JPA 06 자바 표준 ORM 표준 JPA 프로그래밍 [JPQL 기본문법]

김영한
2022.08.07

JPQL

JPA는 다양한 쿼리 방법을 지원한다.

JPQL, JPA Criteria, QueryDSL, 네이티브 SQL

JDBC 직접 사용, MyBatis, Spring JdbcTemplate 등 사용

- JPA를 사용하면 에니티 객체를 중심으로 개발한다.
- 검색을 할 때에도 테이블이 아닌 엔티티 객체를 대상으로 검색하게 된다. 하지만 모든 DB 데이터를 객체로 변환해서 검색하는것은 성능. 비용면에서 불가능 결국 SQL이 필요하다.
- **JPA**는 **SQL**을 추상화 한 **JPQL**이라는 객체 지향 쿼리 언어를 제공한다**.**

예시**)**

```
List<Member> resultList = em.createQuery(
       "select m From Member m where m.username like '%kim%'",
       Member.class
).getResultList();
```

콘솔에 표시

```
/* select  m FromMember m where m.username like '%kim%' */
```

JPQL

```
select  member0_.MEMBER_ID as member_i1_6_,
            member0_.city as city2_6_,
            member0_.street as street3_6_,
            member0_.zipcode as zipcode4_6_,
            member0_.USERNAME as username5_6_
        from Member member0_
        where member0_.USERNAME like '%kim%'
```

변환된 **SQL** 쿼리**.**

JPA Criteria

동적 쿼리짤때 편하다 잠시 이전 쿼리를 보면

`select  m FromMember m where m.username like '%kim%'`

는 단순 스트링이다. 그러므로 동적으로 변경해주다가 오류나기 좋은 환경이다.

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Member> query = cb.createQuery(Member.class);

Root<Member> m = query.from(Member.class);
CriteriaQuery<Member> cq = query.select(m).where(cb.equal(m.get("username"),
"kim"));
List<Member> resultList1 = em.createQuery(cq).getResultList();
```

자바 코드로서 쿼리를 만들기 때문에 동적쿼리를 만들어 수정하기도 좋다.

→ 다만 알아 보기 힘들고 유지보수가 까다로워서 실무에서 사용이 어렵다**.**

QueryDSL

오픈소스 라이브러리 이다. → 문자가 아닌 자바 코드로 JPQL 작성가능

```
//JPQL
//select m from Member m where m.age > 18
JPAFactoryQuery query = new JPAQueryFactory(em);
QMember m = QMember.member;

List<Member> list =
    query.selectFrom(m)
         .where(m.age.gt(18))
         .orderBy(m.name.desc())
         .fetch();
```

QueryDSL 소개

- JPQL 빌더역할
- 컴파일 시점에 문법 오류를 찾을 수 있음
- 동적 쿼리 작성 편리함
- 단순하고 쉽다
- 실무에 사용하기 좋다.

www.querydsl.com 레퍼런스가 잘 갖춰져있다

JPQL 을 잘 이해하고 사용하면 이해가 쉽다.

네이티브 **SQL**

JPA에서 직접 사용.

