DAOUTECH

연관관계 매핑

플랫폼서비스팀

김설한

2021.09.29

• 목차

- 연관 관계 정의 규칙
- 다중성
- 연관관계 편의 기능

01

연관 관계 정의 규칙

연관 관계 정의 규칙

연관 관계를 매핑할 때, 아래 3가지를 고려해야 한다.

- **방향** : 단방향, 양방향 (객체 참조)
- **연관 관계의 주인** : 양방향일 때, 연관 관계에서의 관리 주체
- **다중성** : 다대일(N:1), 일대다(1:N), 일대일(1:1), 다대다(N:M)

연관 관계 정의 규칙

- **방향 (단 방향/양 방향)**
  - DB 테이블은 외래 키를 통해 양 쪽 테이블 조인이 가능하다.

    하지만, 객체는 참조 용 필드를 갖는 객체만 다른 객체를 참조하는 것이 가능하다.
  - 두 객체 사이에 하나의 객체만 참조 용 필드를 갖고 참조하면 **단 방향** 관계.
  - 두 객체 모두가 각각 참조 용 필드를 갖으면 **양 방향** 관계.
  - 엄밀히 말하면 객체는 양 방향 관계는 없고,

    두 객체가 서로를 향하는 단 방향 참조를 각각 가져 양방향 관계처럼 사용하는 것

연관 관계 정의 규칙

- **방향 (단 방향/양 방향)**
  - JPA를 사용해 데이터베이스와 패러다임을 맞추기 위해서

    객체는 단 방향 연관 관계를 가질지, 양 방향 연관 관계를 가질 지 선택해야한다.
  - 선택은 **비즈니스 로직에서 객체가 서로 참조가 필요한지** 고민해 보면 된다.
    - board.getPost() 처럼 참조가 필요하면 Board -> Post 단 방향 참조
    - post.getBoard() 처럼 참조가 필요하면 Post -> Board 단 방향 참조
  - 비즈니스 로직에 맞게 선택했는데 두 객체가 서로 단 방향 참조를 했다면, 양 방향 연관 관계
  - But, 객체 입장에서 양 방향 매핑을 했을 때 오히려 복잡해 질 수 있으므로

    기본적으로는 단 방향 매핑을 하고, 역 방향으로 객체 탐색이 꼭 필요할 때만 추가

연관 관계 정의 규칙

- **연관 관계의 주인**
  - 양방향(2개의 단 방향) 관계 중,

    제어의 권한(외래 키를 비롯한 테이블 레코드 저장/수정/삭제)**을 갖는 주체**
  - 두 객체(A, B)가 양 방향 관계(단 방향 관계 1개 : A->B, B->A)를 맺을 때,

    연관 관계의 주인을 지정해야 한다.
  - 기본적으로, **외래 키를 갖는 곳**을 **연관 관계 주인**으로 정하면 된다.
  - 데이터 베이스 관점에선, 연관 관계 주인만 제어를 해 주는게 맞지만

    객체 관점에선, 두 객체 데이터 동기화를 위해 함께 변경해 주는 것이 좋다. (연관관계 편의 메서드)
  - 연관 관계의 주인이 아닌 객체에서 mappedBy 속성을 사용해 주인을 지정해주면 된다.

02

다중성

다중성

- **다대일 (N:1)**
  - 게시판(Board)과 게시글(Post)의 관계로 예를 들면,
    - 하나(1)의 게시판(Board)에는 여러(N) 게시글(Post)를 작성할 수 있다.
    - 하나(1)의 게시글(Post)은 하나의(1) 게시판(Board)에만 작성할 수 있다.
    - 게시글(Post)와 게시판(Board)는 다대일(N:1) 관계를 갖는다.
  - 즉, 외래키를 게시글(N)이 관리하는 일반적인 형태이다.

    (데이터베이스는 무조건 다(N)쪽 테이블이 외래 키를 갖는다.)

