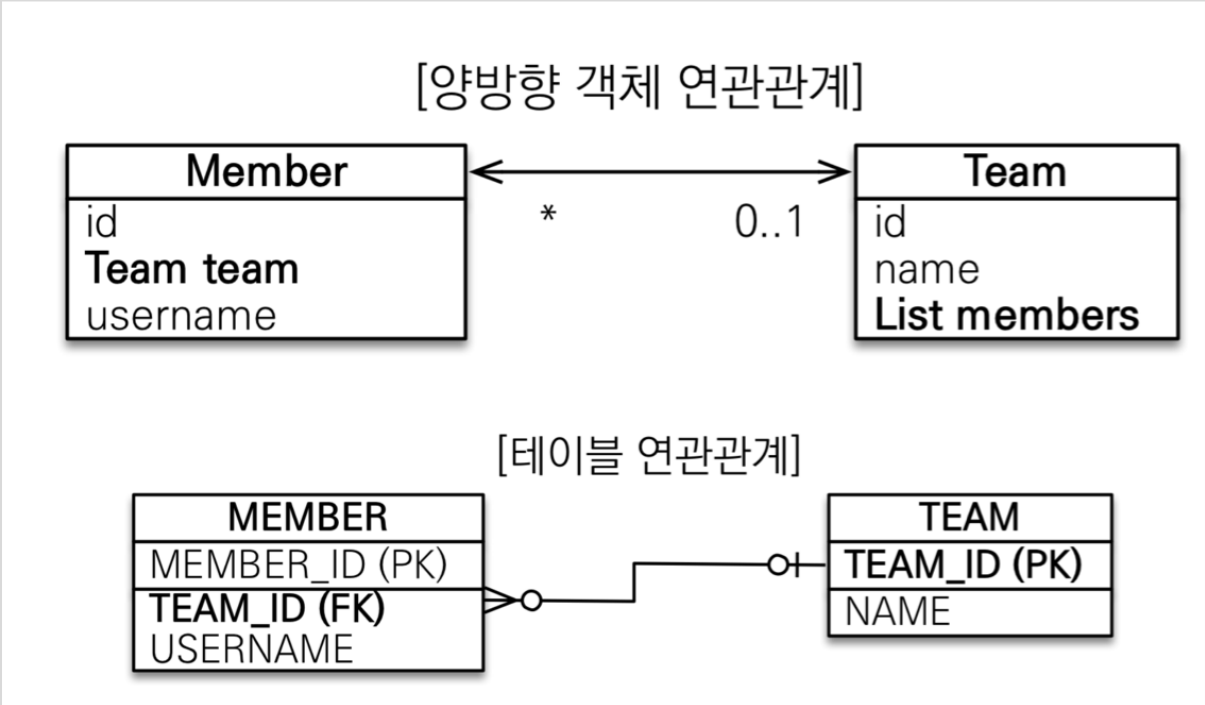
6-1. 양방향 연관관계와 연관관계의 주인 1 - 기본

어려운 이유?

객체와 테이블의 패러다임이 다름 - 객체 : 참조 / 테이블 : 외래키 로 연관관계 설정

양방향 매핑



양방향 연관관계 : Member, Team이 서로 참조해서 넘나들 수 있다.

