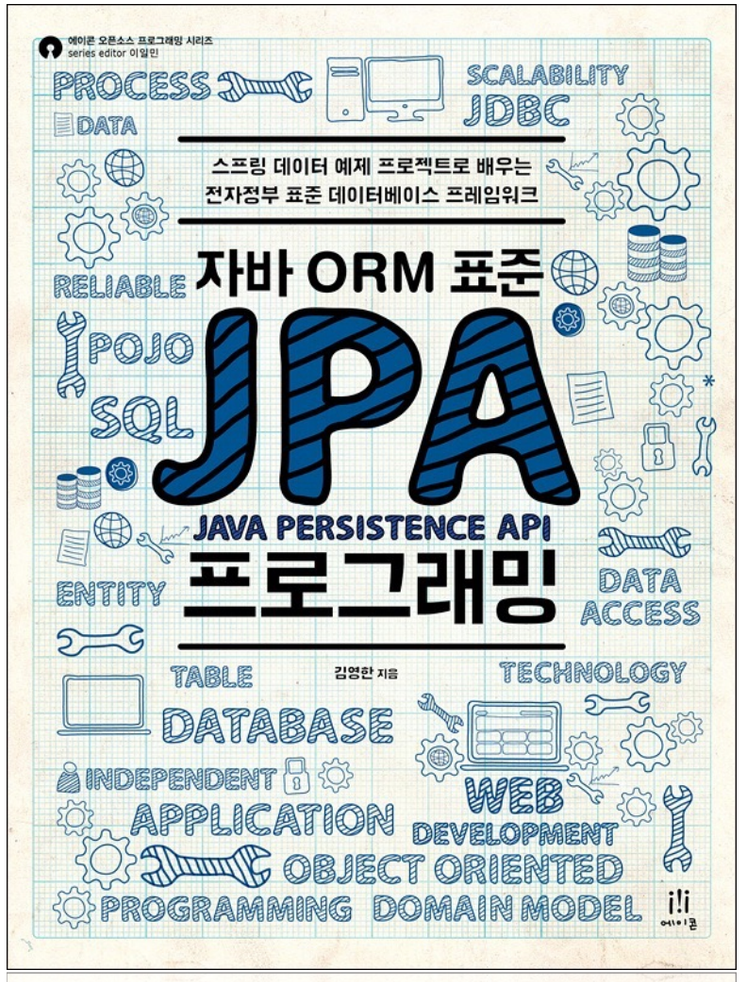
객체 지향 쿼리

ex07, ex08

김영한

SI, J2EE 강사, DAUM, SK 플래닛

저서: 자바 ORM 표준 **JPA** 프로그래밍



목차

- JPA와 객체지향 쿼리
- QueryDSL

JPA와 객체지향 쿼리

JPA는 다양한 쿼리 방법을 지원

- **JPQL**
- JPA Criteria
- **QueryDSL**
- 네이티브 SQL
- JDBC API 직접 사용, MyBatis, SpringJdbcTemplate 함께 사용

JPQL 소개

- 가장 단순한 조회 방법
  - EntityManager.find()
  - 객체 그래프 탐색(a.getB().getC())
- 나이가 18살 이상인 회원을 모두 검색하고 싶다면?

JPQL

- JPA를 사용하면 엔티티 객체를 중심으로 개발
- 문제는 검색 쿼리
- 검색을 할 때도 테이블이 아닌 엔티티 객체를 대상으로 검색해야 함.
- 모든 DB 데이터를 객체로 변환해서 검색하는 것은 불가능
- 애플리케이션이 필요한 데이터만 DB에서 불러오려면 결국 검색 조건이 포함된 SQL이 필요

JPQL

- JPA는 SQL을 추상화한 JPQL이라는 객체 지향 쿼리 언어를 제공

- SQL과 문법 유사, SELECT, FROM, WHERE, GROUP BY, HAVING, JOIN 지원

- JPQL은 엔티티 객체를 대상으로 쿼리

- SQL은 데이터베이스 테이블을 대상으로 쿼리

JPQL

```
//검색
String jpql = "select m From Member m where m.name like ‘%hello%'";
List<Member> result = em.createQuery(jpql, Member.class)
        .getResultList();
```

JPQL

- 테이블이 아닌 객체를 대상으로 검색하는 객체 지향 쿼리다.
- SQL을 추상화해서 특정 데이터베이스 SQL에 의존하지 않는다
- JPQL을 한마디로 정의하면 객체 지향 SQL