다중성

- 다대일 (N:1) 단 방향

```java
@Entity
public class Post {

    @Id @GeneratedValue
    @Column(name = "POST_ID")
    private Long id;

    @Column(name = "TITLE")
    private String title;

    @ManyToOne
    @JoinColumn(name = "BOARD_ID")
    private Board board;

}

@Entity
public class Board {

    @Id @GeneratedValue
    @Column(name = "BOARD_ID")
    private Long id;

    @Column(name = "TITLE")
    private String title;

}
```

- 다대일 (N:1) 양 방향

```java
@Entity
public class Post {

    @Id @GeneratedValue
    @Column(name = "POST_ID")
    private Long id;

    @Column(name = "TITLE")
    private String title;

    @ManyToOne
    @JoinColumn(name = "BOARD_ID")
    private Board board;

}

@Entity
public class Board {

    @Id @GeneratedValue
    @Column(name = "BOARD_ID")
    private Long id;

    @Column(name = "TITLE")
    private String title;

    @OneToMany(mappedBy="board")
    private List<Post> posts = new ArrayList<>();

}
```

다중성

- **일대다 (1:N)**
  - 다 대일의 반대 입장. 연관 관계의 주인을 일(1)쪽에 두는 것.
  - 데이터베이스 입장에서는 무조건 다(N)쪽에서 외래 키를 관리하는데,

    일(1)쪽 객체에서 다(N)쪽 객체를 생성/수정/삭제한다.

    (실무에서는 일대다(1:N) 단 방향은 거의 사용하지 않는다.)

```java
post = new Post();
post.setTitle("가입인사");
entityManager.persist(post); // post 저장

Board board = new Board();
board.setTitle("자유게시판");
board.getPosts().add(post);
entityManager.persist(board); // board 저장
```

Post insert 쿼리

(1) Board insert 쿼리
→ (2) Post update 쿼리

Post 테이블의 FK(BOARD_ID)를 저장할 방법이 없어, 조인 및 업데이트 쿼리가 발생한다.

```
Hibernate: insert into board (board_id, title) values (null, ?)
Hibernate: update post set board_id=? where post_id=?
```

```java
@Entity
public class Post {

    @Id @GeneratedValue
    @Column(name = "POST_ID")
    private Long id;

    @Column(name = "TITLE")
    private String title;
}

@Entity
public class Board {

    @Id @GeneratedValue
    @Column(name = "BOARD_ID")
    private Long id;

    @Column(name = "TITLE")
    private String title;

    @OneToMany
    @JoinColumn(name="POST_ID")
    private List<Post> posts = new ArrayList<>();
}
```

다중성

- **일대다 (1:N)**
  - 단점
    - ➢ 엔티티가 관리하는 외래 키가 다른 테이블에 있음
    - ➢ 1에 해당 하는 객체만 수정했는데, 다른 수정이 생겨 쿼리가 발생하는 것

      : Board를 저장했는데 Post가 수정됨

  - 일대다(1:N) 단 방향 매핑이 필요한 경우는 그냥 다대일(N:1) 양방향 매핑을 하는 것이 유지보수에 훨씬 수월하다. (단, JPA 값 타입을 대신해 사용하는 경우에는 유용하다.)

  - 즉, **일대다(1:N) 단 방향을 쓰지 말고, 다대일(N:1) 양방향으로 사용하자!**

다중성

- **일대일 (1:1)**
  - 주 테이블에 외래 키를 넣을 수도 있고, 대상 테이블에 외래 키를 넣을 수 도 있다.
  - 일대일(1:1) 단 방향 : 외래 키를 주 테이블이 갖고 있는 상태.
    - ➢ Post테이블(주 테이블)에서 외래 키(FK)인 Attach 테이블(대상 테이블)의 PK를 갖는 상태.

```java
@Entity
public class Post {
  @Id @GeneratedValue
  @Column(name = "POST_ID")
  private Long id;

  @Column(name = "TITLE")
  private String title;

  @OneToOne @JoinColumn(name = "ATTACH_ID")
  private Attach attach;
}

@Entity public class Attach {
  @Id @GeneratedValue
  @Column(name = "ATTACH_ID")
  private Long id;

  private String name;
}
```

