5. 실무 활용 - 순수 JPA와 Querydsl

#1.인강/JPA활용편/querydsl/강의

- /순수 JPA 리포지토리와 Querydsl
- /동적 쿼리와 성능 최적화 조회 - Builder 사용
- /동적 쿼리와 성능 최적화 조회 - Where절 파라미터 사용
- /조회 API 컨트롤러 개발

순수 JPA 리포지토리와 Querydsl

순수 JPA 리포지토리

```
package study.querydsl.repository;

import com.querydsl.core.BooleanBuilder;
import com.querydsl.core.types.dsl.BooleanExpression;
import com.querydsl.jpa.impl.JPAQueryFactory;
import org.springframework.stereotype.Repository;
import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.dto.MemberTeamDto;
import study.querydsl.dto.QMemberTeamDto;
import study.querydsl.entity.Member;

import javax.persistence.EntityManager;
import java.util.List;
import java.util.Optional;

import static org.springframework.util.StringUtils.hasText;
import static org.springframework.util.StringUtils.isEmpty;
import static study.querydsl.entity.QMember.member;
import static study.querydsl.entity.QTeam.team;

@Repository
public class MemberJpaRepository {

    private final EntityManager em;
    private final JPAQueryFactory queryFactory;

    public MemberJpaRepository(EntityManager em) {
```

```
        this.em = em;
        this.queryFactory = new JPAQueryFactory(em);
    }

    public void save(Member member) {
        em.persist(member);
    }

    public Optional<Member> findById(Long id) {
        Member findMember = em.find(Member.class, id);
        return Optional.ofNullable(findMember);
    }

    public List<Member> findAll() {
        return em.createQuery("select m from Member m", Member.class)
                .getResultList();
    }

    public List<Member> findByUsername(String username) {
        return em.createQuery("select m from Member m where m.username
= :username", Member.class)
                .setParameter("username", username)
                .getResultList();
    }

}
```

순수 JPA 리포지토리 테스트

```
package study.querydsl.repository;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.transaction.annotation.Transactional;
import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.dto.MemberTeamDto;
import study.querydsl.entity.Member;
import study.querydsl.entity.Team;

import javax.persistence.EntityManager;
import java.util.List;
```

```java
import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest
@Transactional
class MemberJpaRepositoryTest {

    @Autowired
    EntityManager em;
    @Autowired
    MemberJpaRepository memberJpaRepository;

    @Test
    public void basicTest() {
        Member member = new Member("member1", 10);
        memberJpaRepository.save(member);

        Member findMember = memberJpaRepository.findById(member.getId()).get();
        assertThat(findMember).isEqualTo(member);

        List<Member> result1 = memberJpaRepository.findAll();
        assertThat(result1).containsExactly(member);

        List<Member> result2 = memberJpaRepository.findByUsername("member1");
        assertThat(result2).containsExactly(member);
    }
```

Querydsl 사용

순수 JPA 리포지토리 - Querydsl 추가

```java
    public List<Member> findAll_Querydsl() {
        return queryFactory
                .selectFrom(member).fetch();
    }

    public List<Member> findByUsername_Querydsl(String username) {
        return queryFactory
                .selectFrom(member)
                .where(member.username.eq(username))
                .fetch();
    }
```

Querydsl 테스트 추가

```
@Test
public void basicQuerydslTest() {
    Member member = new Member("member1", 10);
    memberJpaRepository.save(member);

    Member findMember = memberJpaRepository.findById(member.getId()).get();
    assertThat(findMember).isEqualTo(member);

    List<Member> result1 = memberJpaRepository.findAll_Querydsl();
    assertThat(result1).containsExactly(member);

    List<Member> result2 =
memberJpaRepository.findByUsername_Querydsl("member1");
    assertThat(result2).containsExactly(member);
}
```

JPAQueryFactory 스프링 빈 등록

다음과 같이 `JPAQueryFactory`를 스프링 빈으로 등록해서 주입받아 사용해도 된다.

```
@Bean
JPAQueryFactory jpaQueryFactory(EntityManager em) {
    return new JPAQueryFactory(em);
}
```

> 참고: 동시성 문제는 걱정하지 않아도 된다. 왜냐하면 여기서 스프링이 주입해주는 엔티티 매니저는 실제 동작 시점에 진짜 엔티티 매니저를 찾아주는 프록시용 가짜 엔티티 매니저이다. 이 가짜 엔티티 매니저는 실제 사용 시점에 트랜잭션 단위로 실제 엔티티 매니저(영속성 컨텍스트)를 할당해준다.
> 더 자세한 내용은 자바 ORM 표준 JPA 책 13.1 트랜잭션 범위의 영속성 컨텍스트를 참고하자.

