객체지향 쿼리 언어

하이버네이트가 100% 지원하는 것은 아님
JPQL로 거의 되지만 네이티브를 사용해야 될 경우가 있음

JPQL: Java Persistence Query Language (철저하게 잘하자 !!)
 문제는 검색 쿼리 (검색 조건이 포함된 SQL 필요)
 엔티티 객체를 대상으로 쿼리 -> 결국에는 SQL로 변환됨(feat Dialect)
 객체 지행 SQL, 특정 데이터베이스 SQL에 의존하지 않음
 동적쿼리를 만들기 어려움
 위치기반 파라미터 매핑은 웬만하면 사용하지 말자(좋은 대안으로 해결하자)
 프로젝션
  대상: 엔티티, 임베디드 타입, 스칼라 타입(숫자, 문자등 기본 데이터 타입), DISTINCT로 중복 제거 가능
 페이징
    setFirstResult(0) - 0부터 시작
  데이터베이스에 상관없이 동작
 조인
  내부 조인
  외부 조인
  세타 조인(막조인)
  기타 - ON 절
      JPA 2.1부터 지원
   조인 대상 필터링 가능, 연관관계 없는 엔티티 외부 조인(하이버네이트 5.1 부터 가능)
 서브 쿼리
    exists, ALL, ANY 사용 가능, (NOT) IN 사용가능
    JPA WHERE, HAVING절에서만 가능, SELECT 절은 하이버네이트에서 지원
      FROM 절의 서브 쿼리는 JPQL에서 불가능(조인으로 풀수 있으면 풀어서 해결)
   아니면 Native Query 나 Application에서 처리 , 아니면 2번 쿼리해야 됨
  JPQL 타입 표현 및 기타식
  문자: 'HELLO', 'She''s'
  숫자: 10L, 10D, 10F
    Boolean: TRUE, FALSES
    ENUM: FQCN
  엔티티 타입: TYPE(m) = Member
  기타: EXISTS, IN, AND, OR, NOT, =, >, >=, <, <=, <>, BETWEEN, LIKE, IS NULL
 조건식
  기본 CASE 식 : 조건 추가
  단순 CASE 식 : 정확하게 일치
    COALESCE: 내용이 없으면 리턴
    NULLIF: 조건이 맞으면 널 리턴
 기본 함수
    CONCAT, SUBSTRING, TRIM, LOWER, UPPER, LENGTH, LOCATE, ABS, SQRT, MOD, SIZE, INDEX(JPA용도)
  사용자 정의 함수(Dialect에 등록하여 사용하거나 등록된 함수 사용 가능), select function('group_concat', i.name) from Item i ...
 경로 표현식
  상태 필드: 단순히 값을 저장, m.username, 경로 탐색의 끝, 더 이상 탐색 불가
  연관 필드:
   단일 값 연관 필드: @ManyToOne, @OneToOne, 대상이 엔티티, 묵시적 내부 조인(INNER JOIN)발생, 추가 탐색 가능
   컬렉션 값 연관 필드: @OneToMany, @ManyToMany, 대상이 컬렉션, m.orders, 묵시적 내

부 조인 발생, 탐색 불가(size 정도는 가능)
FROM절에서 명시적 조인을 통해 별칭을 얻으면 별칭을 통해 탐색 가능
실무: 가급적 묵시적 조인 대신에 항상 명시적 조인 사용(join 명시)
묵시적 조인은 조인이 일어나는 상황을 한눈에 파악하기 어려움, 성능에 지대한 영향(JOIN 튜닝 해야 됨)

페치 조인(fetch join)
JPQL에서 성능 최적화를 위해 제공하는 기능(연관된 엔티티들을 SQL 한 번으로 조회)
연관된 엔티티나 컬렉션을 SQL 한 번에 함께 조회하는 기능
join fetch 명령어 사용
연관 관계가 지연이라도 fetch join이 우선 함(글로벌 로딩 전략보다 우선 - 실무에서 글로벌 로딩 전략은 모두 지연 로딩)
최적화가 필요한 곳은 페치 조인 적용
컬렉션 페치 조인은 중복이 발생할 수 있음
JPA DISTINCT를 사용하여 중복 제거(SQL DISTINCT는 안되지만 JPA DISTINCT 는 추가로 애플리케이션에서 중복 제거 시도)
즉시 로딩(연관된 엔티티도 함께 조회) - 대부분의 N+1 문제 해결
한계
페치 조인 대상에는 별칭이 불가능(없는게 관례), 하이버네이트는 가능
둘 이상의 컬렉션은 페치 조인 할 수 없다. (정합성 문제)
컬렉션을 페치 조인하면 페이징 API는 상요할 수 없다.
일대일, 다애일 같은 단일 값 연관 필드들은 페치 조인해도 페이징 가능, 방향을 뒤집어서 해결
하이버네이트는 경고 로그를 남기고 메모리에서 페이징(매우 위험)
배치 사이즈로 해결할 수 있음(N+1 해결) - 실무에서 배치사이즈를 글로벌로 1000이사 적절하게 사용

다향성 쿼리
TREAT(JAP 2.1), 다운 캐스팅 가능

엔티티 직접 사용
JPQL에서 엔티티를 집접 사용하면 해당 엔티티의 기본 키 값을 사용하는 효과가 있음

Named 쿼리
미리 이름을 부여하여 사용하는 JPQL
애노테이션으로 @Entity아래에 정의 (엔티티 지저분해짐), XML로 정의 가능
애플리케이션 로딩 시점에 초기화 후 재사용 가능
애클리케이션 로딩 시점에 SQL로 변환하여 파싱&캐시하고 있음, 검증도 함께 함 - 엄청난 장점
JPA에서 인터페이스 함수 위에서 이름 없는 Named쿼리로 사용 가능
벌크 연산
JPA는 실시간에 최적화 됨
UPDATE, DELETE 지원
executeUpdate()의 결과는 영향 받은 개수
INSERT(insert into .. select, 하이버네이트 지원)
벌크 연산은 영속성 컨텍스트를 무시하고 데이터베이스에 직접 쿼리
벌크 연산 수행 후 영속성 컨텐스트 초기화하면 됨

JPA Criteria
JPQL 제너레이터
JAVA 표준이지만 쉽지 않음(복잡함, 실용성이 부족함).
SQL 문법 오류가 줄어 듦(자바 코드로 구현)
동적 쿼리를 만들기 쉬움, SQL스럽지 않음
엔티티와 속성은 대소문자 구문
JPQL 키워드는 대소문자 구분하지 않음
엔티티 이름 사용, 테이블 이름이 아님(Member)

```
 별칭은 필수(m) (as는 생략 가능) - 가급적 사용하자

QueryDSL
  JPQL 제너레이터
 단순하고 읽기 쉬움
 문법 오류를 찾을 수 있음, 오타가 없음
 동적 쿼리를 작성하기 쉬움
 실무 사용 권장

네이티브 SQL
 웬만하면 지양하자

JDBC API 직접 사용, MyBatis, SpringJdbcTemplage 함께 사용
 적절하게 em.persist()를 사용해야 됨

// flush -> commit, query
createNativeQuery, createQuery의 경우 flush와 함께 동작
```

jpa, jqpl

From:
http://rwiki.repia.com/ - **문제를 잘 정의하는 것은 문제를 절반 해결한 것이다. - 2023.12**

Permanent link:
http://rwiki.repia.com/doku.php?id=wiki:spring:jpa:jpql&rev=1657103604

Last update: **2022/07/06 19:33**