- 양방향으로 참조하게 할 때도 테이블은 전혀 변환이 없다.
  ➜ 그 이유? 테이블을 생각해보면 team_id라는 외래키로 TEAM에 조인하면 된다.
  멤버에서 내가 속한 팀을 알고싶으면 외래키로 조인하면됨.
  반대로 팀에서 속한 멤버들을 알고싶으면 외래키로 조인하면됨.
  => **테이블의 연관관계에서는 외래키 하나만으로 관계 세팅이 된다. 즉 방향이 없다**
- 하지만 객체는 다르다. 멤버는 팀을 가졌기 때문에 팀을 참조할 수 있었으나 팀은 멤버를 가지지 못했기 때문에 참조하지 못했다. (`team.getMember()` 하지 못했다)
  ➜ 팀에다가 members라는 리스트를 넣어주어야 양쪽으로 참조가 가능하다.
  ➜ 이는 외래키 하나만 넣어주면 양방향 조인이 가능한 테이블과는 정말 다른 속성을 지닌다. (객체는 양쪽에 모두 세팅. 그러나 테이블을 한쪽에만 외래키 세팅.)

```
package hellojpa;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
```

```
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import java.util.List;

public class JpaMain {
  public static void main(String[] args) {
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("hello");

    EntityManager em = emf.createEntityManager();

    EntityTransaction tx = em.getTransaction();
    tx.begin();
    try {

      Team team = new Team();
      team.setName("TeamA");
      em.persist(team); // 영속상태가 되면 무조건 pk값이 세팅되고 영속상태가 됨.

      Member member = new Member();
      member.setName("member1");
      member.setTeam(team);
      em.persist(member);

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

      Member findMember = em.find(Member.class, member.getId());

      Team findTeam = findMember.getTeam();
      List<Member> members = findMember.getTeam().getMembers();

      for (Member member1 : members) {
        System.out.println("member1 = " + member1.getName());
      }

      tx.commit();
    } catch (Exception e) {
      tx.rollback();
    } finally {
```

```
        em.close();
    }

    emf.close();

  }

}
```

```
package hellojpa;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Team {

  @Id @GeneratedValue
  private Long id;
  private String name;
  @OneToMany(mappedBy = "team") // 팀의 입장에서는 일대다이기 때문에 OneToMany. 관계를 맺는 반대편의 외래키를 표시함.
  private List<Member> members = new ArrayList<Member>();

  public List<Member> getMembers() {
    return members;
  }

  public void setMembers(List<Member> members) {
    this.members = members;
  }

  public Long getId() {
```

```
    return id;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }


}
```

```
Hibernate:
    select
        member0_.id as id1_0_0_,
        member0_.USERNAME as USERNAME2_0_0_,
        member0_.TEAM_ID as TEAM_ID3_0_0_,
        team1_.id as id1_1_1_,
        team1_.name as name2_1_1_
    from
        Member member0_
    left outer join
        Team team1_
            on member0_.TEAM_ID=team1_.id
    where
        member0_.id=?
Hibernate:
    select
        members0_.TEAM_ID as TEAM_ID3_0_0_,
        members0_.id as id1_0_0_,
        members0_.id as id1_0_1_,
        members0_.USERNAME as USERNAME2_0_1_,
        members0_.TEAM_ID as TEAM_ID3_0_1_
    from
        Member members0_
    where
        members0_.TEAM_ID=?
member1 = member1
```

연관관계의 주인과 mappedBy

객체가 연관관계를 맺는 방식과 테이블이 연관관계를 맺는 차이??

