2025.03.18

김희진

Spring Data JPA N+1 문제

목차

N+1 문제란?

요청이 1개의 쿼리로 처리되길 기대했는데 추가로 N개의 쿼리가 발생하는 현상

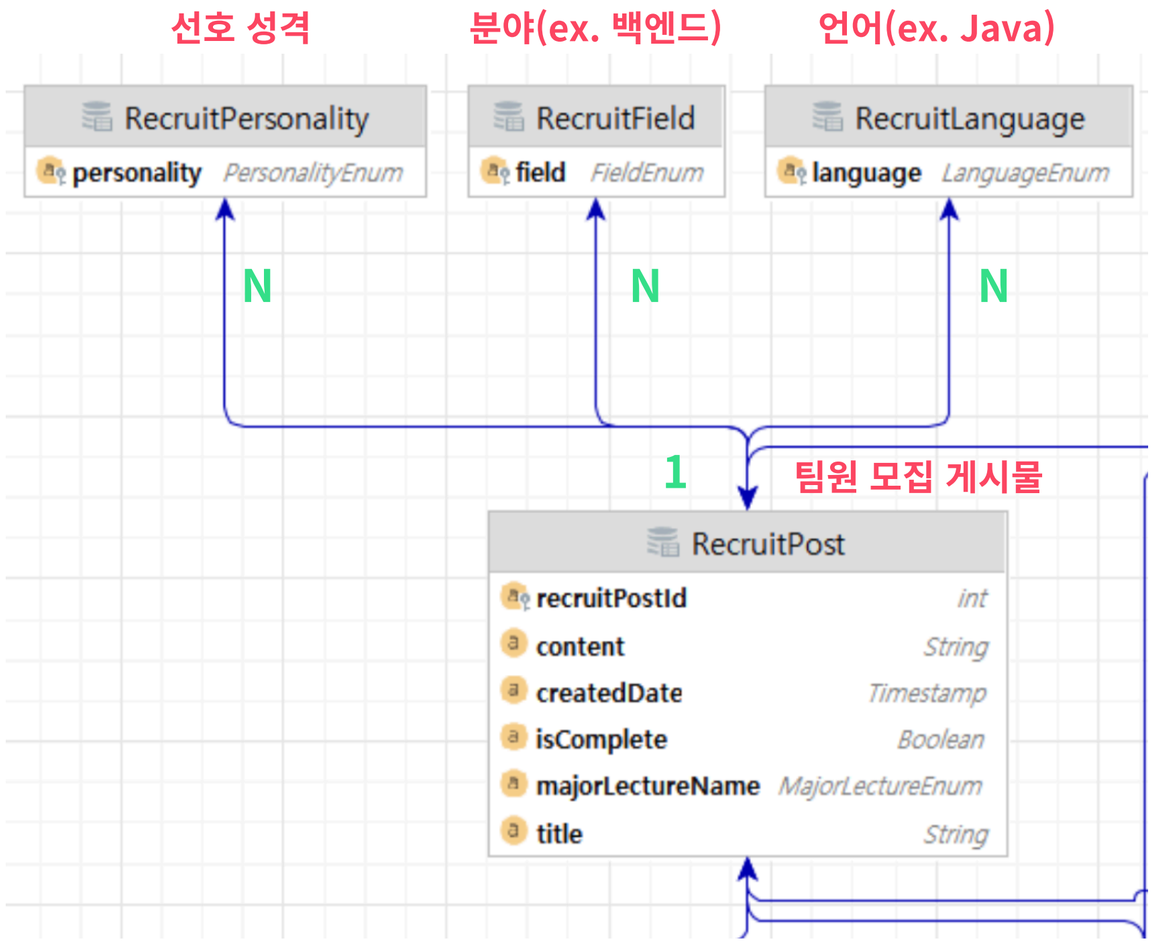
=> 흐름상 1+N 문제라고 부르는게 더 맞는

01

프로젝트 상황

ERD

[프로젝트 팀원 모집 기능]



