도 데이터베이스에는 아무런 영향을 끼치지 않는다.

하지만 객체 관점에서는 넣어줘야 오류가 발생하지 않을 수 있다.

따라서 양쪽 모두 관계를 맺어주자.

5.6.2 연관관계 편의 메소드

양방향 연관관계는 결국 양쪽 다 신경 써서 관계를 설정해야 한다.

둘 중 하나라도 빼먹게 되면 양방향이 깨질 수 있기 때문에

member.setTeam(team1);에서 setTeam 메소드에 team.getMembers().add(); 코드를 추가하면 안전하게 사용할 수 있다.

5.6.3 연관관계 편의 메소드 작성 시 주의사항

위에서 setTeam메소드를 수정해서 양방향 연관관계를 맺을 수 있게 해줬다.

그런데 team을 수정해서 외래 키가 변경되면 어떻게 되는가

team1에서 team2로 변경하면 team2는 제대로 member가 추가되고, 외래 키가 수정되겠지만 team1에 member가 그대로 남아있기 때문에 수정을 할 경우에는 기존 team에서 member를 삭제해주어야 한다.