JPA 07 자바 표준 ORM 표준 JPA 프로그래밍 [JPQL 중급문법]

김영한
2022.08.08

경로 표현식

. (점) 을 찍어서 객체 그래프를 탐색하는 것

```
select m.username -> 상태 필드
  from Member m
    join m.team t    -> 단일 값 연관 필드
    join m.orders o -> 컬렉션 값 연관 필드
where t.name = '팀A'
```

3가지의 경로의 표현식이 있다.

동작하는 방식이 다르고 결과도 다르게 나오게 된다.

상태 필드**? :** 단순히 값을 저장하기 위한 필드

연관 필드**? :** 연관 관계를 위한 필드

- 단일 값 연관 필드 ⇒ 대상이 엔티티 일때 (m.team)
- 컬렉션 값 연관 필드 ⇒ 대상이 컬렉션 (m.orders)

특징

상태 필드 : 경로 탐색의 끝, 탐색 X

단일 값 연관 경로 **:** 묵시적 조인 내부 조인(inner join) 발생, 탐색 O

컬렉션 값 연관 경로 **:** 묵시적 내부 조인 발생, 탐색 X

→ 값을 가져오기 위해 필연적으로 조인이 필요하다.

⇒웬만해서는 묵시적인 내부조인을 발생하지 않게 쿼리를 짜야한다 (튜닝을 해야한다.)

- 명시적인 조인을 통해 별칭을 얻으면 별칭을 통해 탐색이 가능하다.

명시적 조인, 묵시적 조인

join 키워드를 사용하느냐 않느냐의 차이이다

묵시적 조인은 경로 표현식에서 연관 필드를 탐색할때 inner 조인이 계속 발생하게 된다.

가능한 묵시적 조인을 사용하는 것을 피하고 명시적 조인을 통해서 사용하도록 한다.

페치 조인(fetch join)

- SQL 조인 종류X, JPQL에서 성능 최적화를 위해 제공하는 기능
- 연관된 엔티티나 컬렉션을 SQL 한번에 함께 조인

엔티티 페치 조인

회원을 조회하면 서 팀도 함께 조인하고 싶을때

[JPQL]
select m from Member m **join fetch** m.team

[SQL]
SELECT M.*, **T.*** FROM MEMBER M
INNER JOIN TEAM T ON M.TEAM_ID=T.ID

멤버1, 2, 3 이 각각 team1, team1, team2 를 team으로 가지고 있을때,

```
List<Member> resultList = em.createQuery(query, Member.class).getResultList();
for (Member member1 : resultList) {
   System.out.println("member1 = " + member1.getUsername() + ", " +
member1.getTeam().getName());
}
```

fetch- LAZY 지연 로딩이기 때문에 member.team 은 프록시가 들거아 있다.

getTeam().getName()을 호출할 때 마다 Team을 조회하는 쿼리가 계속 나간다.

N+1 쿼리가 나간다. 방법은 페치 조인을 사용한다.

페치 조인을 하게 되면 프록시가 아니라 실제 엔티티가 올라간다.

-주의

일 대 다 관계에서 조인을 하게되면 튜플이 뻥튀기 된다. 조인을 하게 되면 일대 다 이기 때문에 다에 맞춰져서 나오게된다. → **DISTINCT**사용

DISTINCT

중복을 제거하는데 엔티티의 중복도 제거 , 애플리케이션에서 중복 제거해준다 (JPQL)

DISTINCT 쿼리만으로는 안줄여준다.

페치 조인과 일반 조인과의 차이

- 일반 조인 실행시 연관된 엔티티를 함께 조회하지 않는다.
- (영속성 컨텍스트에 포함되지 않는다)
- 페치 조인은 객체 그래프를 SQL 한번에 조회하는 개념 ⇒ 실무에서 사용많다

페치 조인 특징과 한계

- 페치 조인 대상에는 별칭을 줄 수 없다. 가급적 사용X
- 둘 이상의 컬렉션은 페치 조인 할 수 없다. - 데이터가 잘 안맞을 수 있다. 일대 다 처럼 .
- 컬렉션을 페치 조인하게 되면 페이징 API를 사용할 수 없다.(setFirstResult, setMaxResults) → 데이터가 늘어나거나 하지 않는다.
    - 일대일 다대일 같은 단일 값 연관 필드들은 페치 조인해도 페이징이 가능하다.