- 객체 연관관계

- 회원 ➡ 팀 : 단방향
- 팀 ➡ 회원 : 단방향

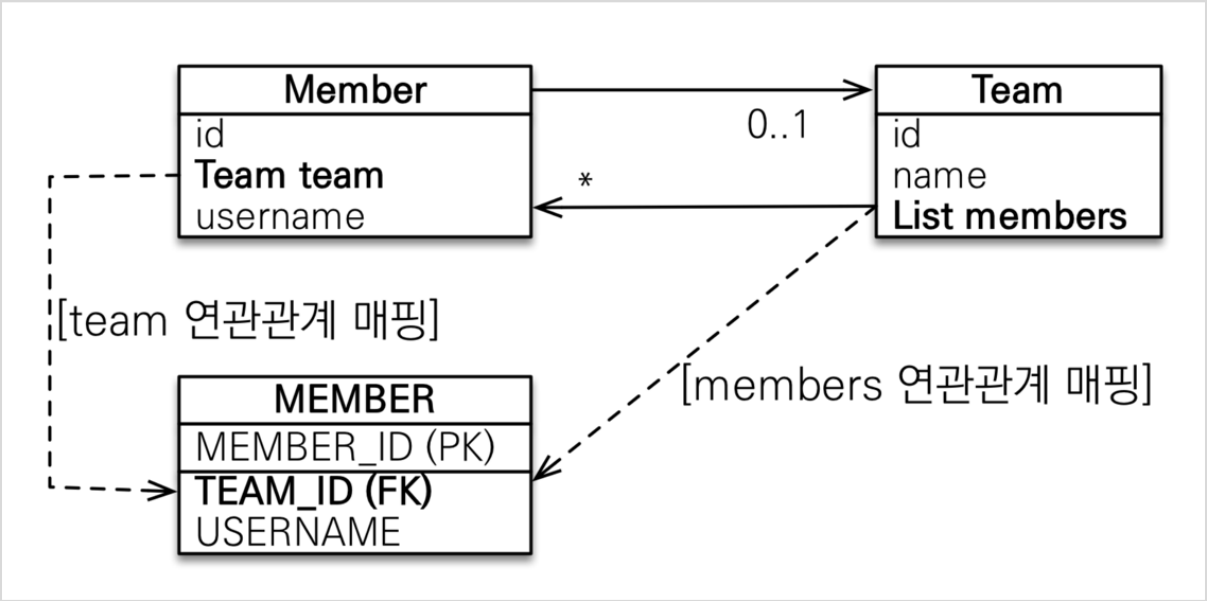
=> 양방향! 단방향x2

- 테이블 연관관계
  - 회원 <-> 팀 : 양방향
    - 회원 ➡ 팀 : 외래키로 조인하면 회원이 어디 팀 소속인지 알수있음
    - 팀 ➡ 회원 : pk값으로 조인하면 팀에 회원이 누가있는지 알수있음

=> 방향이 없음. 하나만 있으면 양쪽으로 왔다갔다 가능!

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

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

딜레마...



객체를 두 방향임. 참조가 두개라는거. 팀 ➡ 멤버, 멤버 ➡ 팀

둘 중에 뭘로 매핑해야하나?

멤버스의 팀값을 바꿨을 때 외래키가 업데이트 되어야 하나

팀의 멤버스 값을 바꿨을 때 외래키가 업데이트 되어야 하나

즉, 내가 팀을 바꾸고 싶거나 팀에서 멤버를 교체하고 싶을 때 내 팀을 바꿀것이냐 팀의 멤버스를 바꿀것이

냐

테이블 연관관계에서는 외래키 하나로도 양방향 소통이 가능했다.
그래서 위 굵은글씨의 경우에도 그냥 상관없이 외래키만 업데이트하면 되었다.
단순하게 생각해보자.
테이블 관점에서는 멤버의 팀 id만 바꾸면 멤버가 어느 팀에 있는지, 그리고 그 팀에는 어떤 멤버들이 속해 있는지를 확인하기가 정말 편했다.
하지만 객체 관점에서는 멤버의 팀 아이디를 바꾸면 팀에서도 멤버스 배열에서의 참조 중 하나의 값을 바꾸어주어야

=> 둘 중 하나로 외래 키를 관리해야한다!