동적 쿼리와 성능 최적화 조회 - Builder 사용

MemberTeamDto - 조회 최적화용 DTO 추가

```
package study.querydsl.dto;
```

```
import com.querydsl.core.annotations.QueryProjection;
import lombok.Data;

@Data
public class MemberTeamDto {
    private Long memberId;
    private String username;
    private int age;
    private Long teamId;
    private String teamName;

    @QueryProjection
    public MemberTeamDto(Long memberId, String username, int age, Long teamId,
String teamName) {
        this.memberId = memberId;
        this.username = username;
        this.age = age;
        this.teamId = teamId;
        this.teamName = teamName;
    }
}
```

- `@QueryProjection`을 추가했다. `QMemberTeamDto`를 생성하기 위해 `./gradlew compileQuerydsl`을 한번 실행하자.

> 참고: `@QueryProjection`을 사용하면 해당 DTO가 Querydsl을 의존하게 된다. 이런 의존이 싫으면, 해당 에노테이션을 제거하고, `Projection.bean(), fields(), constructor()`을 사용하면 된다.

회원 검색 조건

```
package study.querydsl.dto;

import lombok.Data;

@Data
public class MemberSearchCondition {
    //회원명, 팀명, 나이(ageGoe, ageLoe)

    private String username;
    private String teamName;
    private Integer ageGoe;
    private Integer ageLoe;
}
```

- 이름이 너무 길면 `MemberCond` 등으로 줄여 사용해도 된다.

동적쿼리 - Builder 사용

Builder를 사용한 예제

```
//Builder 사용
//회원명, 팀명, 나이(ageGoe, ageLoe)
public List<MemberTeamDto> searchByBuilder(MemberSearchCondition condition) {

    BooleanBuilder builder = new BooleanBuilder();
    if (hasText(condition.getUsername())) {
        builder.and(member.username.eq(condition.getUsername()));
    }
    if (hasText(condition.getTeamName())) {
        builder.and(team.name.eq(condition.getTeamName()));
    }
    if (condition.getAgeGoe() != null) {
        builder.and(member.age.goe(condition.getAgeGoe()));
    }
    if (condition.getAgeLoe() != null) {
        builder.and(member.age.loe(condition.getAgeLoe()));
    }

    return queryFactory
            .select(new QMemberTeamDto(
                    member.id,
                    member.username,
                    member.age,
                    team.id,
                    team.name))
            .from(member)
            .leftJoin(member.team, team)
            .where(builder)
            .fetch();
}
```

> **오류 정정**
> 강의 영상에서는 `member.id.as("memberId")` 라고 적었는데, `QMemberTeamDto` 는 생성자를 사용하기 때문에 필드 이름을 맞추지 않아도 된다. 따라서 `member.id` 만 적으면 된다.

조회 예제 테스트

```
@Test
public void searchTest() {
    Team teamA = new Team("teamA");
    Team teamB = new Team("teamB");
    em.persist(teamA);
    em.persist(teamB);

    Member member1 = new Member("member1", 10, teamA);
    Member member2 = new Member("member2", 20, teamA);
    Member member3 = new Member("member3", 30, teamB);
    Member member4 = new Member("member4", 40, teamB);

    em.persist(member1);
    em.persist(member2);
    em.persist(member3);
    em.persist(member4);

    MemberSearchCondition condition = new MemberSearchCondition();
    condition.setAgeGoe(35);
    condition.setAgeLoe(40);
    condition.setTeamName("teamB");

    List<MemberTeamDto> result = memberJpaRepository.searchByBuilder(condition);

    assertThat(result).extracting("username").containsExactly("member4");
}
```

