6장 다양한 연관관계 매핑

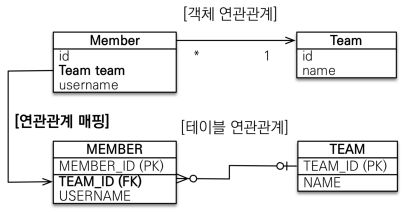
▼ 엔티티의 연관관계를 매핑할 때 고려해야 하는 것 204p

- **다중성**
  - 다대일(@ManyToOne)
  - 일대다(@OneToMany)
  - 일대일(@OneToOne)
  - 다대다(@ManyToMany) - 실무에서 거의 사용 x
- **단방향, 양방향**
- **연관관계의 주인**

6.1 다대일 206p

- 데이터베이스 테이블의 일(1), 다(N) 관계에서 외래 키는 항상 다 쪽에 있다.
- 따라서 객체 양방향 관계에서 연관관계의 주인은 항상 다(N)쪽이다.

6.1.1 다대일 단방향[N:1] 206p




```java
@Entity
public class Member {
    //
    @Id
    @GeneratedValue
    @Column(name = "MEMBER_ID")
    private Long id;

    private String username;

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

    // Getter / Setter
}
```

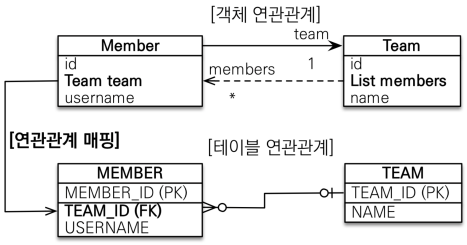
```java
@Entity
public class Team {
    //
    @Id
    @GeneratedValue
    @Column(name = "TEAM_ID"
    private Long id;

    private String name;

    // Getter / Setter
}
```

- 회원은 `Member.team` 으로 팀 엔티티를 참조할 수 있지만 반대로 팀에는 회원을 참조하는 필드가 없다.
- 따라서 회원과 팀은 다대일 단방향 연관관계다.

- `@JoinColumn(name = "TEAM_ID")` 를 사용해서 `Member.team` 필드를 `TEAM_ID` 외래 키와 매핑했다.

- 따라서 `Member.team` 필드로 회원 테이블의 `TEAM_ID` 외래 키를 관리한다.

6.1.2 다대일 양방향 [N:1, 1:N] 207p



- 다대일 양방향의 객체 연관관계에서 실선이 연관관계의 주인(`Member.team`)이고 점선(`Team.members`)은 연관관계의 주인이 아니다.

```
@Entity
public class member {
    //
    @Id
    @GeneratedValue
    @Column(name = "MEMBER_ID")
    private Long id;

    private String username;

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

    public void setTeam(Team team) {
        this.team = team;

        // 무한루프에 빠지지 않도록 체크
        if (!team.getMembers().contains(this)) {
            team.getMembers().add(this);
        }
    }
}
```

```
@Entity
public class Team {
    //
    @Id
    @GeneratedValue
    @Column(name = "TEAM_ID")
    private Long id;

    private String name;

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

    public void addMember(Member member) {
        this.members.add(member);

        // 무한루프에 빠지지 않도록 체크
        if (member.getTeam() != this) {
            member.setTeam() = this;
        }
    }
}
```

- **양방향은 외래키가 있는 쪽이 연관관계의 주인이다.**
- **양방향 연관관계는 항상 서로를 참조해야 한다.**

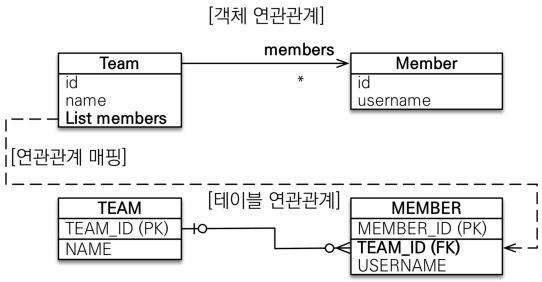
6.2 일대다 209p

- 일대다 관계는 엔티티를 하나 이상 참조할 수 있으므로 자바 컬렉션인 Collection, List, Set, Map 중에 하나를 사용해야 한다.

6.2.1 일대다 단방향 [1:N] p.209

ex)

- 하나의 팀은 여러 회원을 참조할 수 있다.
- 팀은 회원들은 참조하지만 반대로 회원은 팀을 참조하지 않으면 둘의 관계는 단방향이다.



- 일대다 관계에서 외래 키는 항상 다쪽 테이블에 있다.
- 하지만 다 쪽인 Member 엔티티에는 외래 키를 매핑할 수 있는 참조 필드가 없다.
- 대신 반대쪽인 Team 엔티티에만 참조 필드인 `members` 가 있다.