JPQL과 실행된 SQL

```
//검색
String jpql = "select m from Member m where m.age > 18";

List<Member> result = em.createQuery(jpql, Member.class)
        .getResultList();
```

```
실행된 SQL
    select
        m.id as id,
        m.age as age,
        m.USERNAME as USERNAME,
        m.TEAM_ID as TEAM_ID
    from
        Member m
    where
        m.age>18
```

JPQL 문법

```
select_문 :: =
    select_절
    from_절
    [where_절]
    [groupby_절]
    [having_절]
    [orderby_절]

update_문 :: = update_절 [where_절]
delete_문 :: = delete_절 [where_절]
```

JPQL 문법

- select m from Member m where m.age > 18
- 엔티티와 속성은 대소문자 구분(Member, username)
- JPQL 키워드는 대소문자 구분 안함(SELECT, FROM, where)
- 엔티티 이름을 사용, 테이블 이름이 아님(Member)
- 별칭은 필수(m)

결과 조회 API

- query.getResultList(): 결과가 하나 이상, 리스트 반환
- query.getSingleResult(): 결과가 정확히 하나, 단일 객체 반환 (정확히 하나가 아니면 예외 발생)

파라미터 바인딩 - 이름 기준, 위치 기준

SELECT m FROM Member m where m.username=**:username**

query.setParameter("**username**", usernameParam);

SELECT m FROM Member m where m.username=**?1**

query.setParameter(**1**, usernameParam);

프로젝션

- SELECT **m** FROM Member m -> 엔티티 프로젝션
- SELECT **m.team** FROM Member m -> 엔티티 프로젝션
- SELECT **username, age** FROM Member m -> 단순 값 프로젝션
- **new** 명령어: 단순 값을 DTO로 바로 조회
  SELECT **new** jpabook.jpql.UserDTO(m.username, m.age)
  FROM Member m
- DISTINCT는 중복 제거

페이징 API

- JPA는 페이징을 다음 두 API로 추상화
- setFirstResult(int startPosition) : 조회 시작 위치 (0부터 시작)
- setMaxResults(int maxResult) : 조회할 데이터 수

페이징 API 예시

```
//페이징 쿼리
String jpql = "select m from Member m order by m.name desc";
List<Member> resultList = em.createQuery(jpql, Member.class)
      .setFirstResult(10)
      .setMaxResults(20)
      .getResultList();
```

페이징 API - MySQL 방언

```
SELECT
     M.ID AS ID,
     M.AGE AS AGE,
     M.TEAM_ID AS TEAM_ID,
     M.NAME AS NAME
FROM
     MEMBER M
ORDER BY
     M.NAME DESC LIMIT ?, ?
```

페이징 API - Oracle 방언

```
SELECT * FROM
     ( SELECT ROW_.*, ROWNUM ROWNUM_
     FROM
          ( SELECT
               M.ID AS ID,
               M.AGE AS AGE,
               M.TEAM_ID AS TEAM_ID,
               M.NAME AS NAME
          FROM MEMBER M
          ORDER BY M.NAME
          ) ROW_
     WHERE ROWNUM <= ?
     )
WHERE ROWNUM_ > ?
```

집합과 정렬

```
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
```

집합과 정렬

- GROUP BY, HAVING
- ORDER BY

조인

- 내부 조인: SELECT m FROM Member m [INNER] JOIN m.team t
- 외부 조인: SELECT m FROM Member m LEFT [OUTER] JOIN m.team t
- 세타 조인: select count(m) from Member m, Team t where m.username = t.name
- **한계: 세타 조인시은 내부 조인만 사용할 수 있다.**

페치 조인

- 엔티티 객체 그래프를 한번에 조회하는 방법
- 별칭을 사용할 수 없다.
- JPQL: select m from Member m join **fetch** m.team
- SQL: SELECT M.*, T.* FROM MEMBER T INNER JOIN TEAM T ON M.TEAM_ID=T.ID

페치 조인 예시

```
String jpql = "select m from Member m join fetch m.team";

List<Member> members = em.createQuery(jpql, Member.class)
                         .getResultList();

for (Member member : members) {
    //페치 조인으로 회원과 팀을 함께 조회해서 지연 로딩 발생 안함
    System.out.println("username = " + member.getUsername() + ", " +
                       "teamname = " + member.getTeam().name());
}
```

