JPA 데이터매핑(연관관계매핑)

1. JPA에서 "연관관계 매핑"이란?

JPA(Java Persistence API)는 **객체와 데이터베이스 테이블을 매핑**해주는 ORM(Object Relational Mapping) 기술.

하지만 **객체의 세계와 테이블의 세계는 다릅니다.**

구분	객체(Object)	테이블(Table)
관계 표현	필드로 참조 (ex: `member.getTeam()`)	외래키(Foreign Key)로 참조
방향성	단방향, 양방향 존재	항상 양방향 (FK 기준)
식별	참조를 통해	기본키, 외래키를 통해

그래서 객체 간 관계(연관관계)를 어떻게 **테이블의 외래키와 연결할지**를 지정하는 것이 **연관관계 매핑이다**.

---

2. 연관관계의 종류

JPA에서는 **엔티티 간의 관계**를 4가지로 나눕니다:

관계	예시	설명
`@ManyToOne`	회원 N : 팀 1	가장 흔한 관계 (N쪽이 FK 소유)
`@OneToMany`	팀 1 : 회원 N	주로 조회용 (FK는 반대쪽에 있음)
`@OneToOne`	회원 1 : 프로필 1	일대일 관계
`@ManyToMany`	학생 N : 강의 N	중간 테이블 필요 (실무에서는 잘 안 씀)

---

3. 예제: 회원(Member) ↔ 팀(Team)

(1) 테이블 구조

```
TEAM
- id (PK)
- name

MEMBER
- id (PK)
- username
- team_id (FK)
```

---

(2) 엔티티 매핑

Team 엔티티

```
@Entity
public class Team {

    @Id @GeneratedValue
    private Long id;

    private String name;

    // 양방향 관계일 경우: mappedBy로 주인 아님을 표시
    @OneToMany(mappedBy = "team")
    private List<Member> members = new ArrayList<>();
```

Member 엔티티

```
@Entity
public class Member {

    @Id @GeneratedValue
    private Long id;

    private String username;

    // FK를 가진 쪽이 "연관관계의 주인"
```

```
    @ManyToOne
    @JoinColumn(name = "team_id")  // FK 컬럼 이름 지정
    private Team team;
}
```

4. 연관관계의 "주인(Owner)" 개념

- **연관관계의 주인**: 실제로 **DB 외래키를 가지고 있는 엔티티**
- **주인이 아닌 쪽(mappedBy 사용)**: 단순히 **조회용(읽기 전용)**

> 외래키는 하나뿐이므로, JPA도 실제 DB 변경 시점을 결정하기 위해 “주인”을 명시해야 함.

---

5. 양방향 관계 설정 시 주의점

JPA에서는 아래처럼 두 객체 모두를 연결해야 함.

```
Team team = new Team();
team.setName("Team A");
em.persist(team);

Member member = new Member();
member.setUsername("User1");
member.setTeam(team); // 연관관계 주인 쪽 설정
em.persist(member);

// 양방향일 때는 아래 코드도 같이 해줘야 함 (연관관계 편의 메서드)
team.getMembers().add(member);
```

하지만 위처럼 매번 두 번 써주는 건 실수 유발.

그래서 “**연관관계 편의 메서드**”를 사용함.

연관관계 편의 메서드 예시

```
@Entity
public class Member {
```

```
    ...
    public void changeTeam(Team team) {
        this.team = team;
        team.getMembers().add(this);
    }
}
```

6. 지연로딩(LAZY) vs 즉시로딩(EAGER)

관계 매핑 시 fetch 전략도 중요함.

옵션	설명	특징
`FetchType.LAZY`	필요할 때만 쿼리 실행	성능 최적화 (권장)
`FetchType.EAGER`	항상 즉시 조회	쿼리 폭발 주의!

예:

```
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "team_id")
private Team team;
```

7. 실무에서의 베스트 프랙티스

연관관계 매핑 시 권장 사항:

1. **단방향 매핑 먼저 설계**하고, 필요할 때만 양방향 추가
2. **연관관계의 주인**은 외래키를 가진 쪽으로
3. **지연로딩(LAZY)** 기본
4. **연관관계 편의 메서드**로 양방향 동기화
5. `@ManyToMany`는 피하고 **중간 엔티티(매핑 테이블)** 사용

8. 시각화로 요약

```
[Member] *  ---------------------- 1 [Team]
  FK: team_id                PK: id
```

- Member가 FK(team_id)를 가지고 있으므로 **연관관계의 주인**
- Team은 mappedBy="team"으로 읽기 전용