  ⇒ 반대편 테이블의 외래 키를 관리하는 특이한 모습
- 일대다 단방향은 일대다(1:N)에서 **일(1)이 연관관계의 주인**이다.

```
@Entity
public class Team {
    //
    @Id
    @GeneratedValue
    @Column(name = "TEAM_ID")
    private Long id;

    private String name;

    @OneToMany(mappedBy = "team")
    @JoinColumn(name = "TEAM_ID") // MEMBER 테이블의 TEAM_ID(FK)
    private List<Member> members = new ArrayList<Member>();

    // Getter / Setter
}
```

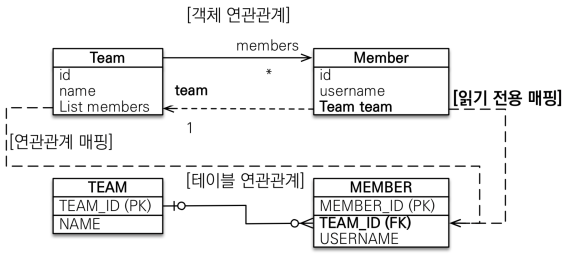
```
@Entity
public class Member {
    //
    @Id
    @GeneratedValue
    @Column(name = "MEMBER_ID")
    private Long id;

    private String username;

    // Getter / Setter
}
```

- 일대다 단방향 관계를 매핑할 때는 `@JoinColumn` 을 명시해야 한다.
  - 그렇지 않으면 JPA는 연결 테이블을 중간에 두고 연관관계를 관리하는 조인 테이블 전략을 기본으로 사용해서 매핑한다(중간에 테이블을 하나 추가함). ⇒ 7.4절 참고
- 일대다 단방향 매핑보다는 다대일 양방향 매핑을 사용하자

6.2.2 일대다 양방향 [1:N, N:1] p.212

- 일대다 양방향 매핑을 공식적으로 존재하지 않는다.
- 양방향 매핑에서 @OneToMany 는 연관관계의 주인이 될 수 없다.

    - 관계형 데이터베이스의 특성상 일대다, 다대일 관계는 항상 다 쪽에 외래 키가 있기 때문.
    - 따라서 @OneToMany, @ManyToOne 둘 중에 연관관계의 주인은 항상 다 쪽인 @ManyToOne 을 사용한 곳이다.
    - 이런 이유로 @ManyToOne 에는 `mappedBy` 속성이 없다.
- 일대다 양방향 매핑이 가능하게 하려면, 일대다 단방향 매핑 반대편에 같은 외래 키를 사용하는 다대일 단방향 매핑을 읽기 전용으로 하나 추가하면 된다.
- `@JoinColumn(insertable=false, updatable=false)`



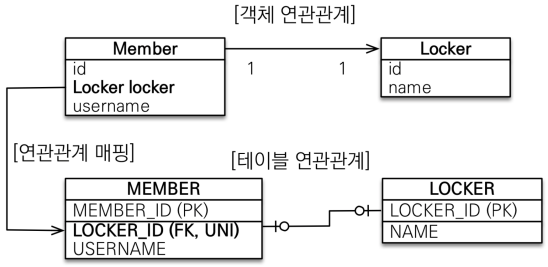
6.3 일대일 [1:1] p.214

- 테이블 관계에서 일대다, 다대일 관계는 항상 다(N)쪽이 외래키를 가진다.
- 반면에 일대일 관계는 주 테이블이나 대상 테이블 둘 중 어느곳이나 외래 키를 가질 수 있다.

6.3.1 주 테이블에 외래 키 p.215

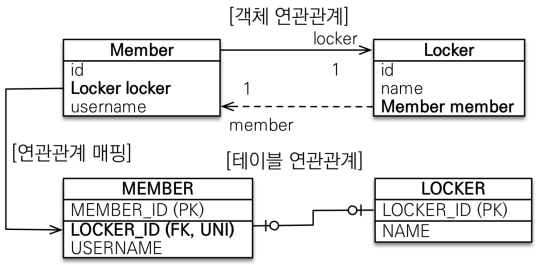
- 객체지향 개발자들이 선호
- 주 객체가 대상 객체를 참조하는 것처럼 주 테이블에 외래 키를 두고 대상 테이블을 참조한다.
- 주 테이블이 외래 키를 가지고 있으므로 주 테이블만 확인해도 대상 테이블과 연관관계가 있는지 알 수 있다.

