# 여러가지 테이블 관계

엔티티 간의 관계는 여러 관계가 있음

## 1:1 `@OneToOne`

사람과 사람이 결혼 관계를 맺는다. (상대는 무조건 1)

사람과 사람이 연애 관계를 맺는다. (상대가 0혹은 1)


## N:1 `@ManyToOne`

사람과 사람의 소속 기업은 1:N 관계이다. (사람은 0 혹은 1개의 소속 기업을 갖는다.)

기업과 직원 사람은 N:1 관계이다. (기업은 0 혹은 여러개의 사람을 갖는다.)


## 1:N `@OneToMany`

출판사의 책은 0개 이상의 책을 갖는다.


## N:M `@ManyToMany`

장바구니에는 0개 이상의 상품을 담는다. (M)

상품은 여러 장바구니에도 담길 수 있다. (N)


# 연관관계의 방향

데이터베이스는 방향이라는 것이 없다. 외래키만 설정되어있으면 양방향이다.

즉, 방향이 아예 없든지 양방향이든지 무조건 둘 중 하나다.

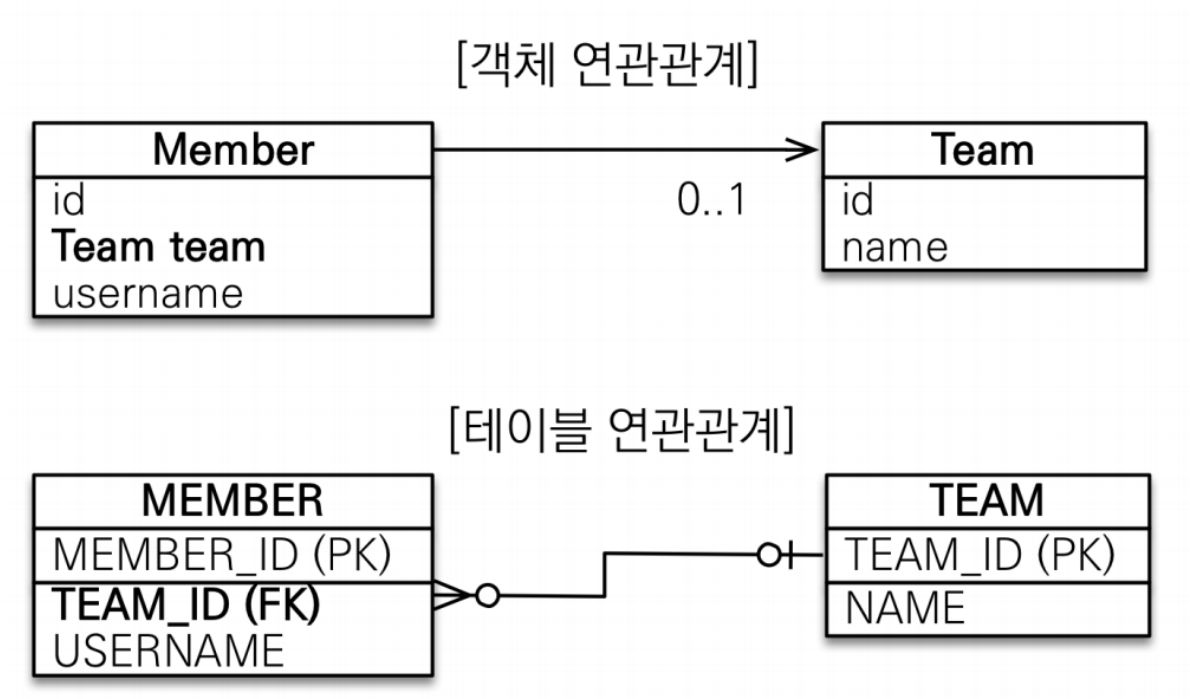
하지만 객체는 방향이 있다.  (참조를 하기 때문)

## 1. 단방향

Member 는 내가 누구의 Team 인지 알 수 있으나

Team은 어떠한 Member가 나의 소속인지 모르는 상태.

즉 외래키는 Member가 갖는다. (Team의 id를 갖는다.)



## Member.java

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

    private String name;
    private int age;

    @Column(name = "team_id")
    private Long teamId;

}
```

## Team.java

```
@Entity
public class Team {

    @Id
    @GeneratedValue
    private Long id;

}
```

## 결과

결과

```
Hibernate:

    drop table Member if exists
Hibernate:

    drop table Team if exists
Hibernate:

    drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:

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

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

외래키는 생성되지 않는다. 위 예시는 Team과 Member의 아무 연관관계가 없다.