- 팀원 모집 게시물: 선호 성격 = 1:N
- 팀원 모집 게시물: 분야 = 1:N
- 팀원 모집 게시물: 언어 = 1:N

01

프로젝트 상황

Entity

```java
@Setter  dabeann +2
@Entity
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@Getter
@Table(name = "recruit_post")
public class RecruitPost {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "recruit_post_id", nullable = false)
    private int recruitPostId;

    // N:1 user
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    private User userId;

    @Column(name = "title", nullable = false)
    private String title;

    @Column(name = "content", length = 1000, nullable = false)
    private String content;

    @Column(name = "major_lecture_name", length = 200)
    @Enumerated(EnumType.STRING)
    private MajorLectureEnum majorLectureName;
```

[RecruitPost Entity]

```java
@Column(name = "created_date")
private Timestamp createdDate;

@Column(name = "is_complete", nullable = false, columnDefinition = "boolean default false")
private Boolean isComplete;

// 1:N recruitField
@OneToMany(mappedBy = "recruitPostId")
private List<RecruitField> recruitFields;

// 1:N recruitLanguage
@OneToMany(mappedBy = "recruitPostId")
private List<RecruitLanguage> recruitLanguages;

// 1:N recruitPersonality
@OneToMany(mappedBy = "recruitPostId")
private List<RecruitPersonality> recruitPersonalities;

// 1:N Application
@OneToMany(mappedBy = "recruitPostId")
private List<Application> applications;
```

- **양방향**
- **FetchType.LAZY(지연로딩)**
  : 연관된 엔티티를 처음에는 조회하지 않고, 실제로 해당 엔티티가 필요한 시점에 조회하는 방식

01

프로젝트 상황

Service & Repository

[사용자 맞춤 팀원 모집 글 추천 로직]

```java
@Override  1개 사용 위치  heejjinkim *
public List<RecRecruitPostDTO> recommendRecruitPost(int userId) {
    // user 가져오기
    User user = memberRepository.findUserWithDetailsByUserId(userId);

    // 1) isComplete==false인 게시물 && 자신(user)이 작성한 게시물 제외
    List<RecruitPost> recommendedPostList = recruitPostRepository.findAllByIsCompleteAndUserIdNot( isComplete: false, user);

    // 각 게시물에 대한 우선순위 부여
    HashMap<RecruitPost, Integer> priorityMap = calculatePriorities(user, recommendedPostList

    // priorityMap을 값으로 정렬하여 상위 4개 선택 (우선순위 높은 4개)
    List<RecruitPost> topPosts = selectTopPriorityPosts(priorityMap, limitSize: 4);

    // DTO 생성 및 반환
    return createRecruitPostDTOList(topPosts);
}
```

```java
@Repository  17개 사용 위치  juhno1023 +1 *
public interface RecruitPostRepository extends JpaRepository<RecruitPost, Integer> {
    RecruitPost findRecruitPostByRecruitPostId(int id);  4개 사용 위치  juhno1023

    RecruitPost findRecruitPostByTitle(String title);  4개 사용 위치  juhno1023

    RecruitPost save(RecruitPost recruitPost);  juhno1023

    List<RecruitPost> findAllByIsCompleteAndUserIdNot(Boolean isComplete, User user);

}
```

02

트러블 슈팅

Service

[사용자 맞춤 팀원 모집 글 추천 로직]

```
@Override  1개 사용 위치  heejjinkim *
public List<RecRecruitPostDTO> recommendRecruitPost(int userId) {
    // user 가져오기
    User user = memberRepository.findUserWithDetailsByUserId(userId);

    // 1) isComplete==false인 게시물 && 자신(user)이 작성한 게시물 제외
    List<RecruitPost> recommendedPostList = recruitPostRepository.findAllByIsCompleteAndUserIdNot( isComplete: false, user);

    // 각 게시물에 대한 우선순위 부여
    HashMap<RecruitPost, Integer> priorityMap = calculatePriorities(user, recommendedPostList);

    // priorityMap을 값으로 정렬하여 상위 4개 선택 (우선순위 높은 4개)
    List<RecruitPost> topPosts = selectTopPriorityPosts(priorityMap, limitSize: 4);

    // DTO 생성 및 반환
    return createRecruitPostDTOList(topPosts);
}
```

-> 모든 팀원 모집 글 조회

```
// RecruitPost의 우선순위를 저장하는 map
HashMap<RecruitPost, Integer> priorityMap = new HashMap<RecruitPost, Integer>();
recommendedPostList.forEach(post -> {
    int priority = 0;
    // 여기서 언어, 관심분야, 성격 일치하는지 확인하여 우선순위 계산
    priority += calculatePriorityForMatchingAttributes(post, userLanguageList, userInterestFieldList, userPersonalityList);
    priorityMap.put(post, priority);
});
return priorityMap;
```