단방향



- 주 테이블 : MEMBER, 대상 테이블 : LOCKER
- 일대일 관계이므로 `@OneToOne` 사용
- 데이터베이스에는 `LOCKER_ID` 외래키에 유니크 제약조건(UNI) 추가
- 이 관계는 다대일 단방향(@ManyToOne)과 거의 비슷

양방향



- MEMBER 테이블이 **외래 키를 가지고 있으므로** Member 엔티티에 있는 `Member.lcoker` 가 **연관관계의 주인**이다.
- 따라서 반대 매핑인 사물함의 `Locker.member` 는 `mappedBy` 를 선언해서 연관관계의 주인이 아니라고 설정

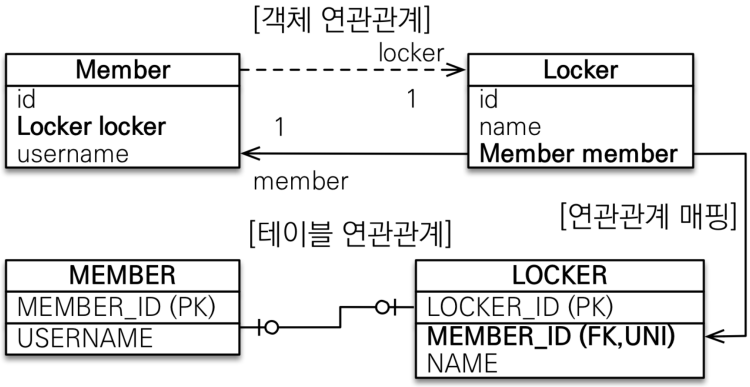
6.3.2 대상 테이블에 외래 키 p.218

- 테이블 관계를 일대일에서 일대다로 변경할 테이블 구조를 그대로 유지할 수 있다.

단방향

- 일대일 관계 중 대상 테이블에 외래 키가 있는 단방향 관계는 JPA에서 지원하지 않는다.
- 양방향 관계는 지원한다.

양방향



- 일대일 매핑에서 대상 테이블에 외래 키를 두고 싶으면 이렇게 양방향으로 매핑해야 한다.
- 주 엔티티인 Member 엔티티 대신에 대상 엔티티인 Locker 를 연관관계의 주인으로 만들어서 LOKCER 테이블의 외래 키를 관리하도록 했다.

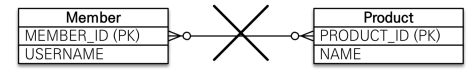
> 💡 부모 테이블의 기본 키를 자식 테이블에서도 기본 키로 사용하는 일대일 관계는 7.3.5절 참고

> 프록시를 사용할 때 외래키를 직접 관리하지 않는 일대일 관계는 지연 로딩으로 설정해도 즉시 로딩된다.
> 예를 들어 위 예제에서 `Locker.member` 는 지연 로딩할 수 있지만, `Member.locker`는 지연 로딩으로 설정해도 즉시 로딩된다.
> 프록시의 한계 때문에 발생하는 문제인데, 프록시 대신에 `bytecode instrumentaion` 을 사용하면 해결할 수 있다.
> (프록시와 지연 로딩은 8장 참고)

6.4 다대다 [N:M] p.220

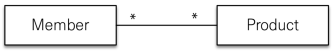
관계형 데이터베이스

- 정규화된 테이블 2개로 다대다 관계를 표현할 수 없다.
- 연결 테이블을 추가해서 일대다, 다대일 관계로 풀어내야한다.



객체

- **컬렉션을 사용해서 객체 2개로 다대다 관계 가능하다**




```
@Entity
public class Member {
    //
    @Id
    @Column(name = "MEMBER_ID")
    private String id;

    private String username;

    @ManyToMany
    @JoinTable(name = "MEMBER_PRODUCT",
            joinColumn = @JoinColumn(name = "MEMBER_ID"),
            inverseJoinColumns = @JoinColumn(name = "PRODUCT_ID"))
    private List<Product> products = new ArrayList<Product>();
}
```

```
@Entity
public class Product {
    //
    @Id
    @Column(name = "PRODUCT_ID")
    private String id;

    private String name;
}
```

- 회원 엔티티와 상품 엔티티를 `@ManyToMany` 로 매핑.
- `@ManyToMany` 와 `@JoinTable` 을 사용해서 연결 테이블을 바로 매핑함.

- 따라서 회원과 상품을 연결하는 회원_상품 (Member_Product) 엔티티 없이 매핑을 완료할 수 있다.