JPQL 기타

- 서브 쿼리 지원
- EXISTS, IN
- BETWEEN, LIKE, IS NULL

JPQL 기본 함수

- CONCAT
- SUBSTRING
- TRIM
- LOWER, UPPER
- LENGTH
- LOCATE
- ABS, SQRT, MOD
- SIZE, INDEX(JPA 용도)

CASE 식

기본 CASE 식

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

단순 CASE 식

```
select
     case t.name
          when '팀A' then '인센티브110%'
          when '팀B' then '인센티브120%'
          else '인센티브105%'
     end
from Team t
```

CASE 식

- COALESCE: 하나씩 조회해서 null이 아니면 반환
- NULLIF: 두 값이 같으면 null 반환, 다르면 첫번째 값 반환

```
select coalesce(m.username,'이름 없는 회원') from Member m
```

```
select NULLIF(m.username, '관리자') from Member m
```

사용자 정의 함수 호출

- 하이버네이트는 사용전 방언에 추가해야 한다.

```
select function('group_concat', i.name) from Item i
```

Named 쿼리 - 정적 쿼리

- 미리 정의해서 이름을 부여해두고 사용하는 JPQL
- 어노테이션, XML에 정의
- 애플리케이션 로딩 시점에 초기화 후 재사용
- 애플리케이션 로딩 시점에 쿼리를 검증

Named 쿼리 - 어노테이션

```
@Entity
@NamedQuery(
        name = "Member.findByUsername",
        query="select m from Member m where m.username = :username")
public class Member {
    ...
}
```

```
List<Member> resultList =
  em.createNamedQuery("Member.findByUsername", Member.class)
        .setParameter("username", "회원1")
        .getResultList();
```

Named 쿼리 - XML에 정의

[META-INF/persistence.xml]

```
<persistence-unit name="jpabook" >
    <mapping-file>META-INF/ormMember.xml</mapping-file>
```

[META-INF/ormMember.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm" version="2.1">

    <named-query name="Member.findByUsername">
        <query><![CDATA[
            select m
            from Member m
            where m.username = :username
        ]]></query>
    </named-query>

    <named-query name="Member.count">
        <query>select count(m) from Member m</query>
    </named-query>

</entity-mappings>
```

Named 쿼리 환경에 따른 설정

- XML이 항상 우선권을 가진다.
- 애플리케이션 운영 환경에 따라 다른 XML을 배포할 수 있다.

JPQL 실습

- ex07
- 단순 쿼리
- 조인
- 페치 조인
- 페이징 API

QueryDSL

QueryDSL 소개

- JPQL을 코드로 작성할 수 있도록 도와주는 빌더 API
- JPA 크리테리아에 비해서 편리하고 실용적임
- 오픈소스
- 한국어 번역(최범균): www.querydsl.com/static/querydsl/3.6.3/reference/ko-KR/html_single/

JPQL의 문제점

- SQL, JPQL은 문자, Type-check 불가능
- 실행해 보기 전까지 작동여부 확인 불가

QueryDSL 장점

- **문자가 아닌 코드로 작성**
- **컴파일 시점에 오류 발견**
- **코드 자동완성**
- **단순함, 쉬움: 코드 모양이 JPQL과 거의 흡사.**
- **동적 쿼리**

작동 방식

QueryDSL - 메이븐 - 라이브러리

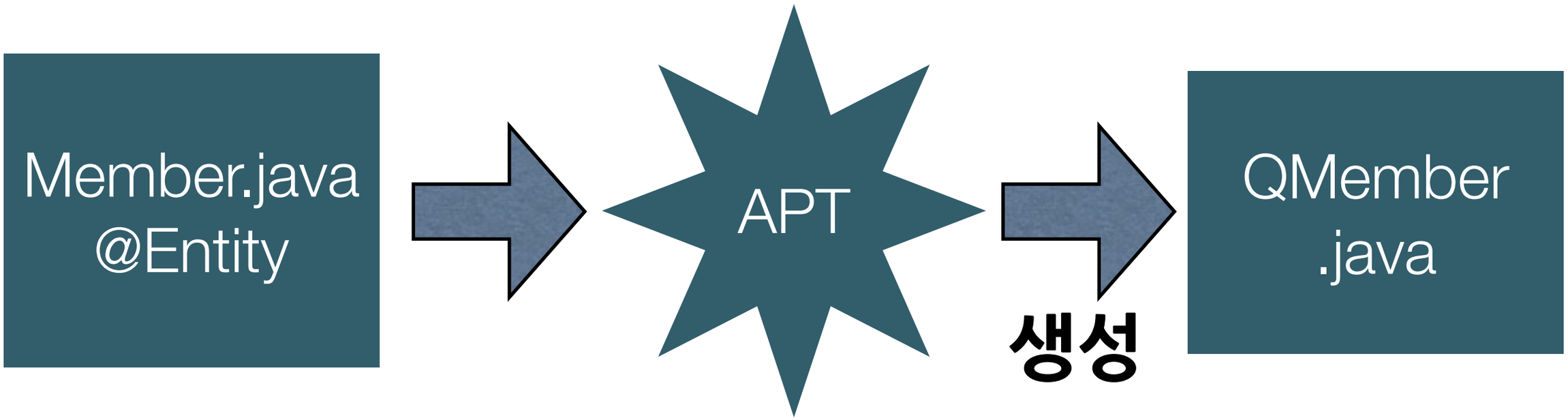
```
<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-jpa</artifactId>
  <version>3.6.3</version>
</dependency>