← 게시 글에 첨부파일을 1개만 등록할 수 있는 경우

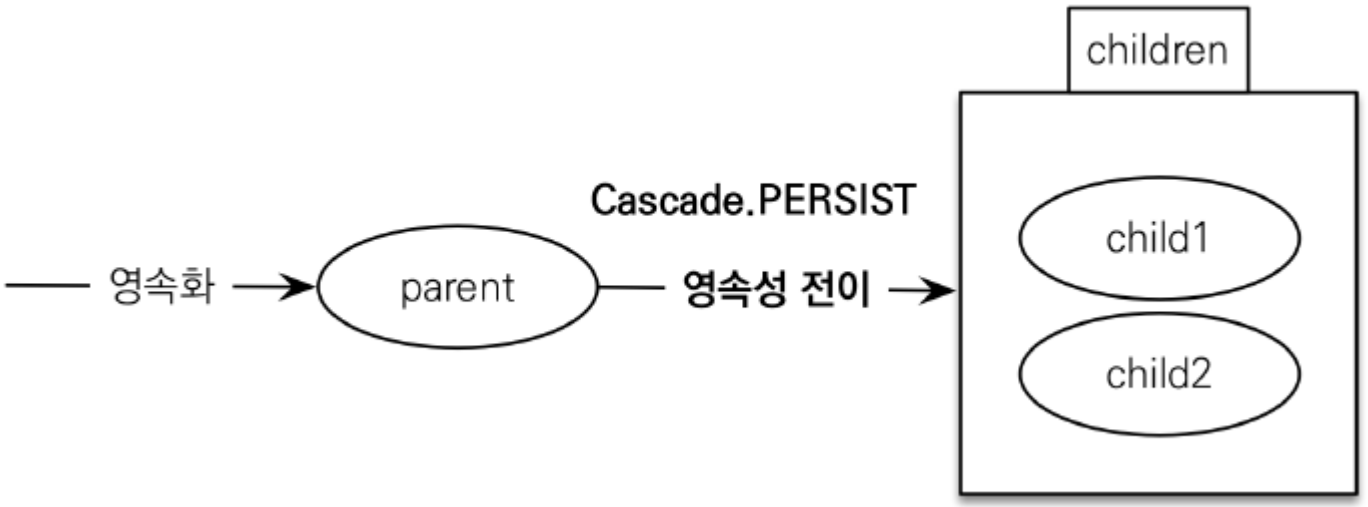
03

연관관계 편의 기능

연관관계 편의 기능

- **영속성 전이 (CASCADE)**
  - 특정 엔티티를 영속 상태로 만들 때, 연관된 엔티티도 함께 영속 상태로 만드는 기능
  - 다대일 관계에서 일(1) 쪽에 연관관계 엔티티를 추가하고, persist 하면 다(N)쪽도 persist 된다.
    - 예시 : 부모 엔티티를 저장할 때, 자식 엔티티도 함께 저장

```
@OneToMany(mappedBy="parent", cascade = CascadeType.PERSIST)
```



- ✓ **ALL : 모두 적용**
- ✓ **PERSIST: 영속**
- ✓ **REMOVE : 삭제**

...

연관관계 편의 기능

- **고아 객체**
  - 고아 객체 제거 : 부모 엔티티와 연관관계가 끊어진 자식 엔티티를 자동 삭제
    - orphanRemoval = true

```
Parent parent = em.find(Parent.class, id);
parent.getChildren().remove(0);
// 0번 인덱스의 자식 엔티티를 컬렉션에서 제거
// 그러면 DB에서 delete 쿼리가 날아감
```

  - 주의 사항
    - 참조하는 곳이 하나이며, 특정 엔티티가 개인 소유할 때 사용해야한다.
    - @OneToOne, @OneToMany에서만 사용 가능
    - CascadeType.REMOVE와 유사하게 동작한다.

연관관계 편의 기능

- **영속성 전이 + 고아 객체 제거 의 생명 주기**
  - CascadeType.ALL + orphanRemoval = true
  - 두 옵션을 모두 활성화하면, 부모 엔티티를 통해 자식 엔티티의 생명주기를 관리할 수 있다.
    - ➢ DB로 따지자면, 자식 엔티티의 DAO나 Repository가 없어도 된다는 의미
  - 도메인 주도 설계(DDD)의 Aggregate Root 개념을 구현할 때 유용하다.

감사합니다 !