JPA는 외래키 대신 Join(엔티티 포함) 개념을 사용한다. (객체 지향에서의 `has-a` 관계)

## Member.java 수정

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

    private String name;
    private int age;

    // 이 부분!!!!
    @ManyToOne  // Member와 Team은 다대일 관계다.
    @JoinColumn(name = "team_id")
    private Team team;

}
```

## 수정 후 결과

결과

```
Hibernate:

    drop table Member if exists
Hibernate:

    drop table Team if exists
Hibernate:

    drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:

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

    create table Team (
       id bigint not null,
        primary key (id)
    )
Hibernate:

    alter table Member
       add constraint FK5nt1mnqvskefwe0nj9yjm4eav
       foreign key (team_id)
       references Team
```

실제 DB에는 team_id로 fk가 자동생성되어 저장된다.

예

select * from team;

| ID | TEAM_NAME |
|---|---|
| 5 | 회계부 |
| 6 | 사업부 |

(2 행, 1 ms)

select * from member;

| ID | NAME | TEAM_ID |
|---|---|---|
| 1 | 홍길동 | 5 |
| 2 | 김길동 | 5 |
| 3 | 고길동 | 6 |
| 4 | 황길동 | 6 |

## 2. 양방향

서로가 서로의 존재를 알고 있는 것.

DBMS 는 외래키만 있으면 양쪽 모두에서 참조가 가능하므로 무조건 양방향이다.

```
SELECT *
FROM MEMBER M
JOIN TEAM T ON M.TEAM_ID = T.TEAM_ID
```

이렇게 해도 되고

```
SELECT *
FROM TEAM T
JOIN MEMBER M ON T.TEAM_ID = M.TEAM_ID
```

이렇게 해도 된다.

그런데 객체는..?

```
class Member {
    ...
    Long team_ID;
}
```

```
class Team {
    ...
    List<Member> members;
}
```

이렇게 둘 모두가 서로 서로 참조해야 양방향이 가능하다.

즉, 객체는 단방향 구조가 2개인 것이다.

- DBMS : 외래키가 있으면 양방향
- 객체 : 단방향 2개

## 여기서 발생하는 딜레마

DB와 클래스의 구조가 서로 패러다임간의 격차가 생긴다.

- 팀원이 새로 들어왔을 경우?
    1. DB라면
        - INSERT INTO member(team) VALUES ( 팀 );
    2. 객체라면
        - 새  Member 객체의 team 지정
        - 추가로 Team의 members 에도 add() 해야 함

- 팀원이 팀을 바꾸었을 경우?
    1. DB 라면
        - UPDATE member SET team=새 팀 WHERE id = ?;
    2. 객체라면
        - 팀원 Member의 team 수정
        - 원래 팀의 members 에 팀원 remove()
        - 새 팀의 members 에 팀원 add()

- 팀원이 삭제되었을 경우?
    1. DB 라면
    - DELETE FROM member WHERE id = ?
    2. 객체라면
        - 팀의 members 에 팀원 remove()
        - Team 에 팀원 remove()

등등...

## 해결방법 1

두 개의 단방향 구조를 모두 손대는 방법

- 단점 : 무결성을 보장할 수 없다.

## 해결방법2

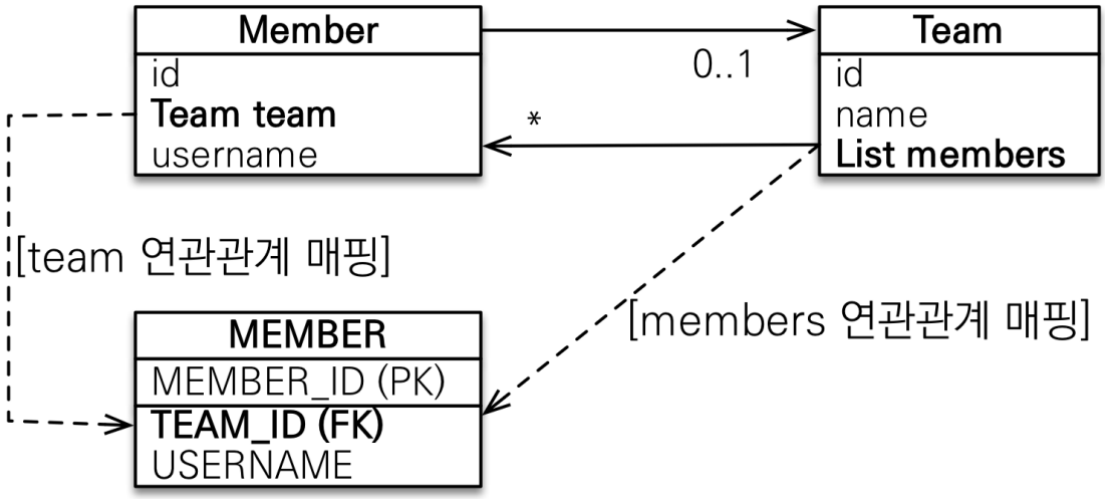
Team 의 members 업데이트 시 fk 를 업데이트, 그 후에 Member의 Team 업데이트

## 해결방법3

Member의 team 업데이트 시 fk 를 업데이트, 그 후에 Team 의 members 업데이트

## 결론 : 방법 2, 방법 3 중 하나를 선택해야 한다.

**둘 중 하나로 외래 키를 관리해야 한다.**




## 바람직한 양방향 구현

• FK를 가진 쪽(외래키를 참조하는 쪽)을 **연관관계의 주인**으로 지정

• 연관관계의 **주인만이 외래 키를 관리**(등록, 수정)

• 주인이 아닌쪽은 **읽기만 가능**

• 주인은 `mappedBy` 속성 사용 하지 않음

• 주인이 아니면 `mappedBy` 속성으로 주인 지정

## Member.java

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

    private String name;


    @ManyToOne
    @JoinColumn(name = "team_id")
    private Team team;

}
```

