초격차 패키지 Online.

복잡한 쿼리의 작성과 응용

PART1 | Querydsl vs. Jooq

Querydsl 과 Jooq 기본 살펴보기

PART2 | Querydsl 활용

Querydsl + Spring Boot 2.5

PART3 | Jooq 활용

Jooq, 정말 쓸만할까요?

PART4 | eager fetch, lazy fetch, N+1 문제

JPA 사용하면서 가장 흔히 겪는 문제 1탄

PART5 | 순환 참조 문제

JPA 사용하면서 가장 흔히 겪는 문제 2탄

복잡한 쿼리의 작성과 응용

4 eager fetch, lazy fetch, N+1 문제

Fetch?

애플리케이션이 DB로부터 데이터를 가져오는 것

DB와 통신하여 데이터를 읽는 것에는 큰 비용이 소모되기 때문에, 똑똑하게 가져오는 전략이 필요

- eager: 프로그램 코드가 쿼리를 날리는 시점에 데이터를 즉시 가져오기
  - ex: select a.id from A a inner join B b on a.b_id = b.id (b를 보지 않았지만 일단 다 가져옴)
- lazy: 가져오려는 데이터를 애플리케이션에서 실제로 접근할 때 가져오기
  - ex: select a.id from A; (select b from B b where b.id = ?)
- lazy 전략은 기본적으로
  - ORM 의 특징이자 기능적 장점
  - 더 빠르고 경제적인 쿼리 (적절히만 사용한다면)
  - 잘못 사용하면 데이터 접근 에러 (ex: LazyInitializationException)

fetch 기본 전략 (default setting)

각 JPA 연관관계 애노테이션은 기본 fetch 전략을 가지고 있다.

기본 세팅의 핵심은 "어느 쪽이 효율적인가"

- @OneToOne: FetchType.EAGER
- @ManyToOne: FetchType.EAGER
- @OneToMany: FetchType.LAZY
- @ManyToMany: FetchType.LAZY

fetch 전략의 설정 (실전)

효율성 - 데이터가 어느 쪽으로 더 자주 사용될 것 같은가 예측

- default 내버려두기: 필요한 시점에 최선의 방식으로 데이터를 가져옴
- LAZY 사용: 연관 관계가 있는 엔티티에서 자식 엔티티만 가져오는 시나리오일 때
  - 프로그래머가 로직 흐름에서 join 을 의식하고 있지 않음
  - LAZY 세팅이 후속 쿼리 발생 방지를 보장하지는 않음
    - ex: 불러들인 자식 엔티티가 서비스 레이어 어딘가에서 결국 부모 엔티티 필드를 건드렸을 경우
- EAGER 사용: 연관 관계가 있는 엔티티에서 무조건 다 가져오는 시나리오일 때
  - 프로그래머가 join 을 사용해야 하는 상황임을 인지하고 있음
  - EAGER 세팅이 join 동작을 보장하지는 않음
    - ex: Spring Data JPA 쿼리 메소드 findAll()
    - JPQL 을 직접 작성해서 join 을 영속성 컨텍스트에 알려줘야 함 (ex: querydsl)

N+1 query problem

4.
eager fetch,
lazy fetch,
N+1 문제

나는 한 번 쿼리를 날렸을 뿐인데, 1 + N개의 쿼리가 더 생겼다

```
Hibernate: select event0_.id as id1_2_, event0_.capacity as capacity2_2_, event0_.created_at as created_3_2_, event0_.current_number_of_people as cur
  event_en5_2_, event0_.event_name as event_na6_2_, event0_.event_start_datetime as event_st7_2_, event0_.event_status as event_st8_2_, event0_.memo a
  modifie10_2_, event0_.place_id as place_i11_2_ from event event0_
Hibernate: select place0_.id as id1_3_0_, place0_.address as address2_3_0_, place0_.capacity as capacity3_3_0_, place0_.created_at as created_4_3_0_,
    as modified6_3_0_, place0_.phone_number as phone_nu7_3_0_, place0_.place_name as place_na8_3_0_, place0_.place_type as place_ty9_3_0_ from place p
2021-10-04 19:05:25.342 TRACE 25877 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [1] as [BIGINT] - [1]
Hibernate: select place0_.id as id1_3_0_, place0_.address as address2_3_0_, place0_.capacity as capacity3_3_0_, place0_.created_at as created_4_3_0_,
    as modified6_3_0_, place0_.phone_number as phone_nu7_3_0_, place0_.place_name as place_na8_3_0_, place0_.place_type as place_ty9_3_0_ from place p
2021-10-04 19:05:25.353 TRACE 25877 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [1] as [BIGINT] - [2]
Hibernate: select place0_.id as id1_3_0_, place0_.address as address2_3_0_, place0_.capacity as capacity3_3_0_, place0_.created_at as created_4_3_0_,
    as modified6_3_0_, place0_.phone_number as phone_nu7_3_0_, place0_.place_name as place_na8_3_0_, place0_.place_type as place_ty9_3_0_ from place p
2021-10-04 19:05:25.356 TRACE 25877 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [1] as [BIGINT] - [3]
```

이벤트를 조회했을 뿐인데

이벤트들이 바라보는 장소들이
하나하나 select 됨

N+1 query problem 해결

3가지 방법

- 똑똑한 lazy
  - 비즈니스 로직을 면밀히 분석하여, 불필요한 연관 관계 테이블 정보를 불러오는 부분을 제거
  - 가장 똑똑하고 효율적인 방법
- eager fetch + join jpql
  - join 쿼리를 직접 작성하는 방법은 다양 (@Query, querydsl, ...)
  - 쿼리 한 번에 오긴 하겠지만, join 쿼리 연산 비용과 네트워크로 전달되는 데이터가 클 수 있음
- 후속 쿼리를 in 으로 묶어주기: N + 1 -> 1 + 1 로 I/O 줄일 수 있음
  - 하이버네이트 프로퍼티: default_batch_fetch_size
  - 스프링 부트에서 쓰는 법: spring.jpa.properties.hibernate.default_batch_fetch_size
  - 100 ~ 1000 사이를 추천
  - 모든 쿼리에 적용되고, 복잡한 도메인에서 join 쿼리를 구성하는 것이 골치아플 때 효율적

Reference

- https://docs.oracle.com/cd/E24290_01/coh.371/e23131/bestpractice.htm#CHDHGEDA
- https://docs.oracle.com/javaee/7/tutorial/persistence-intro001.htm#BNBQA