다양한 엔티티 관계

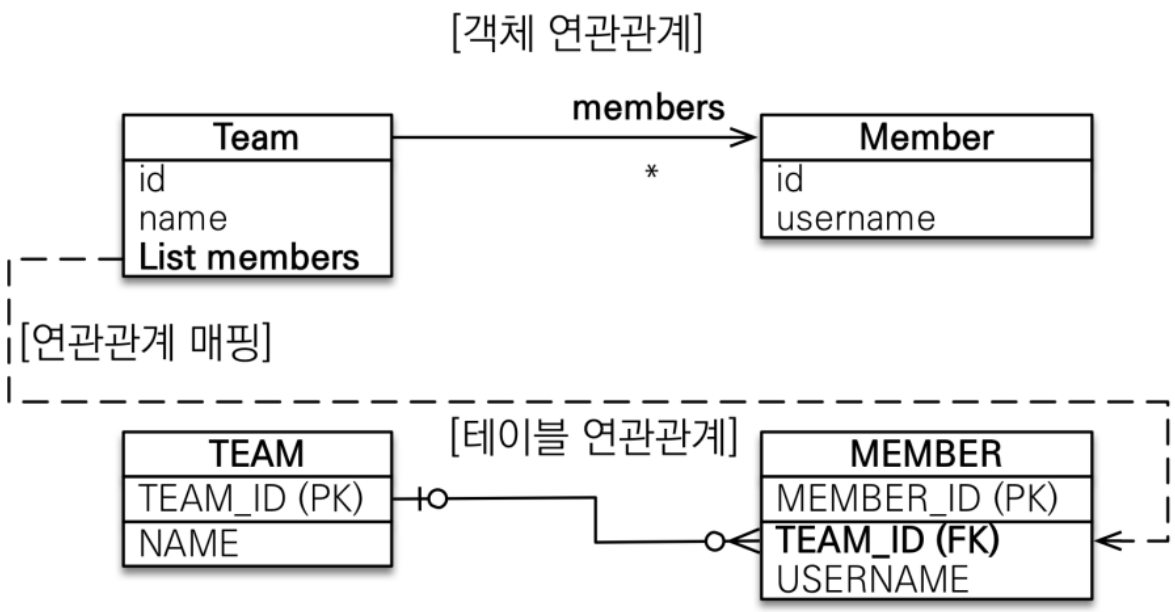
1. N:1 단방향 / 양방향

전 챕터 참고

2. 1:N

2-1. 1:N 단방향

팀은 팀원을 알지만 팀원은 소속팀을 몰라야 하는 경우



Team.java

```
@Entity
@Getter @Setter @ToString
public class Team {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "team_name")
    private String name;


    // 이 부분 유심히 볼 것!
    @OneToMany
    @JoinColumn("team_id")
    List<Member> members = new ArrayList<>();

}
```

Member.java

```
@Entity
@Getter @Setter
public class Member {
    @Id
    @GeneratedValue
    private Long id;

    private String name;

    // 멤버는 소속팀을 모른다.
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {

        EntityManagerFactory factory =
                Persistence.createEntityManagerFactory("myunit");

        EntityManager em = factory.createEntityManager();
        EntityTransaction tx = em.getTransaction();

        tx.begin();

        try {
            Member member1 = new Member();

            Team team1 = new Team();


            em.persist(member1);

            em.persist(team1);

            member1.setName("홍길동");

            team1.setName("회계부");

            team1.getMembers().add(member1);

            System.out.println("----------------------------");
            tx.commit();
            System.out.println("----------------------------");
        } catch (Exception e) {
            e.printStackTrace();
            tx.rollback();
        } finally {
            em.close();
        }
        factory.close();
```

```
    }
}
```

- 일대다 단방향은 일대다(1:N)에서 일(1)이 연관관계의 주인
- 테이블 일대다 관계는 항상 다(N) 쪽에 외래 키가 있음
- 객체와 테이블의 차이 때문에 반대편 테이블의 외래 키를 관리하 는 특이한 구조
- @JoinColumn을 꼭 사용해야 함. 그렇지 않으면 조인 테이블 방식을 사용함(중간에 테이블을 하나 추가함)
- 일대다 단방향 매핑의 단점
  - 엔티티가 관리하는 외래 키가 다른 테이블에 있음
  - 연관관계 관리를 위해 추가로 UPDATE SQL 실행
- 일대다 단방향 매핑보다는 다대일 양방향 매핑을 사용하자

2-2. 1:N 양방향

- 이런 매핑은 공식적으로 존재X
- @JoinColumn(insertable=false, updatable=false)
- 읽기 전용 필드를 사용해서 양방향 처럼 사용하는 방법
- 다대일 양방향을 사용하자

3. 1:1

- 일대일 관계는 그 반대도 일대일
- 주 테이블이나 대상 테이블 중에 외래 키 선택 가능
  - 주 테이블에 외래 키
  - 대상 테이블에 외래 키 (이 경우는 JPA 에서 단방향은 지원하지 않는다.)
- 데이터베이스의 외래키에 유니크 제약조건 추가
- N:1 단방향 + 외래키에 유니크만 추가하면 됨.