## Team.java

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

    @Id
    @GeneratedValue
    private Long id;

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

    @OneToMany(mappedBy = "team")  // Member.team 으로 mapping 당하겠다!
    List<Member> members = new ArrayList<>();

    @Override
    public String toString() {
        return "Team{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", Members='" + members.stream().map(s ->
s.getName()).collect(Collectors.joining(",")) + '\'' +
            '}';
    }
}
```

이렇게 되면

`aTeam.getMembers.add(newMember())` 해도 팀원 추가는 안됨. (List로 새 멤버 추가해도 DB에는 적용 안됨)

`newMember.setTeam(aTeam)` 이렇게 해야 aTeam의 members에 새 팀원 추가 됨.

## Main.java

```
package com.megait.ch02;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Main2 {
    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();
            Member member2 = new Member();

            Member member3 = new Member();
            Member member4 = new Member();

            Team team1 = new Team();
            Team team2 = new Team();

            em.persist(member1);
            em.persist(member2);
            em.persist(member3);
            em.persist(member4);
            em.persist(team1);
            em.persist(team2);

            member1.setName("홍길동");
            member2.setName("김길동");
            member3.setName("고길동");
            member4.setName("황길동");

            team1.setName("회계부");
            team2.setName("사업부");

            member1.setTeam(team1);
            member2.setTeam(team1);

            member3.setTeam(team2);
            member4.setTeam(team2);

            Team team = em.find(Team.class, 5L);
            em.persist(team);

            Member newMember = new Member();
            newMember.setName("피카츄");
            em.persist(newMember);
```

```
            // ㅇㅇ
            newMember.setTeam(team);

            // ㄴㄴ
            // team.getMembers().add(newMember);


            em.flush();
            em.clear();

            // 결과 확인
            Team findTeam = em.find(Team.class, 5L);
            System.out.println(findTeam);


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

## 에러 안나게 개선하기

### 주의점

1. 올바른 객제지향 방식을 고려할 것.

   단방향 2개니 양측 모두 저장해줄 것

   HOW : setTeam() 메서드 수정

#### Member.java

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

    private String name;


    @ManyToOne
    @JoinColumn(name = "team_id")
    private Team team;
```

```
    public void setTeam(Team team){
        this.team = team;
        if(!this.team.members.contains(this))
            this.team.members.add(this); // 이 부분!!
    }
}
```

2. 서로 서로 참조하는 것이기 때문에 무한 참조 조심할 것.

   예) toString()

### Team.java

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

    @Id
    @GeneratedValue
    private Long id;

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

    @OneToMany(mappedBy = "team")
    List<Member> members = new ArrayList<>();

    @Override
    public String toString() {
        return "Team{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", Members='" + members.stream().map(s ->
s.getName()).collect(Collectors.joining(",")) + '\'' + // 이 부분!!! 이 부분에
그냥 members 하면 안됨.
                '}';
    }
}
```