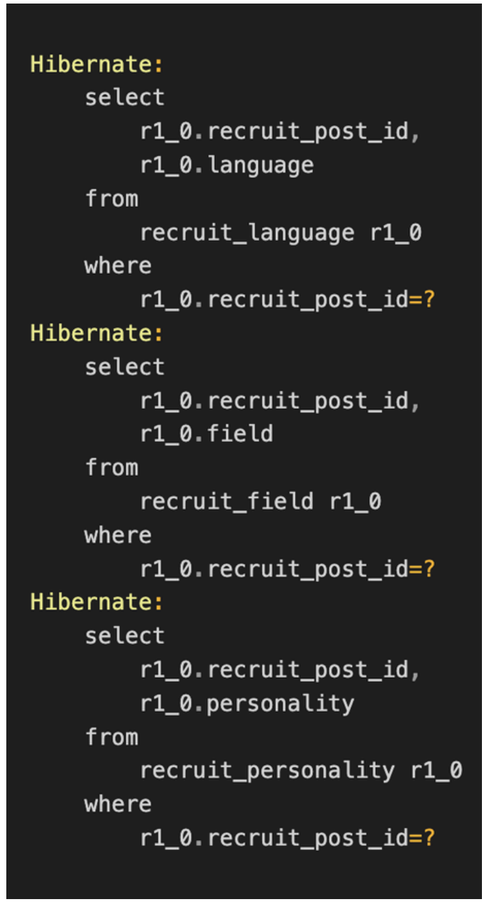
-> 각 팀원 모집글의 언어, 관심분야, 성격 조회

02

트러블 슈팅

로직 실행 결과

전체 RecruitPost 조회 쿼리 1번



```
Hibernate:
    select
        r1_0.recruit_post_id,
        r1_0.language
    from
        recruit_language r1_0
    where
        r1_0.recruit_post_id=?
Hibernate:
    select
        r1_0.recruit_post_id,
        r1_0.field
    from
        recruit_field r1_0
    where
        r1_0.recruit_post_id=?
Hibernate:
    select
        r1_0.recruit_post_id,
        r1_0.personality
    from
        recruit_personality r1_0
    where
        r1_0.recruit_post_id=?
```

*** N번**

RecruitPost 데이터가 N개 있을 때 왼쪽 로그가 총 N번 찍힘

- RecruitPost 마다 쿼리 3번
  - 언어 1번, 분야 1번, 성격 1번
- = **N*3번의 쿼리**

02

트러블 슈팅

N+1 쿼리 발생

[사용자 맞춤 팀원 모집 글 추천 로직]

```java
@Override  1개 사용 위치  heejjinkim *
public List<RecRecruitPostDTO> recommendRecruitPost(int userId) {
    // user 가져오기
    User user = memberRepository.findUserWithDetailsByUserId(userId);

    // 1) isComplete==false인 게시물 && 자신(user)이 작성한 게시물 제외
    List<RecruitPost> recommendedPostList = recruitPostRepository.findAllByIsCompleteAndUserIdNot( isComplete: false, user);

    // 각 게시물에 대한 우선순위 부여
    HashMap<RecruitPost, Integer> priorityMap = calculatePriorities(user, recommendedPostList);

    // priorityMap을 값으로 정렬하여 상위 4개 선택 (우선순위 높은 4개)
    List<RecruitPost> topPosts = selectTopPriorityPosts(priorityMap, limitSize: 4);

    // DTO 생성 및 반환
    return createRecruitPostDTOList(topPosts);
}
```

-> 팀원 모집 글 N개 조회 - 쿼리 1번

+ 1 + (3)N 번 쿼리 발생

```java
// RecruitPost의 우선순위를 저장하는 map
HashMap<RecruitPost, Integer> priorityMap = new HashMap<RecruitPost, Integer>();
recommendedPostList.forEach(post -> {
    int priority = 0;
    // 여기서 언어, 관심분야, 성격 일치하는지 확인하여 우선순위 계산
    priority += calculatePriorityForMatchingAttributes(post, userLanguageList, userInterestFieldList, userPersonalityList);
    priorityMap.put(post, priority);
});
return priorityMap;
```

