Persistence Queries

- JPA(Java Persistence API)는 자바 애플리케이션에서 관계형 데이터베이스를 사용하는 방식을 표준화한 API이다.

- JPA를 사용함으로써 개발자는 객체 지향적으로 데이터베이스를 다룰 수 있으며 데이터베이스 코드를 보다 쉽게 유지보수할 수 있다.

- JPA에서는 엔티티 매니저를 통한 메서드 기반의 데이터 접근 방식 외에도 직접 쿼리를 작성하여 데이터를 조작하고 조회하는 다양한 기술을 제공한다.

1. Native SQL

Native SQL은 Java Persistence API (JPA)에서 제공하는 기능 중 하나로 개발자가 데이터베이스에 직접 실행할 순수 SQL 쿼리를 작성할 수 있게 해준다. 이를 통해 JPA의 JPQL(Java Persistence Query Language)이나 Criteria API가 제공하는 추상화를 벗어나 특정 데이터베이스에 특화된 기능이나 구문을 사용할 수 있다. 하지만 쿼리의 데이터베이스 이식성이 떨어질수 있으며 JPA의 다른 부분과의 일관성을 잃을 수 있으며 객체-관계 매핑의 이점을 일부 포기할 수 있다. Native SQL은 JPA의 일부이며, EntityManager 인터페이스를 통해 사용할 수 있다.

데이터베이스 특화 기능 사용: 데이터베이스가 제공하는 고유한 기능이나 최적화된 쿼리, 고급 SQL 구문(예: 특정 윈도우 함수, 재귀 쿼리 등)을 사용할 수 있다.

유연성: Native SQL을 사용하면, JPA가 제공하는 추상화 계층을 벗어나 직접 SQL을 작성할 수 있어 데이터베이스 작업에 대한 더 높은 수준의 제어가 가능하다.

이식성 저하: Native SQL 쿼리는 특정 데이터베이스에 종속적일 수 있기 때문에 다른 종류의 데이터베이스로 애플리케이션을 이식할 때 문제가 발생할 수 있다.

[예시]

```
@Query(nativeQuery = true , value="SELECT * FROM MEMBER WHERE MEMBER_ID = :id")
```

2. JPQL (Java Persistence Query Language)

JPQL(Java Persistence Query Language)은 JPA(Java Persistence API)를 사용하여 데이터베이스를 쿼리하는 언어이다. JPQL은 SQL과 유사하지만 데이터베이스 테이블 대신 엔티티 객체 모델을 대상으로 쿼리를 작성한다. 이 접근 방식은 객체 지향 프로그래밍과 잘 어울리며, 애플리케이션 코드를 데이터베이스 구조로부터 분리시켜 준다.

플랫폼 독립성: JPQL은 데이터베이스에 독립적이다. 이는 JPQL 쿼리를 작성할 때 특정 데이터베이스 SQL 구문을 사용하지 않아도 된다는 것을 의미한다. 대신, JPA 구현체가 JPQL을 데이터베이스에 특화된 SQL로 변환한다.

엔티티 중심 쿼리: JPQL은 엔티티 클래스와 그 필드를 기반으로 쿼리를 작성한다. 쿼리는 데이터베이스 테이블이 아닌, 엔티티 객체에 대해 실행된다.

타입 안전성: JPQL은 컴파일 타임에 쿼리 구문을 검사할 수 있도록 해주어, 실행 시간에 발생할 수 있는 오류를 줄여준다.

[예시]

```
@Query(value="select m from Member m where memberId = :memberId")
```

3. QueryDSL

QueryDSL은 Java 언어로 안전하고, 유지보수가 용이하며, 타입 안전한 쿼리를 작성할 수 있도록 도와주는 프레임워크이다. 이는 SQL, JPA, JDO, 그리고 컬렉션을 위한 통합된 쿼리 언어를 제공한다. 복잡하고 동적인 쿼리를 타입 안전성을 보장하면서 쉽게 작성할 수 있으며 동적 쿼리를 안전하고 가독성 높게 작성할 수 있다. QueryDSL의 장점은 복잡한 쿼리 작업을 단순화하고, 컴파일 시점에 쿼리의 올바름을 검증하여 런타임 오류를 줄이는 데 있다.

타입 안전성: QueryDSL은 타입 안전한 쿼리를 작성할 수 있게 해준다. 이는 쿼리의 구문과 타입을 컴파일 시점에 검사하여 오류를 사전에 발견할 수 있게 해주며, 이는 개발자가 더 안정적인 코드를 작성할 수 있다.

유동적인 쿼리 작성: 동적 쿼리를 쉽게 작성할 수 있게 해준다. 조건문, 반복문 등을 사용하여 복잡한 쿼리를 유연하게 구성할 수 있다.

플랫폼 독립성: QueryDSL은 JPA, JDO, SQL, MongoDB, Lucene, Hibernate Search 등 다양한 저장소 기술에 대한 지원을 제공한다. 이는 하나의 쿼리 언어로 다양한 데이터 소스를 다룰 수 있게 해준다.

코드 자동 생성: 엔티티나 데이터베이스 스키마로부터 쿼리 타입을 자동으로 생성할 수 있다. 이는 반복적인 작업을 줄이고, 데이터 모델의 변경 사항을 쿼리 클래스에 쉽게 반영할 수 있게 해준다.

[예시]
```
JPAQueryFactory queryFactory = new JPAQueryFactory(em);
QMember qMember = QMember.member;

queryFactory.selectFrom(qMember)
.where(qMember.memberId.eq("qwer1234"))
.fetch();
```

4. Criteria API

Criteria API는 Java Persistence API (JPA)의 일부로서, 프로그래밍 방식으로 타입 안전한 쿼리를 생성할 수 있는 API이다. 이 API를 사용하면, JPQL(Java Persistence Query Language)을 문자열로 작성하는 대신에, 자바 코드를 사용하여 데이터베이스 쿼리를 구성하고 실행할 수 있다. 이러한 접근 방식은 쿼리의 구조를 동적으로 변경하는 경우에 유용하며, 컴파일 시점에 타입 체크를 할 수 있어 오타나 잘못된 타입 사용으로 인한 런타임 오류를 줄일 수 있다.

타입 안전성(Type Safety): Criteria API를 사용하면, 엔티티 클래스와 그 속성을 직접 참조하여 쿼리를 구성할 수 있다. 이는 쿼리 문자열에서 발생할 수 있는 오타나 타입 불일치 문제를 컴파일 시점에 발견할 수 있게 해준다.

동적 쿼리 생성: 쿼리의 조건이 실행 시간에 결정되는 경우, Criteria API는 쿼리를 동적으로 구성하고 수정하는 데 매우 유용하다. 이를 통해 복잡한 사용자 입력에 따라 다르게 반응하는 쿼리를 쉽게 작성할 수 있다.

API 기반의 쿼리: Criteria API는 JPQL이나 SQL 문자열 대신 자바 API를 사용하여 쿼리를 구성한다. 이는 쿼리 구성 과정을 보다 명확하게 하고, 자바 코드의 리팩토링과 통합을 용이하게 한다.

[예시]
```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Member> cq = cb.createQuery(Member.class);
Root<Member> member = cq.from(Member.class);
cq.select(member).where(cb.equal(member.get("memberId"), "qwer1234"));
em.createQuery(cq).getResultList()
```