

## 6. 실무 활용 - 스프링 데이터 JPA와 Querydsl

#1.인강/JPA활용편/querydsl/강의

- /스프링 데이터 JPA 리포지토리로 변경
- /사용자 정의 리포지토리
- /스프링 데이터 페이징 활용1 - Querydsl 페이징 연동
- /스프링 데이터 페이징 활용2 - CountQuery 최적화
- /스프링 데이터 페이징 활용3 - 컨트롤러 개발

### 스프링 데이터 JPA 리포지토리로 변경

스프링 데이터 JPA - MemberRepository 생성

```
package study.querydsl.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import study.querydsl.entity.Member;

import java.util.List;

public interface MemberRepository extends JpaRepository<Member, Long> {

    List<Member> findByUsername(String username);
}
```

스프링 데이터 JPA 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;

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

@SpringBootTest
@Transactional
class MemberRepositoryTest {

    @Autowired
    EntityManager em;
    @Autowired
    MemberRepository memberRepository;

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

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

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

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

```

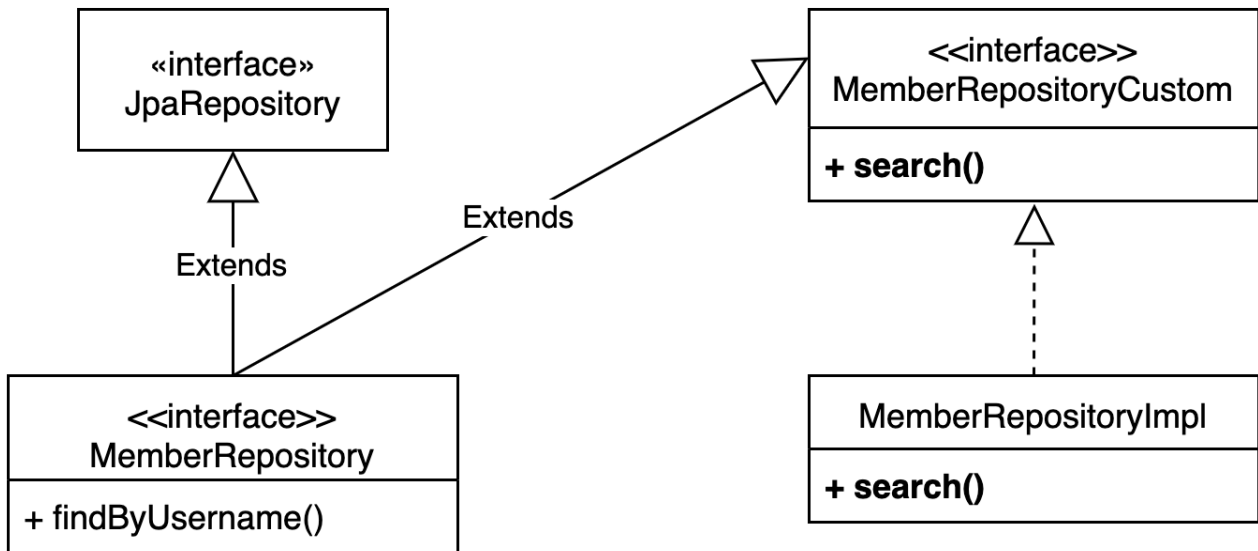
- Querydsl 전용 기능인 회원 search를 작성할 수 없다. → 사용자 정의 리포지토리 필요

## 사용자 정의 리포지토리

### 사용자 정의 리포지토리 사용법

1. 사용자 정의 인터페이스 작성
2. 사용자 정의 인터페이스 구현
3. 스프링 데이터 리포지토리에 사용자 정의 인터페이스 상속

## 사용자 정의 리포지토리 구성



### 1. 사용자 정의 인터페이스 작성

```
package study.querydsl.repository;

import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.dto.MemberTeamDto;

import java.util.List;

public interface MemberRepositoryCustom {

    List<MemberTeamDto> search(MemberSearchCondition condition);
}
```

### 2. 사용자 정의 인터페이스 구현

```
package study.querydsl.repository;

import com.querydsl.core.types.dsl.BooleanExpression;
import com.querydsl.jpa.impl.JPAQueryFactory;
import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.dto.MemberTeamDto;
import study.querydsl.dto.QMemberTeamDto;

import javax.persistence.EntityManager;
```

```

import java.util.List;

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

public class MemberRepositoryImpl implements MemberRepositoryCustom {

    private final JPAQueryFactory queryFactory;

    public MemberRepositoryImpl(EntityManager em) {
        this.queryFactory = new JPAQueryFactory(em);
    }

    @Override
    //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);
    }
}

```

### 3. 스프링 데이터 리포지토리에 사용자 정의 인터페이스 상속

```

package study.querydsl.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import study.querydsl.entity.Member;

import java.util.List;

public interface MemberRepository extends JpaRepository<Member, Long>,
    MemberRepositoryCustom {

    List<Member> findByUsername(String username);
}

```

### 커스텀 리포지토리 동작 테스트 추가

```

@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 = memberRepository.search(condition);

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

```

## 스프링 데이터 페이징 활용1 - Querydsl 페이징 연동

- 스프링 데이터의 Page, Pageable을 활용해보자.
- 전체 카운트를 한번에 조회하는 단순한 방법
- 데이터 내용과 전체 카운트를 별도로 조회하는 방법

### 사용자 정의 인터페이스에 페이징 2가지 추가

```

package study.querydsl.repository;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.dto.MemberTeamDto;

import java.util.List;

public interface MemberRepositoryCustom {

    List<MemberTeamDto> search(MemberSearchCondition condition);