-> N번 루프를 돌면서

각 글의 언어, 관심분야, 성격 조회 - 쿼리 N*3번

03

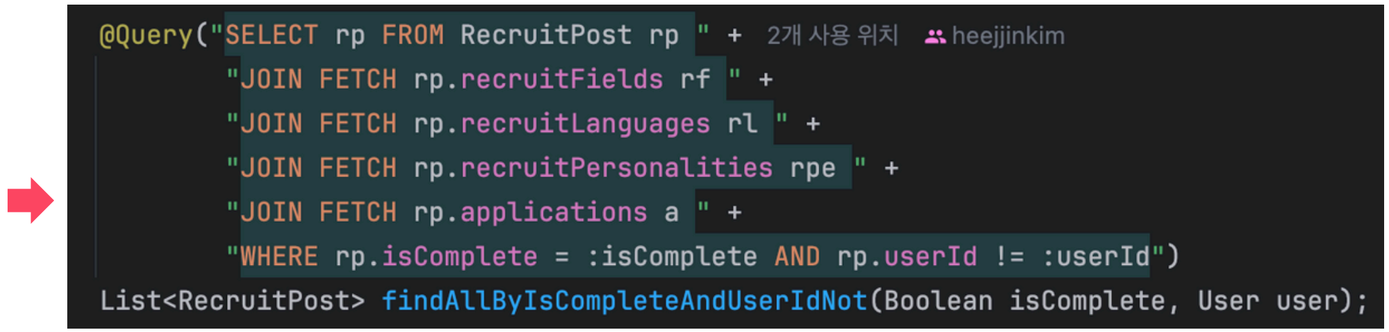
해결 과정 * JPQL: 엔티티 객체를 조회하는 객체지향 쿼리

1. JPQL의 FETCH JOIN 활용 : 한번의 쿼리로 연관된 엔티티를 같이 조회

[RecruitPostRepository]

Spring Data Jpa의 메소드는
메서드 이름을 분석해서 JPQL을 생성하여 실행

```
List<RecruitPost> findAllByIsCompleteAndUserIdNot(Boolean isComplete, User user);
```



```
@Query("SELECT rp FROM RecruitPost rp " +  2개 사용 위치  heejjinkim
        "JOIN FETCH rp.recruitFields rf " +
        "JOIN FETCH rp.recruitLanguages rl " +
        "JOIN FETCH rp.recruitPersonalities rpe " +
        "JOIN FETCH rp.applications a " +
        "WHERE rp.isComplete = :isComplete AND rp.userId != :userId")
List<RecruitPost> findAllByIsCompleteAndUserIdNot(Boolean isComplete, User user);
```

문제

[Hibernate MultipleBagFetchException 발생]

- 2개 이상의 OneToMany 자식 테이블에 Fetch Join했을때 발생하는 예외
- Bag Collecion: List는 순서가 있는 중복 가능한 컬렉션이며, Hibernate는 이를 "Bag"으로 취급
- 두 개 이상의 Bag을 함께 Fetch Join하면 카티시안 곱 발생으로 중복 데이터 발생 가능성 있음
- Hibernate가 중복 데이터를 어떻게 처리할지 결정할 수 없기 때문에 MultipleBagFetchException이 발생

>>> OneToMany 관계가 하나이면 문제 X

03

해결 과정

2. @EntityGraph 사용 : FetchType이 Lazy 인 연관관계 엔티티들을 한번에 (Left Join 으로) 조회

[RecruitPostRepository]

```
@EntityGraph(attributePaths = {"recruitFields", "recruitLanguages", "recruitPersonalities", "applications"},
        type = EntityGraph.EntityGraphType.FETCH)
List<RecruitPost> findAllByIsCompleteAndUserIdNot(Boolean isComplete, User user);
```

마찬가지로
Hibernate MultipleBagFetchException 발생

03

해결 과정

3. Querydsl 사용 : 정적 타입을 이용하여 SQL과 같은 쿼리를 생성

```
public List<RecruitPost> findAllByIsCompleteAndUserIdNot(Boolean isComplete, User user) {
    JPAQueryFactory queryFactory = new JPAQueryFactory(entityManager);

    return queryFactory
            .select(recruitPost)
            .from(recruitPost)
            .leftJoin(recruitPost.recruitFields, recruitField).fetchJoin()
            .leftJoin(recruitPost.recruitLanguages, recruitLanguage).fetchJoin()
            .leftJoin(recruitPost.recruitPersonalities, recruitPersonality).fetchJoin()
            .where(recruitPost.isComplete.eq(isComplete)
                    .and(recruitPost.user.id.ne(user.getId())))
            .fetch();
}
```