@BatchSize를 사용하면 한번에 가져오는 수를 정해서 N+1 문제를 줄일 수 있다.

글로벌 옵션으로도 줄 수도 있다. (1000이하의 큰 값으로 주게 되면 한번에 가져온다)

- ☐ 모두 지연 로딩으로 적용하고 페치 조인으로 적용 하면 페치조인으로 성능 최적화를 시킬 수 있다.
- ☐ 모든 것을 페치조인으로 해결할 수 는 없다. → 객체 그래프를 유지할 때 사용하면 효과적

- ☐ 페치 조인보다 일반 조인 **+ DTO** 반환 하는게 효과적인 순간들이 있다**.**

다형성 쿼리

상속받은 상태에서 쿼리로 사용하게 되면 조회 대상을 특정 대상으로 한정을 할 수 있다.

Item - Book, Movie, Album 이 상속 상태일 때

select i from Item i where type(i) IN (Book, Movie)

이러한 다형성을 유지하고 Book, Movie를 가져올 수 있다.

treat

Book에 저자 정보를 가져오고 싶을 때 treat(i as Book).author = 'kim' 다운 캐스팅 필드 접근처럼 사용 가능하다.

엔티티 직접 사용

엔티티 값을 직접사용하면 엔티티의 기본 키 값을 사용하게 된다.

엔티티를 쓴다? → SQL에서는 기본 키값을 사용한다 로 해석이 된다는 것.

파라미터에 **member** 자체를 엔티티를 직접 사용하게 된다면 기본 키 값으로 사용한다고 인식한다**.**

```
String query = "select m from Member m where m = :member ";
Member findMember = em.createQuery(query, Member.class)
       .setParameter("member", member)   → 기본 키 값으로 SQL 생성
       .getSingleResult();
System.out.println("findMember = " + findMember);
```

m.id = :memberId ⇒ setParameter("memberId", member.getId()) 와 같은 결과가 나온다.

연관된 엔티티 사용

외래 키로 연관된 엔티티를 사용하게 되면 외래 키 아이디로 매핑되어 사용되게 된다.

```
String query = "select m from Member m where m.team = :team ";
Member findMember = em.createQuery(query, Member.class)
        .setParameter("team", team2)
        .getSingleResult();
System.out.println("findMember = " + findMember);
```

Named 쿼리

쿼리에 이름을 부여 할 수 있다. 엔티티 선언에서 Named 쿼리를 선언한다.

미리 정의해서 이름을 부여 해주고 사용한다. 정적 쿼리, 어노테이션, XML에 정의

애플리케이션 로딩 시점에 초기화 후 재사용 → 미리 파싱해서 캐싱 해놓고 있다**.**

⇒애플리케이션 로딩 시점에 쿼리를 검증

[로딩 시점에 나는 오류라면 실행 시킬때 에러가 발생되기 때문에 검증이 된다]

벌크 연산

[UPDATE DELETE 문]

예시) 재고가 10개 미만인 모든 상품의 가격을 10% 상승하려면?

JPA 변경감지 기능으로 하려면 SQL 실행이 무수히 많아진다.

조회 후 -> 변경 감지 후 -> 업데이트 실행

해결 ⇒ **.executeUpdate** 를 실행해 준다**.**

```
int resultCount = em.createQuery("update Member m set m.age = 20")
        .executeUpdate();
```

업데이트를 한번에 수행 하는 쿼리 .

벌크 연산 주의점

- 영속성 컨텍스트를 무시하고 데이터 베이스에 직접 쿼리한다. 해결법

- 방법 **1.** 벌크 연산을 먼저 실행
- 방법 **2.** 벌크 연산 수행 후 영속성 컨텍스트 초기화 (clear)