    Page<MemberTeamDto> searchPageSimple(MemberSearchCondition condition,
    Pageable pageable);
    Page<MemberTeamDto> searchPageComplex(MemberSearchCondition condition,
    Pageable pageable);
}

```

## 전체 카운트를 한번에 조회하는 단순한 방법

### searchPageSimple(), fetchResults() 사용

```
/**
 * 단순한 페이징, fetchResults() 사용
 */
@Override
public Page<MemberTeamDto> searchPageSimple(MemberSearchCondition condition,
Pageable pageable) {

    QueryResults<MemberTeamDto> results = 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()))
        .offset(pageable.getOffset())
        .limit(pageable.getPageSize())
        .fetchResults();

    List<MemberTeamDto> content = results.getResults();
    long total = results.getTotal();

    return new PageImpl<>(content, pageable, total);
}
```

- Querydsl이 제공하는 fetchResults() 를 사용하면 내용과 전체 카운트를 한번에 조회할 수 있다.(실제 쿼리는 2번 호출)
- fetchResult() 는 카운트 쿼리 실행시 필요없는 order by 는 제거한다.

## 데이터 내용과 전체 카운트를 별도로 조회하는 방법

### searchPageComplex()

```
/**
```

```

* 복잡한 페이징
* 데이터 조회 쿼리와, 전체 카운트 쿼리를 분리
*/
@Override
public Page<MemberTeamDto> searchPageComplex(MemberSearchCondition condition,
Pageable pageable) {

    List<MemberTeamDto> content = 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()))
        .offset(pageable.getOffset())
        .limit(pageable.getPageSize())
        .fetch();

    long total = queryFactory
        .select(member)
        .from(member)
        .leftJoin(member.team, team)
        .where(usernameEq(condition.getUsername()),
            teamNameEq(condition.getTeamName()),
            ageGoe(condition.getAgeGoe()),
            ageLoe(condition.getAgeLoe()))
        .fetchCount();

    return new PageImpl<>(content, pageable, total);
}

```

- 전체 카운트를 조회 하는 방법을 최적화 할 수 있으면 이렇게 분리하면 된다. (예를 들어서 전체 카운트를 조회할 때 조인 쿼리를 줄일 수 있다면 상당한 효과가 있다.)
- 코드를 리팩토링해서 내용 쿼리와 전체 카운트 쿼리를 읽기 좋게 분리하면 좋다.



## 스프링 데이터 페이징 활용2 - CountQuery 최적화

PageableExecutionUtils.getPage()로 최적화

```
JPAQuery<Member> countQuery = queryFactory
    .select(member)
    .from(member)
    .leftJoin(member.team, team)
    .where(usernameEq(condition.getUsername()),
            teamNameEq(condition.getTeamName()),
            ageGoe(condition.getAgeGoe()),
            ageLoe(condition.getAgeLoe()));

//      return new PageImpl<>(content, pageable, total);
    return PageableExecutionUtils.getPage(content, pageable,
countQuery::fetchCount);
```

- 스프링 데이터 라이브러리가 제공
- count 쿼리가 생략 가능한 경우 생략해서 처리
  - 페이지 시작이면서 콘텐츠 사이즈가 페이지 사이즈보다 작을 때
  - 마지막 페이지 일 때 (offset + 콘텐츠 사이즈를 더해서 전체 사이즈 구함, 더 정확히는 마지막 페이지이면  
서 콘텐츠 사이즈가 페이지 사이즈보다 작을 때)

## 스프링 데이터 페이징 활용3 - 컨트롤러 개발

실제 컨트롤러

```
package study.querydsl.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
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 study.querydsl.repository.MemberRepository;

import java.util.List;
```

```

@RestController
@RequiredArgsConstructor
public class MemberController {

    private final MemberJpaRepository memberJpaRepository;
    private final MemberRepository memberRepository;

    @GetMapping("/v1/members")
    public List<MemberTeamDto> searchMemberV1(MemberSearchCondition condition) {
        return memberJpaRepository.search(condition);
    }
    @GetMapping("/v2/members")
    public Page<MemberTeamDto> searchMemberV2(MemberSearchCondition condition,
        Pageable pageable) {
        return memberRepository.searchPageSimple(condition, pageable);
    }
    @GetMapping("/v3/members")
    public Page<MemberTeamDto> searchMemberV3(MemberSearchCondition condition,
        Pageable pageable) {
        return memberRepository.searchPageComplex(condition, pageable);
    }
}

```

- <http://localhost:8080/v2/members?size=5&page=2>

## 스프링 데이터 정렬(Sort)

스프링 데이터 JPA는 자신의 정렬(Sort)을 Querydsl의 정렬(OrderSpecifier)로 편리하게 변경하는 기능을 제공한다. 이 부분은 뒤에 스프링 데이터 JPA가 제공하는 Querydsl 기능에서 살펴보겠다.

스프링 데이터의 정렬을 Querydsl의 정렬로 직접 전환하는 방법은 다음 코드를 참고하자.

### 스프링 데이터 Sort를 Querydsl의 OrderSpecifier로 변환

```

JPAQuery<Member> query = queryFactory
    .selectFrom(member);

for (Sort.Order o : pageable.getSort()) {
    PathBuilder pathBuilder = new PathBuilder(member.getType(),
        member.getMetadata());
    query.orderBy(new OrderSpecifier(o.isAscending() ? Order.ASC : Order.DESC,
        pathBuilder.get(o.getProperty())));
}

```

```
List<Member> result = query.fetch();
```

참고: 정렬(Sort)은 조건이 조금만 복잡해져도 Pageable의 Sort 기능을 사용하기 어렵다. 루트 엔티티 범위를 넘어가는 동적 정렬 기능이 필요하다면 스프링 데이터 페이징이 제공하는 Sort를 사용하기 보다는 파라미터를 받아서 직접 처리하는 것을 권장한다.