동적 쿼리와 성능 최적화 조회 - Where절 파라미터 사용

Where절에 파라미터를 사용한 예제

```
//회원명, 팀명, 나이(ageGoe, ageLoe)
public List<MemberTeamDto> search(MemberSearchCondition condition) {
    return queryFactory
```

```
                .select(new QMemberTeamDto(
                        member.id,
                        member.username,
                        member.age,
                        team.id,
                        team.name))
                .from(member)
                .leftJoin(member.team, team)
                .where(usernameEq(condition.getUsername()),
                        teamNameEq(condition.getTeamName()),
                        ageGoe(condition.getAgeGoe()),
                        ageLoe(condition.getAgeLoe()))
                .fetch();
    }

    private BooleanExpression usernameEq(String username) {
        return isEmpty(username) ? null : member.username.eq(username);
    }

    private BooleanExpression teamNameEq(String teamName) {
        return isEmpty(teamName) ? null : team.name.eq(teamName);
    }

    private BooleanExpression ageGoe(Integer ageGoe) {
        return ageGoe == null ? null : member.age.goe(ageGoe);
    }

    private BooleanExpression ageLoe(Integer ageLoe) {
        return ageLoe == null ? null : member.age.loe(ageLoe);
    }
```

참고: where 절에 파라미터 방식을 사용하면 조건 재사용 가능

```
//where 파라미터 방식은 이런식으로 재사용이 가능하다.
public List<Member> findMember(MemberSearchCondition condition) {
    return queryFactory
            .selectFrom(member)
            .leftJoin(member.team, team)
            .where(usernameEq(condition.getUsername()),
                    teamNameEq(condition.getTeamName()),
                    ageGoe(condition.getAgeGoe()),
```

```
                    ageLoe(condition.getAgeLoe()))
            .fetch();
    }
```

조회 API 컨트롤러 개발

편리한 데이터 확인을 위해 샘플 데이터를 추가하자.

샘플 데이터 추가가 테스트 케이스 실행에 영향을 주지 않도록 다음과 같이 프로파일을 설정하자

프로파일 설정

`src/main/resources/application.yml`

```
spring:
  profiles:
    active: local
```

테스트는 기존 application.yml을 복사해서 다음 경로로 복사하고, 프로파일을 test로 수정하자

`src/test/resources/application.yml`

```
spring:
  profiles:
    active: test
```

이렇게 분리하면 main 소스코드와 테스트 소스 코드 실행시 프로파일을 분리할 수 있다.

샘플 데이터 추가

```
package study.querydsl;

import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import study.querydsl.entity.Member;
import study.querydsl.entity.Team;

import javax.annotation.PostConstruct;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
```

```java
@Profile("local")
@Component
@RequiredArgsConstructor
public class InitMember {

    private final InitMemberService initMemberService;

    @PostConstruct
    public void init() {
        initMemberService.init();
    }

    @Component
    static class InitMemberService {

        @PersistenceContext
        EntityManager em;

        @Transactional
        public void init() {
            Team teamA = new Team("teamA");
            Team teamB = new Team("teamB");
            em.persist(teamA);
            em.persist(teamB);

            for (int i = 0; i < 100; i++) {
                Team selectedTeam = i % 2 == 0 ? teamA : teamB;
                em.persist(new Member("member" + i, i, selectedTeam));
            }

        }
    }
}
```

조회 컨트롤러

```java
package study.querydsl.controller;
```

```java
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.dto.MemberTeamDto;
import study.querydsl.repository.MemberJpaRepository;

import java.util.List;

@RestController
@RequiredArgsConstructor
public class MemberController {

    private final MemberJpaRepository memberJpaRepository;

    @GetMapping("/v1/members")
    public List<MemberTeamDto> searchMemberV1(MemberSearchCondition condition) {
        return memberJpaRepository.search(condition);
    }

}
```

- 예제 실행(postman)
- `http://localhost:8080/v1/members?teamName=teamB&ageGoe=31&ageLoe=35`