```
List<Member> resultList = em.createNativeQuery("select MEMBER_ID, city, street,
zipcode, USERNAME from MEMBER",
       Member.class).getResultList();
```

flush → 커밋할때. query가 날아갈 때 다 동작하게 된다. 쿼리를 날리면 commit이 없어도 저장

JDBC 직접사용 , SpringJdbcTemplate 등

flush → 커밋할때. query가 날아갈 때 다 동작하게 된다. 쿼리를 날리면 commit이 없어도 저장

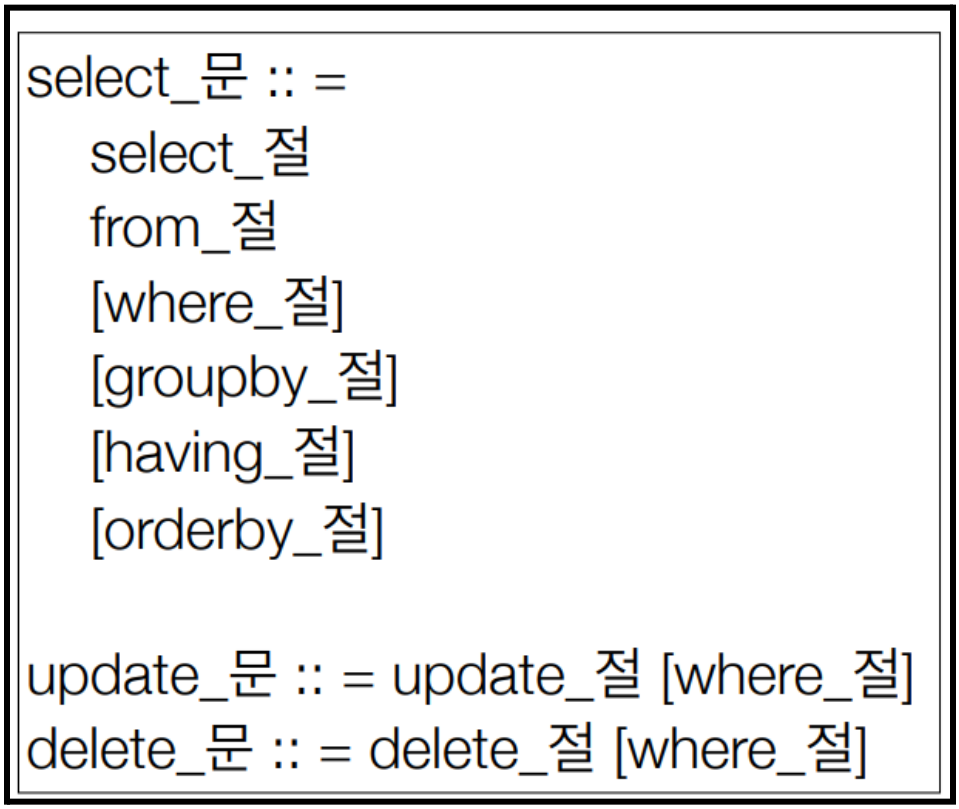
→ 하지만 커넥션을 새로 얻을 때 flush를 해주고 해야한다.

JPQL

JPQL 특징

- 쿼리 언어이다 → 엔티티 객체를 대상으로 쿼리함
- JPQL은 SQL을 추상화한 인터페이스이므로 방언을 옵션으로 갖기 때문에 SQL에 의존하지 않는다.
- JPQL은 결국 SQL로 반환 된다.

JPQL 기본 문법



기본 SQL문과 거의 비슷하다.

- 엔티티와 속성은 대소문자 구분한다. (Member, age 등)
- JQPL 키워드는 대소문자 구분X
- 엔티티의 이름 사용한다. 테이블의 이름이 아니다. (Orders[테이블], Order[엔티티])
- 별칭이 필수로 들어간다 as는 생략이 가능한다.

  ex) select m from Member m (as m)

```
select
    COUNT(m),   //회원수
    SUM(m.age), //나이 합
    AVG(m.age), //평균 나이
    MAX(m.age), //최대 나이
    MIN(m.age)  //최소 나이
from Member m
```

기본으로 사용 가능하다.

- **TypeQuery:** 반환 타입이 명확할 때 사용

- **Query:** 반환 타입이 명확하지 않을 때 사용

```
TypedQuery<Member> result =
       em.createQuery("select m from Member m", Member.class);
TypedQuery<String> result2 =
       em.createQuery("select m.username from Member m", String.class);
Query query = em.createQuery("select m.age, m.username from Member m");
```

```
List<Member> resultList = result.getResultList();
Member singleResult = result.getSingleResult();
```

결과 반환할때 문제가 생길수 있다

getResultList 는 결과가 없어도 그냥 빈 리스트를 반환한다.

getSingleResult 는 결과가 무조건 하나가 있을때 사용해야한다. 아니라면 무조건 예외를 발생시킨다.

[Spring Data JPA]에서는 Optional 를 반환하거나 null을 반환해준다. 내부에서 예외를 잡아내게 구현 되어 있다.

파라미터 바인딩 예시

```
Member member = new Member();
member.setUsername("member1");
member.setAge(10);
em.persist(member);

Member singleResult = em.createQuery("select m from Member m where m.username =
:username", Member.class)
       .setParameter("username", "member1")
       .getSingleResult();
System.out.println("singleResult.getUsername() = " + singleResult.getUsername());
```

위치 기준도 사용가능하지만 버그우려가 크기 때문에

이름 기준 바인딩을 사용하는것이 좋다.

프로젝션

- SELECT 절에 조회할 대상을 지정하는 것
- 프로젝션 대상 : 엔티티, 임베디드 타입, 스칼라 타입 (기본 데이터 타입)