<dependency>
  <groupId>com.mysema.querydsl</groupId>
  <artifactId>querydsl-apt</artifactId>
  <version>3.6.3</version>
  <scope>provided</scope>
</dependency>
```

QueryDSL - 메이븐 - 플러그인

```
<build>
    <plugins>
        <plugin>
            <groupId>com.mysema.maven</groupId>
            <artifactId>apt-maven-plugin</artifactId>
            <version>1.1.3</version>
            <executions>
                <execution>
                    <goals>
                        <goal>process</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>target/generated-sources/java</outputDirectory>
                        <processor>com.mysema.query.apt.jpa.JPAAnnotationProcessor</processor>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

QueryDSL - 쿼리타입 생성



자동 생성된 쿼리 타입(Q)

```
@Generated("com.mysema.query.codegen.EntitySerializer")
public class QMember extends EntityPathBase<Member> {

  private static final long serialVersionUID = 1928823744L;

  private static final PathInits INITS = PathInits.DIRECT2;

  public static final QMember member = new QMember("member1");

  public final NumberPath<Integer> age = createNumber("age", Integer.class);

  public final NumberPath<Long> id = createNumber("id", Long.class);

  public final StringPath name = createString("name");

  public final QTeam team;
  …
```

QueryDSL 사용

```
//JPQL
select m from Member m where m.age > 18
```

```
JPAQuery query = new JPAQuery(em);

QMember m = QMember.member;
List<Member> list =
        query.from(m)
       .where(m.age.gt(18))
       .list(m);
```

QueryDSL - 조인

```
JPAQuery query = new JPAQuery(em);

QMember m = QMember.member;
QTeam t = QTeam.team;
```

```
List<Member> list = query.from(m)
        .join(m.team, t)
        .where(t.name.eq("teamA"))
        .list(m);
```

QueryDSL - 페이징 API

```
JPAQuery query = new JPAQuery(em);
QMember m = QMember.member;

List<Member> list = query.from(m)
.orderBy(m.age.desc())
.offset(10)
.limit(20)
.list(m);
```

QueryDSL - 동적 쿼리

```
String name = "member";
int age = 9;

JPAQuery query = new JPAQuery(em);
QMember m = QMember.member;

BooleanBuilder builder = new BooleanBuilder();
if (name != null) {
  builder.and(m.name.contains(name));
}
if (age != 0) {
  builder.and(m.age.gt(age);
}

List<Member> list = query.from(m)
      .where(builder)
      .list(m);
```

기능 정리

- from
- innerJoin, join, leftJoin, fullJoin, on
- where (and, or, allOf, anyOf)
- groupBy
- having
- orderBy (desc, asc)
- limit, offset, restrict(limit + offset) (Paging)

기능 정리

- list
- listResults (list + Paging Info(totalCount))
- iterate
- count
- singleResult, uniqueResult

DTO로 반환

```
QItem item = QItem.item;

List<ItemDTO> result =
  query.from(item).list(Projections.bean(ItemDTO.class,
                            item.name.as("username"), item.price));
```

실습 ex08

- ex07 JPQL -> QueryDSL로 변환
- 동적쿼리 추가