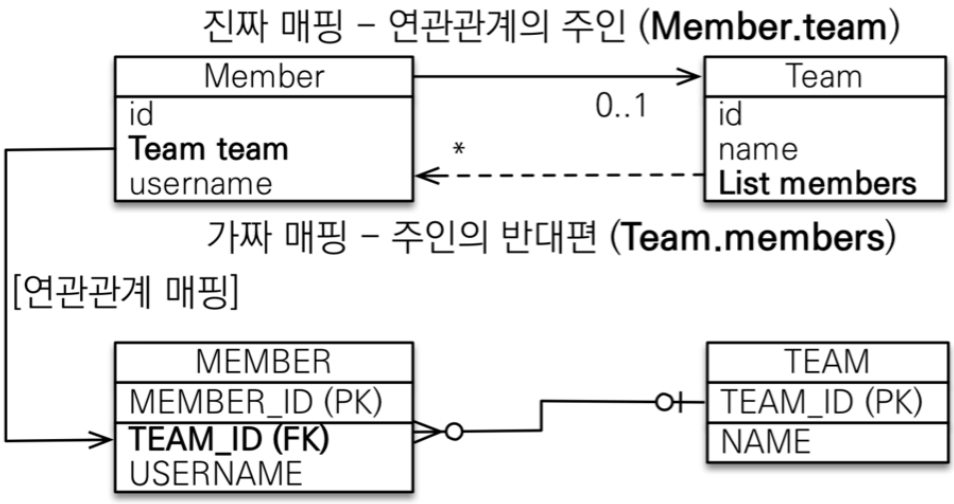
연관관계의 주인
둘 중 하나로 외래 키를 관리해아하는데, 이 둘 중 하나를 연관관계의 주인으로 정하자!!!
양방향 매핑 규칙은 강의록 참조.
mappedBy : 주인에 의해 내가 매핑 당했다… 이건 그래서 노예의 것.

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

=> members 리스트에 어떤걸 넣어도 바뀌지않는다!! 값을 변경할 때에는 무조건 Member에서 수정해야함. 여기서는 조회만 가능!!!!!

누구를 주인으로 할 것인가?

- 외래 키가 있는 있는 곳을 주인으로 정해라
- 여기서는 **Member.team**이 연관관계의 주인



이유

- 팀의 멤버스의 값을 바꿨다. 근데 다른 테이블(멤버)의 업데이트 쿼리가 나간다.
- 성능이슈도 좀 있음!
- 외래키에 있는 곳에다가 해버리면 헷갈릴게 없다!
  - 멤버는 인서트할때 외래키를 챙길 수 있어서 인서트가 한방에 되는데
  - 팀은 인서트할때 외래키를 못챙겨서 팀도 인서트, 멤버도 업데이트 해야함.
- 외래키가 있는 곳이 무조건 n. 없는 곳이 무조건 1. 일대다에서는 다 쪽이 무조건 주인!

