JPA 03 자바 표준 ORM 표준 JPA 프로그래밍

김영한
2022.08.03

다양한 연관관계 매핑

- 다중성, 단방향, 양방향, 연관 관계 주인

단방향, 양방향

테이블 : 외래 키 하나로 양쪽 조인 가능 → 방향 개념이 없다.

객체 : 참조용 필드가 있는쪽만 참조 가능 → 한쪽만 참조하면 단방향, 양쪽이면 양방향

- 사실 참조도 참조 입장으로 보면 단방향이 양쪽으로 있는 것.

  → 그러므로 참조가 두군데이기 때문에 외래키를 관리 할 **(** 주인 **)** 을 정해줘야 한다**.**

다대일 **[N : 1]**

다대일 단방향

외래 키 (주인) 은 다에 있어야 한다. 외래키 있는것을 기준으로 연관 참조를 걸어주면 된다.

다대일 양방향

테이블에서는 변경점이 없고 객체간의 관계만 정해진다.

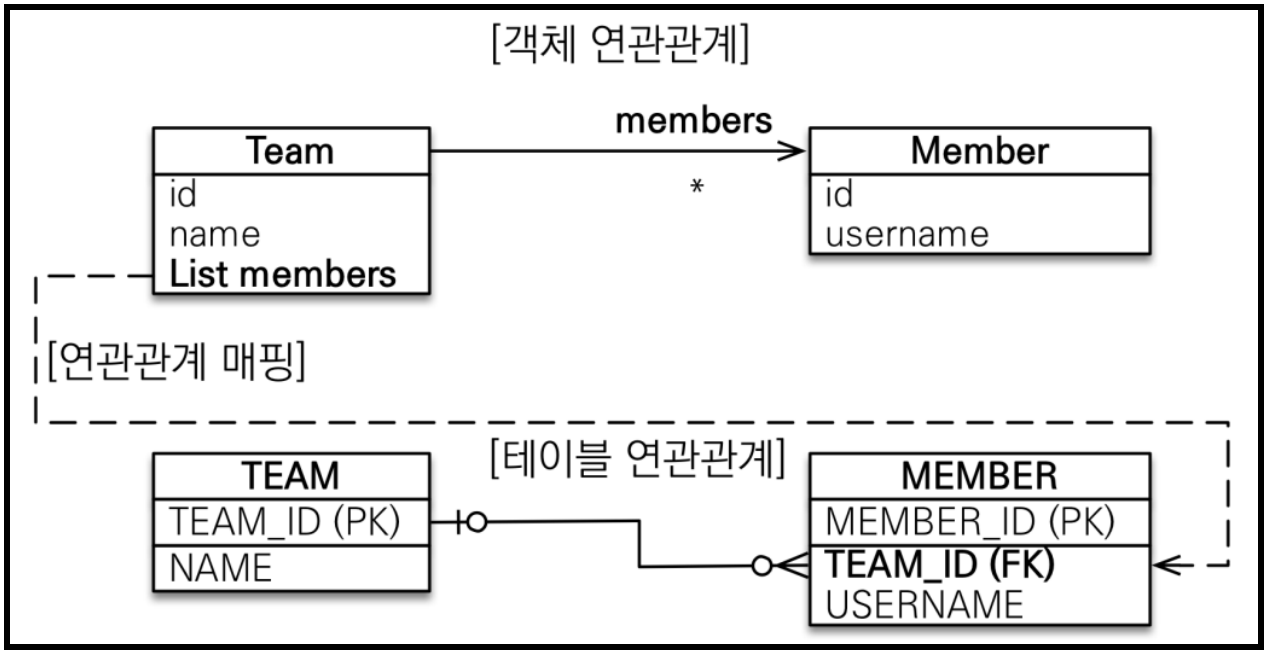
→ 양쪽을 서로 참조

일대다 **[1 : N]**

일대다 단방향

팀은 멤버를 알고 싶지만 멤버는 팀을 알고싶지않은 경우 외래 키는 무조껀 다 쪽으로 들어간다.

팀에만 List members 가 있을때 반대로 Member테이블에서 변경 되어야 한다. 그림으로 보게 된다면.



동작을 하게 된다 .

```
@OneToMany
@JoinColumn(name = "TEAM_ID")
private List<Member> members = new ArrayList<>();
```

외래 키가 Member에 존재하지만 Team 에서 저장하게 된다면 update 쿼리가 Member쪽으로 나가게 된다.