@JoinTable

속성	설명	예제 적용
name	연결 테이블을 지정한다.	MEMBER_PRODUCT 테이블을 선택
joinColumns	현재 방향인 회원과 매핑할 조인 컬럼 정보를 지정한다.	MEMBER_ID 로 지정
inverseJoinColumns	반대 방향인 상품과 매핑할 조인 컬럼 정보를 지정한다.	PROUCT_ID 로 지정

6.4.2 다대다: 양방향 p224

- 다대다 매핑으로 역방향도 `@ManyToMany` 를 사용
- 양쪽 중 원하는 곳에 `mappedBy` 를 지정 ⇒ `mappedBy` 가 없는 곳이 연관관계의 주인

```
@Entity
public class Product {
    //
    @Id
    private String id;

    @ManyToMany(mappedBy = "products") // 역방향 추가
    private List<Member> members;
}
```

6.4.3 다대다: 매핑의 한계와 극복, 연결 엔티티 사용 p.225

- `ManyToMany` 매핑이 **편리해 보이지만 실무에서 사용 X**
- 연결 테이블이 단순히 연결만 하고 끝나지 않음

⇒ 연결 테이블을 매핑하는 **연결 엔티티**를 만들고 이곳에 추가한 컬럼들을 매핑해야 한다.

```
@Entity
@IdClass(MemberProductId.class)
public class MemberProduct {
    //
    @Id
    @ManyToOne
    @JoinColumn(name = "MEMBER_ID")
    private Member member; // MemberProductId.member 와 연결

    @Id
    @ManyToOne
    @JoinColumn(name = "PRODUCT_ID")
    private Product product; // MemberProductId.product 와 연결
```

```
public class MemberProductId implements Serializab
    //
    private String member; // MemberProduct.member
    private String product; // MemberProduct.produ

    // hashCode and equals

    @Override
    public boolean equals(Object o) {...}

    @Override
    public int hashCode() {...}
```

```
    private int orderAmount;
}
```

- 기본키를 매핑하는 `@Id` 와 외래 키를 매핑하는 `@JoinColumn` 을 동시에 사용해서 기본 키 + 외래키를 한번에 매핑했다.
- `@IdClass` 를 사용해서 복합 기본 키를 매핑했다.

복합 기본 키

- 회원상품 엔티티(`MemberProduct`)는 기본 키가 `MEMBER_ID` 와 `PRODUCT_ID` 로 이루어진 복합 기본 키다.
- JPA 에서 복합키를 사용하려면 별도의 식별자 클래스(`MemberProductId`)를 만들어야한다.
- 그리고 엔티티에 `@IdClass` 를 사용해서 식별자 클래스를 지정하면 된다.

▼ **식별자 클래스**(`MemberProductId`)

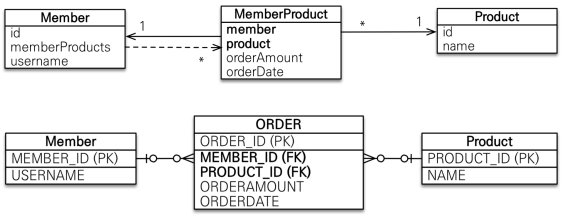
- 복합 키는 별도의 식별자 클래스로 만들어야 한다.
- `Serializable` 을 구현해야 한다.
- `equals` 와 `hashCode` 메소드를 구현해야 한다.
- 기본 생성자가 있어야 한다.
- 식별자 클래스는 public 이어야 한다.
- `@IdClass` 를 사용하는 방법 외에 `@EmbeddedId` 를 사용하는 방법도 있다. ⇒ @IdClass, @EmbeddedId 는 7.3절 참고

식별 관계

- 부모 테이블의 기본 키를 받아서 자신의 기본 키 + 외래 키로 사용하는 것을 데이터 베이스 용어로 식별 관계라 한다.

6.4.4 다대다 : 새로운 기본 키 사용 p.231

- 기본키 생성 전략 : ORM 매핑 시 복합키를 만들지 않아도 됨.




```
@Entity
public class Order {
    //
    @Id
    @GeneratedValue
    @Column(name = "ORDER_ID")
    private Long Id;

    @ManyToOne
    @JoinColumn(name = "MEMBER_ID")
    private Member member;
```

```
    @ManyToOne
    @JoinColumn(name = "PRODUCT_ID")
    private Product product;

    private int orderAmount;
}
```

6.4.5 다대다 연관관계 정리 p.233

- 다대다 관계를 일대다 다대일 관계로 풀어내기 위해 연결 테이블을 만들 때 식별자를 어떻게 구성해야 할지 선택해야 한다.
  - 식별관계 : 받아온 식별자를 기본 키 + 외래 키로 사용한다.
  - 비식별관계 : 받아온 식별자는 외래 키로만 사용하고 새로운 식별자를 추가한다.
- 객체 입장에서 보면 **비식별 관계**를 사용하는 것이 복합 키를 위한 식별자 클래스를 만들지 않아도 되므로 단순하고 편리하게 ORM 매핑을 할 수 있다.
- 7.3절 참고