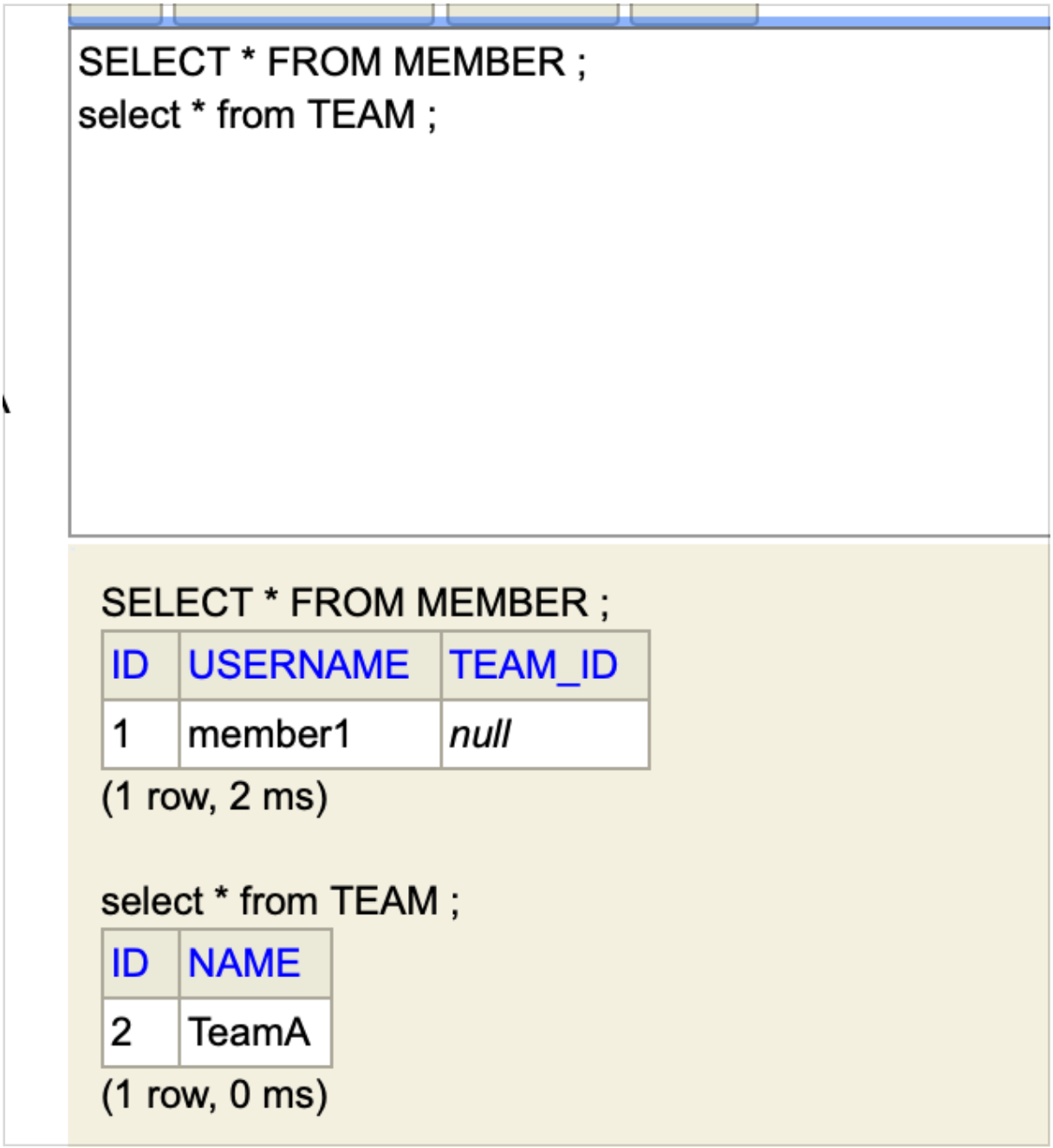
양방향 매핑시 가장 많이 하는 실수

```
package hellojpa;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import java.util.List;

public class JpaMain {
  public static void main(String[] args) {
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("hello");
```

```
    EntityManager em = emf.createEntityManager();

    EntityTransaction tx = em.getTransaction();
    tx.begin();
    try {

      Member member = new Member();
      member.setName("member1");
      em.persist(member);

      Team team = new Team();
      team.setName("TeamA");
      team.getMembers().add(member);
      em.persist(team);

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

      tx.commit();
    } catch (Exception e) {
      tx.rollback();
    } finally {
        em.close();
    }

    emf.close();

  }

}
```

```
Hibernate:
    /* insert hellojpa.Member
        */ insert
        into
            Member
            (USERNAME, TEAM_ID, id)
        values
            (?, ?, ?)
Hibernate:
    /* insert hellojpa.Team
        */ insert
        into
            Team
            (name, id)
        values
            (?, ?)
```

