😄

# 시니어 미션

- **JPA의 깊은 개념을 이해하고 실제 서비스에서 발생하는 문제를 해결**한다.
- **복잡한 연관관계 매핑 & 성능 최적화**를 고려한 엔티티 설계를 적용한다.

---

## 📌 미션 상세 내용

### 1️⃣ 성능을 고려한 연관관계 매핑 & 최적화 적용

- `@OneToMany` **컬렉션을 조회할 때** `List<MemberPrefer>` **를** `Set<MemberPrefer>` **로 변경 후 차이점 분석**

> 리스트(List)는, 인덱스 컬럼이 명시되지 않은 경우, Hibernate에서는 단순한 bag(순서가 없는 집합)으로 처리됨..

> Hibernate를 사용할 때 주목할 만한 차이점 중 하나는, 두 개의 서로 다른 리스트를 한 번의 쿼리로 fetch(가져오기)할 수 없다.
>
> 예를 들어, Member 엔티티가 `contacts` 리스트와 `addresses` 리스트를 가진다면, 이 두 리스트를 모두 포함해서 Member 를 로드하려고 할 때 한 번의 쿼리로 처리할 수 없다.

> 이 경우의 해결책은 두 개의 쿼리를 사용하는 것(이렇게 하면 Cartesian product를 피할 수 있음)이거나, 두 컬렉션 중 적어도 하나를 List 대신 Set으로 바꾸는 것임.

> 하지만 Hibernate에서 Set을 사용하는 것은 종종 어려움.
>
> → **엔티티에** `equals` **와** `hashCode` **를 정의해야 하고**, 엔티티에 불변의 기능적 키(immutable functional key)가 없는 경우에는 이 작업이 까다롭다.

> 또한 중복이 불가한 Set은  기존에 가지고 있던 Entity(값 타입)에 중복된 데이터가 있는지 비교를 해야 하는데 이 시점에 Set에 있는 모든 데이터를 로딩해야만 하고 이 때 Proxy가 강제로 초기화된다. 별로다.

- **데이터 정합성을 고려하여 `orphanRemoval = true`가 필요한 곳 확인 후 적용**

### 1. 자식 엔티티가 오직 하나의 부모에만 종속되고, 부모가 관계를 끊으면 더 이상 의미가 없는 경우

- 예: 댓글(자식)은 게시글(부모)에 종속. 게시글에서 댓글을 제거하면 DB에서도 삭제되어야 정합성 유지됨.

```
@OneToMany(mappedBy = "post", cascade = CascadeType.ALL, orpha
nRemoval = true)
private List<Comment> comments = new ArrayList<>();
```

### 2. 부모 객체에서 자식 객체를 remove() 했을 때, DB에서도 삭제되어야 할 경우

→ 고아 객체를 고려한듯?

### 3. 양방향 연관관계에서 주인이 아닌 쪽을 수정하지 않고, 컬렉션에서만 제거하는 방식으로 삭제를 수행하고자 할 때

---

## 2️⃣ 트랜잭션 & 동시성 이슈 처리

- **하나의 트랜잭션에서 여러 엔티티를 처리하는 비즈니스 로직 작성**
  - 예) `Member`가 탈퇴할 경우 **관련된 모든 데이터를 삭제하는 API** 구현
  - `@Transactional`을 적용하고, `@Modifying`을 활용하여 **Batch Delete 쿼리 최적화**

```
@Service
@RequiredArgsConstructor
```

```
public class MemberService {

    private final MemberRepository memberRepository;
    private final PostRepository postRepository;
    private final CommentRepository commentRepository;
    private final LikeRepository likeRepository;
    private final InbodyRepository inbodyRepository;

    @Transactional
    public void deleteMember(Long memberId) {

        commentRepository.deleteAllByMemberId(memberId);
        postRepository.deleteAllByMemberId(memberId);

        memberRepository.deleteById(memberId);
    }
}
```

```
public interface CommentRepository extends JpaRepository<Comment, Long
    @Modifying
    @Query("DELETE FROM Comment c WHERE c.member.id = :memberId")
    void deleteAllByMemberId(@Param("memberId") Long memberId);
}
```

(같은 방식으로 postRepository, posts 구성하면 됨.)

```
@OneToMany(mappedBy = "post", cascade = CascadeType.REMOVE, orphan
private List<Comment> comments = new ArrayList<>();
```

- **동시성 문제가 발생할 수 있는 시나리오**를 고민하고 해결책 적용
  - 예) 같은 회원이 동시에 같은 `Store`를 찜하려고 할 때 중복이 발생하지 않도록 `@Lock` 사용
  - 다양한 락킹 전략에 대해 공부해보고, 이를 정리하기

예전에 동시성 문제를 해결하기 위해 다양한 동시성 처리를 해본 레포입니다.

https://github.com/dongjune8931/Coupon-system