주 테이블에 외래 키

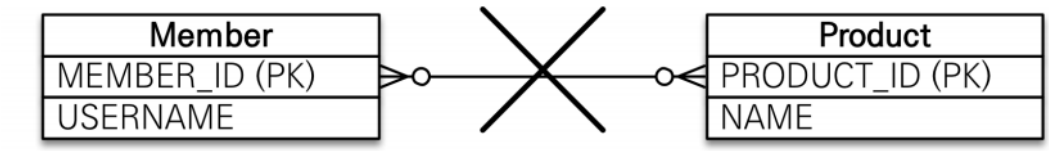
- 주 객체가 대상 객체의 참조를 가지는 것 처럼 주 테이블에 외래 키를 두고 대상 테이블을 찾음
- 객체지향 개발자 선호
- JPA 매핑 편리
- 장점: 주 테이블만 조회해도 대상 테이블에 데이터가 있는지 확인 가능
- 단점: 값이 없으면 외래 키에 null 허용

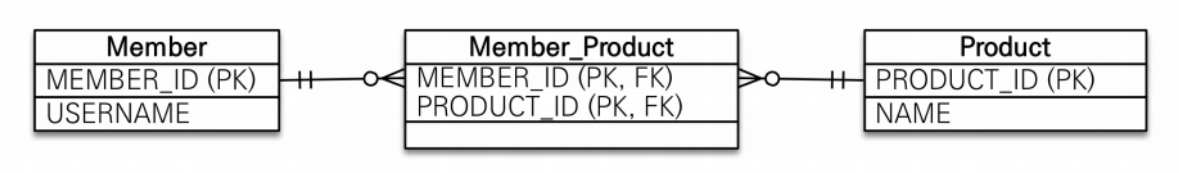
대상 테이블에 외래 키

- 대상 테이블에 외래 키가 존재
- 전통적인 데이터베이스 개발자 선호
- 장점: 주 테이블과 대상 테이블을 일대일에서 일대다 관계로 변경할 때 테이블 구조 유지
- 단점: 프록시 기능의 한계로 지연 로딩으로 설정해도 항상 즉시 로딩됨(프록시는 뒤에서 설명)

4. N:M

- 관계형 데이터베이스는 정규화된 테이블 2개로 다대다 관계를 표현할 수 없음
- 연결 테이블을 추가해서 일대다, 다대일 관계로 풀어내야함





- 컬렉션은 다대다가 가능하기 때문에 `@ManyToMany`를 사용할 수 있음
- `@JoinTable`로 연결테이블 이름 지정

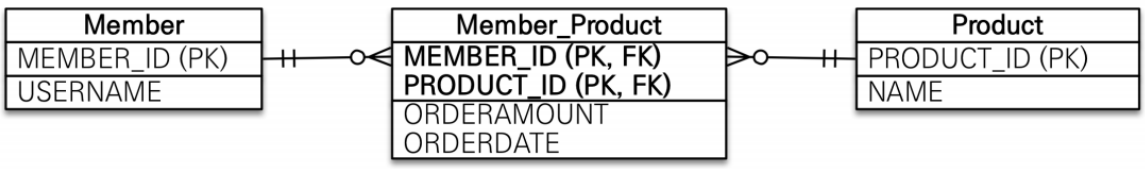
```java
@Entity
public class Product {

    @Id @GeneratedValue
    private Long id;

    @ManyToMany
    @JoinTable(name = "prod_member")
    private List<Member> members = new ArrayList<>();
}
```

결과

```
Hibernate:

    create table Member (
       id bigint not null,
        name varchar(255),
        team_id bigint,
        primary key (id)
    )
Hibernate:

    create table prod_member (
       Product_id bigint not null,
        members_id bigint not null
    )
Hibernate:

    create table Product (
       id bigint not null,
        primary key (id)
    )
```

다대다 매핑의 한계

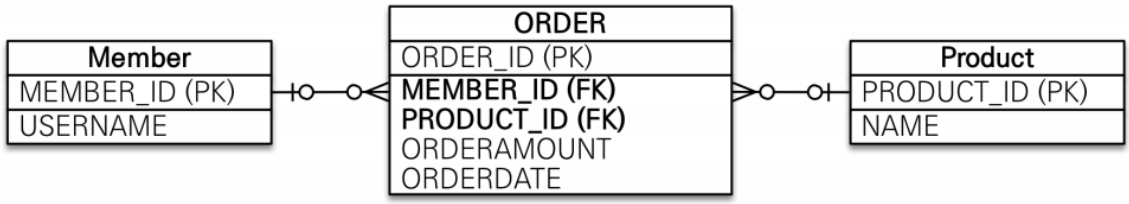
- 편리해 보이지만 실무에서 사용X
- 연결 테이블이 단순히 연결만 하고 끝나지 않음
- 주문시간, 수량 같은 데이터가 들어올 수 있음



해결 방법

- 연결 테이블용 엔티티 추가(연결 테이블을 엔티티로 승격)
- @ManyToMany -> @OneToMany, @ManyToOne


Member
id
memberProducts
username
1
*
MemberProduct
member
product
orderAmount
orderDate
*
1
Product
id
name


Member
MEMBER_ID (PK)
USERNAME
ORDER
ORDER_ID (PK)
MEMBER_ID (FK)
PRODUCT_ID (FK)
ORDERAMOUNT
ORDERDATE
Product
PRODUCT_ID (PK)
NAME