분명히 인서트 쿼리가 두방 나갔는데

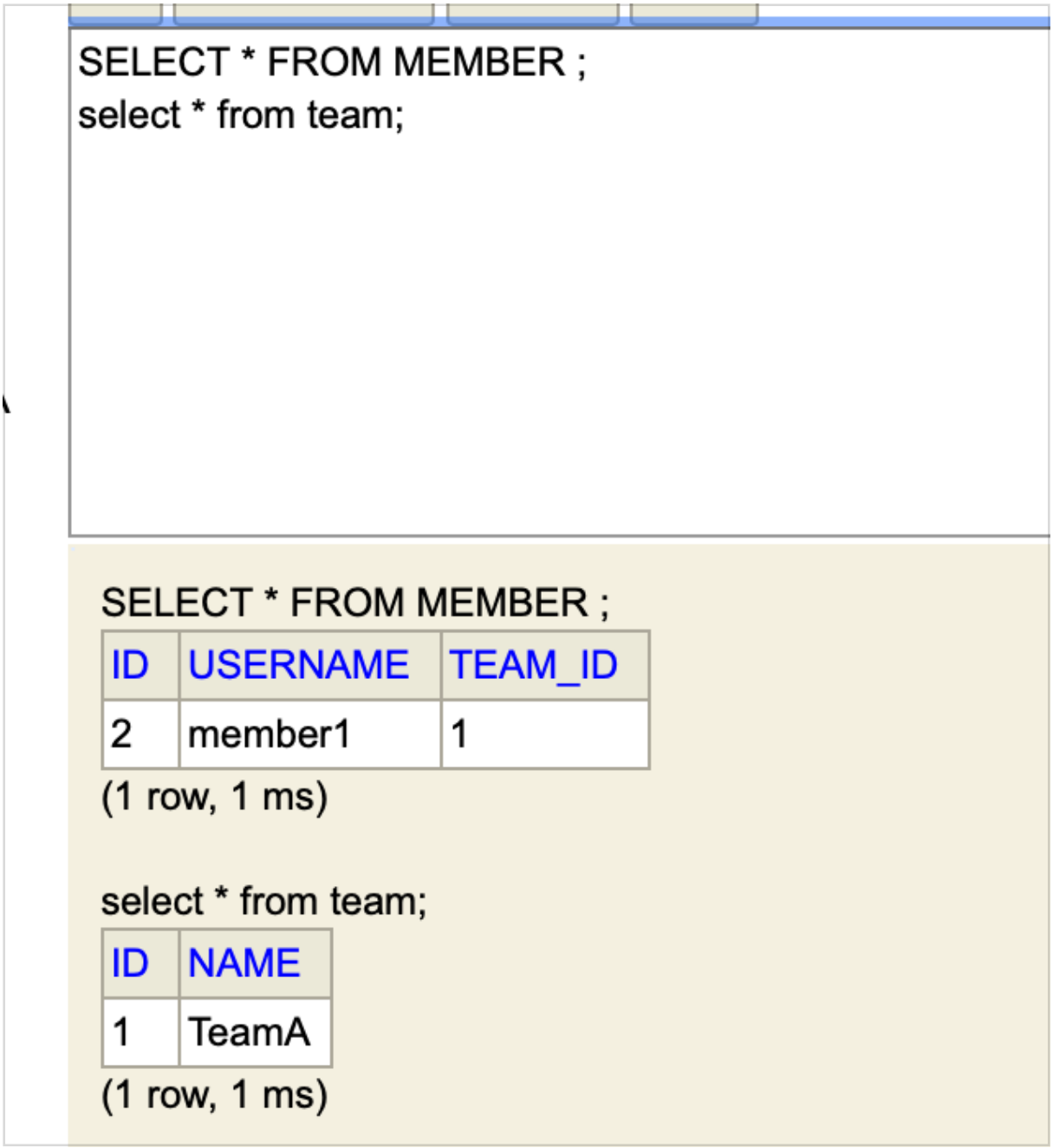


디비에는 멤버의 팀 id가 null !!!

=> 연관관계의 주인이 멤버의 team이다. 이 경우에는 지금 가짜 매핑을 해준 것.

```
package hellojpa;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import java.util.List;

public class JpaMain {
```

```
  public static void main(String[] args) {
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("hello");

    EntityManager em = emf.createEntityManager();

    EntityTransaction tx = em.getTransaction();
    tx.begin();
    try {

      Team team = new Team();
      team.setName("TeamA");
//      team.getMembers().add(member);
      em.persist(team);

      Member member = new Member();
      member.setName("member1");
      member.setTeam(team);
      em.persist(member);



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

      tx.commit();
    } catch (Exception e) {
      tx.rollback();
    } finally {
        em.close();
    }

    emf.close();

  }

}
```



진짜 매핑을 해 주어서 디비에 잘 들어갔다.
주인인 쪽에 세팅을 해주니까 잘 들어가네..

#JPA