- 차라리 양쪽으로 다만들어주는게 낫다. (1이 연관관계의 주인이다.) 그렇지만 다쪽에 외래 키가 존재하고 반대편 테이블에 외래 키를 관리하는 이상한 관계가 된다.
- 꼭 @JoinColunm 으로 외래 키를 적어 줘야 한다. 안 적어주면 jointable로 테이블을 조인해 버린다.

단점 **:** 외래 키가 다른 테이블, 추가 update 쿼리가 나간다.

그러므로 다대일 양방향 매핑으로 사용하는 것이 좋다**.**

일대다 양방향

```
@ManyToOne
@JoinColumn(name = "TEAM_ID", insertable = false, updatable = false)
private Team team;
```

읽기 전용으로 매핑을 해주는 것.

→ 다대일 양방향 사용하자

일대일 **[1 : 1]**

- 외래키는 주 테이블, 대상 테이블 모두 가능하다.
- 다대일과 매우 유사 하고
- 양방향일시에는 mappedBy사용

일대일 단방향이 외래키가 대상 테이블에 존재하는 것은 지원이 안된다**.**

일대일 양방향일 때는 가능하다**.**

정리

주 테이블에 외래 키 → 객체적으로 좋다. (편하다)

대상 테이블에 외래 키 → 변경이 쉽다. 단점 : 지연 로딩을 걸어도 즉시 로딩된다. (프록시 객체를 만들기 위해서 locker를 확인해야한다.)

다대다 **[N : M]**

- 관계형 데이터 베이스는 정규화된 테이블 2개로 표현 불가능하기 때문에 연결 테이블을 추가해서 일대다 다대일 관계로 풀어낸다.
- 객체는 가능하다.....!? 그저 컬렉션 참조만 넣으면 된다.

조인테이블을 지정해 주고 사용한다.

한계점

단순 연결 뿐아니라 주문시간, 수량 같은 데이터가 같이 들어가야하는데 넣지 못한다.

엔티티를 추가해서 사용한다!

상속관계 매핑

- 관계형 데이터베이스는 상속관계 X
- 슈퍼타입 서브타입 관계라는 모델링 기법이 객체 상속과 유사

물리 모델 구체화

조인 전략

- 테이블을 생성한후에 필요할 때마다 조인해서 값을 가져온다.
- insert를 각각해주고 pk,fk 를 같이 가져가는 서브타입을 가지고 결정

단일 테이블 전략

- 하나의 테이블로 전부 관리하고 하나의 컬럼으로 구분자로 쓴다.

구현 클래스마다 테이블 전략

- 각각 공통 내용까지 생성하는 전략

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn
public class Item {
```

단일테이블 전략

@DiscriminationtorColumn이 자동으로 생성된다.

DTYPE은 운영상 항상 있는게 좋다. 단일테이블은 insert quary가 한번만 발생한다.

JOINED, SINGLE_TABLE,

TABLE_PER_CLASS : 사용하게 되면 Item 이라는 상위객체로 원하는 정보를 얻어올때 모든 타입의 테이블을 전부 검색하게 된다. (상당히 비효율)

조인 전략

장점

- 테이블 정규화 , 외래 키 참조 무결성 제약조건을 사용가능, 저장공간 효율화

단점

- 조회 시 조인을 많이 사용(성능 저하), 조회 쿼리가 복잡함, 저장시에 INSERT 2번 이상

단일테이블 전략

장점

- 조인이 필요없다(빠르다), 조회 커리가 단순함

단점

- 자식 엔티티가 매핑한 컬럼은 모두 null 허용을 해야한다.
- 테이블이 커질 수 있다 상황에 따라 조회가 느려질 수 있다.

구현 클래스마다 테이블 전략

장점 - 서브 타입을 명확하게 구분해서 처리할 때 효과적, not null 가능

단점 - 성능이 느려진다 (union sq) , 통합처리가 힘들다 (어렵다)

@MappedSuperclass

공통 매핑 정보가 필요할 때 사용한다.

객체 입장에서 속성만 공통으로 사용하고 싶을때 사용한다.

```
@MappedSuperclass                필요한 엔티티 속성이 필요할때 상속해서 사용하면된다.
public class BaseEntity {
    private String createBy;
    private LocalDateTime createdDate;
    private String lastModifiedBy;
    private LocalDateTime lastModifiedDate;
```

엔티티는 아니기 때문에 상위 클래스로 조회가 안되고

추상클래스로 사용이 권장된다.