문제

마찬가지로
Hibernate MultipleBagFetchException 발생

MultipleBagFetchException해결

(1) List -> Set 변경
(2) fetch_size 설정

03

해결 과정

4. MultipleBagFetchException 해결 - (1) List -> Set 변경

[RecruitPost Entity]

```java
// 1:N recruitField
@OneToMany(mappedBy = "recruitPostId")
private Set<RecruitField> recruitFields;

// 1:N recruitLanguage
@OneToMany(mappedBy = "recruitPostId")
private Set<RecruitLanguage> recruitLanguages;

// 1:N recruitPersonality
@OneToMany(mappedBy = "recruitPostId")
private Set<RecruitPersonality> recruitPersonalities;
```

=>

```
Hibernate:
    /* <criteria> */ select
        r1_0.recruit_post_id,
        a1_0.recruit_post_id,
        a1_0.application_id,
        a1_0.content,
        a1_0.self_intro,
        a1_0.user_id,
        r1_0.content,
        r1_0.created_date,
        r1_0.is_complete,
        r1_0.major_lecture_name,
        r3_0.recruit_post_id,
        r3_0.field,
        r5_0.recruit_post_id,
        r5_0.language,
        r7_0.recruit_post_id,
        r7_0.personality,
        r1_0.title,
        r1_0.user_id
    from
        recruit_post r1_0
    left join
        application a1_0
            on r1_0.recruit_post_id=a1_0.recruit_post_id
    left join
        recruit_field r3_0
            on r1_0.recruit_post_id=r3_0.recruit_post_id
    left join
        recruit_language r5_0
            on r1_0.recruit_post_id=r5_0.recruit_post_id
    left join
        recruit_personality r7_0
            on r1_0.recruit_post_id=r7_0.recruit_post_id
    where
        r1_0.is_complete=?
        and r1_0.user_id<>?
```

MultipleBagFetchException 해결

- Set은 중복 데이터 발생하지 않으므로 에러 발생 X
- fetch join하여 한 번의 쿼리로 연관된 엔티티 모두 한 번에 조회

하지만, 여전히 카티시안 곱이 생기므로 성능 보장 X

03

해결 과정

4. MultipleBagFetchException 해결 - (2) batch size 설정

: 연관된 엔티티를 몇 개씩 가져올 지 설정
(IN 절에 들어갈 값 개수를 설정)

[application.yml]

```yaml
spring:
  jpa:
    properties:
      hibernate.default_batch_fetch_size: 1000
```

프로젝트 전역에 배치 사이즈 적용

[RecruitPost Entity]

```java
@BatchSize(size = 100)
@OneToMany(mappedBy = "recruitPostId")
private Set<RecruitField> recruitFields;
```

각각 배치 사이즈 적용

WHERE 절이 같은 여러 개의 SELECT 쿼리들을 하나의 IN 쿼리로 만들어줌

[기존 N개의 쿼리]

Select * From RecruitField Where RecruitPost.id = 1
Select * From RecruitField Where RecruitPost.id = 2
...

[1개의 IN 쿼리로]

Select * From RecruitField Where RecruitPost.id IN (1, 2....)

04

결론

[N+1 문제란?]

요청이 1개의 쿼리로 처리되길 기대했는데 추가로 N개의 쿼리가 발생하는 현상

[연관된 Collection이 하나일 때 해결방법]

1. JPQL Fetch Join
2. @EntityGraph
3. Querydsl
=> 두 개 이상이면 MultipleBagFetchException 발생

[연관된 Collection이 2개 이상일 때 해결방법]

1. **Batch Size 설정**
2. **List->Set 변경**

나만의 결론:

- fetch join은 페이징 안되고, 카티시안 곱 존재
- Set은 카티시안 곱 여전히 존재

=> Collection이 한 개이든 두 개이든 페이징 가능하고 성능을 보장해주는 batch size를 사용하자

Q & A