```
select m ⇒ 엔티티 프로젝션
select m.team  ⇒  엔티티 프로젝션
select m.address  ⇒ 임베디드 (값 ) 타입 프로젝션
select m.username, m.age   ⇒ 스칼라 타입 프로젝션
```

- DISTINCT로 중복 제거 가능

엔티티 프로젝션

- 가져온 결과가 영속성 컨텍스트에 관리가 된다.

임베디드 타입 프로젝션

- 소속을 명확히 알려야 한다. (엔티티로 부터 시작해야 한다)

스칼라 타입 프로젝션

- 기본 SQL 쓰듯이 그냥 사용하면 된다.

서로 다른 타입이라면 ?

1. Query 타입으로 반환
2. Object[] 타입으로 반환
3. new 명령어로 조회 → 단순 값을 DTO로 바로 조회

```
List<MemberDTO> result = em.createQuery("select new jpql.MemberDTO(m.username,
m.age) from Member m",MemberDTO.class)
       .getResultList();
```

패키지가 길어지면 다 적어야 하지만 queryDSL이면 임포트해서 다 사용가능하다. 순서와 타입이 일치하는 생성자가 필요하다.

페이징 **API**

- JPA는 페이징을 다음 두 API로 추상화
- **setFirstResult** 조회 시작 위치 (0 부터 시작)
- **setMaxResults** 조회할 데이터 수

조인

엔티티 중심으로 조인 쿼리를 날린다.

내부 조인

- 조건이 같은 걸 모두 가져온다

외부 조인

- 조건이 같지 않은것도 것도 모두 가져온다.

세타 조인

- 연관관계가 없는 엔티티를 비교할때

ON 절

1. 조인 대상 필터링
2. 연관 관계 없는 엔티티 외부 조인 가능

예) 회원과 팀을 조인하면서 팀이름이 A인 팀만 조인 할때

```
select m, t from Member m LEFT JOIN m.team t on t.name = 'A'
```

조인하기전에 조건을 줄 수 있다**.** 연관관계가 없어도 조인이 가능

서브쿼리

- SQL 서브쿼리와 똑같다 . 괄호를 열어서 서브쿼리를 적어주고 JPQL을 써주면 SQL로 변경해준다. 서브 쿼리

서브 쿼리 지원함수

- EXISTS 서브쿼리에 결과가 존재하면 참
- ALL, ANY, SOME, IN

서브 쿼리 - 예제

- 팀A 소속인 회원
  select m from Member m
  where **exists** (select t from m.team t where t.name = '팀A')

- 전체 상품 각각의 재고보다 주문량이 많은 주문들
  select o from Order o
  where o.orderAmount > **ALL** (select p.stockAmount from Product p)

- 어떤 팀이든 팀에 소속된 회원
  select m from Member m
  where m.team = **ANY** (select t from Team t)

JPA 서브 쿼리 한계점

- JPA는 WHERE, HAVING 절에서만 서브 쿼리 사용 가능
- SELECT 절도 Hibernate에서는 가능하다.
- FROM 절의 서브 쿼리는 현재 JPQL에서 불가능 ⇒ 조인으로 풀 수 있으면 풀어서 해결

JPQL 타입 표현

문자 : 'HELLO' , 'SHE"s'

숫자 : 10L(Long) , 10D(Double), 10F(Float)

Boolean : TRUE, FALSE

ENUM : jpabook.MembmerType.Admin(패키지 명 포함)

엔티티 타입 : TYPE(m) = Member (상속 관계에서 사용)

JPQL 기타

- SQL과 문법이 같은 식
- EXISTS, IN
- AND, OR, NOT
- =, >, >= , <, <=, <>
- BETWEEN, LIKE, IS NULL

기본 문법들 전부 사용 가능하다.

조건식 **(CASE** 식 **)**

기본 **CASE** 식

```
String query =
        "select " +
        "case when m.age <= 10 then '학생요금' " +
                "when m.age >= 60 then '경로요금' " +
                "else '일반요금' end " +
        "from Member m";
```

단순 **CASE** 식 **: when** 에 값**.**

COALESCE : 하나씩 조회해서 **null**이 아니면 반환

```
String query =
        "select " +
                "coalesce(m.username, '이름 없는 회원') " +
        "from Member m";
```

NULLIF : 두 값이 같으면 **NULL** 반환 **(**관리자 숨기기**)**

```
String query =
        "select " +
                "nullif(m.username, 'teamA') " +
        "from Member m";
```

→teamA 인 경우 null로 표시

JPQL 기본 함수

CONCAT, SUBSTRING, TRIM, LOWER, UPPER, LENGTH, LOCATE, ABS, SQRT, MOD, SIZE, INDEX

사용자 정의 함수 호출

- 데이터베이스 방언에 추가해 놔야 사용가능하다.

방언을 상속받은 새로운 방언 클래스를 생성후에 사용자 정의 함수를 정의해주고 persistence에 방언을 변경해준다 . 그 후에 문법에 맞게 사용해준다.