

## 7. 스프링 데이터 JPA가 제공하는 Querydsl 기능

#1.인강/JPA활용편/querydsl/강의

- /인터페이스 지원 - QuerydslPredicateExecutor
- /Querydsl Web 지원
- /리포지토리 지원 - QuerydslRepositorySupport
- /Querydsl 지원 클래스 직접 만들기

여기서 소개하는 기능은 제약이 커서 복잡한 실무 환경에서 사용하기에는 많이 부족하다. 그래도 스프링 데이터에서 제공하는 기능이므로 간단히 소개하고, 왜 부족한지 설명하겠다.

### 인터페이스 지원 - QuerydslPredicateExecutor

- 공식 URL: <https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#core.extensions.querydsl>

#### QuerydslPredicateExecutor 인터페이스

```
public interface QuerydslPredicateExecutor<T> {  
  
    Optional<T> findById(Predicate predicate);  
    Iterable<T> findAll(Predicate predicate);  
    long count(Predicate predicate);  
    boolean exists(Predicate predicate);  
  
    // ... more functionality omitted.  
}
```

#### 리포지토리에 적용

```
interface MemberRepository extends JpaRepository<User, Long>,  
    QuerydslPredicateExecutor<User> {  
}
```

```
Iterable result = memberRepository.findAll(  
    member.age.between(10, 40)  
    .and(member.username.eq("member1"))  
);
```

### 한계점

- 조인X (목시적 조인은 가능하지만 left join이 불가능하다.)
- 클라이언트가 Querydsl에 의존해야 한다. 서비스 클래스가 Querydsl이라는 구현 기술에 의존해야 한다.
- 복잡한 실무환경에서 사용하기에는 한계가 명확하다.

참고: `QuerydslPredicateExecutor` 는 Pagable, Sort를 모두 지원하고 정상 동작한다.

## Querydsl Web 지원

- 공식 URL: <https://docs.spring.io/spring-data/jpa/docs/2.2.3.RELEASE/reference/html/#core.web.type-safe>

### 한계점

- 단순한 조건만 가능
- 조건을 커스텀하는 기능이 복잡하고 명시적이지 않음
- 컨트롤러가 Querydsl에 의존
- 복잡한 실무환경에서 사용하기에는 한계가 명확

## 리포지토리 지원 - QuerydslRepositorySupport

QuerydslRepositorySupport

### 장점

- `getQuerydsl().applyPagination()` 스프링 데이터가 제공하는 페이징을 Querydsl로 편리하게 변환 가능(단! Sort는 오류발생)
- `from()` 으로 시작 가능(최근에는 QueryFactory를 사용해서 `select()` 로 시작하는 것이 더 명시적)
- EntityManager 제공

## 한계

- Querydsl 3.x 버전을 대상으로 만듦
- Querydsl 4.x에 나온 JPAQueryFactory로 시작할 수 없음
  - select로 시작할 수 없음 (from으로 시작해야함)
- QueryFactory를 제공하지 않음
- 스프링 데이터 Sort 기능이 정상 동작하지 않음

## Querydsl 지원 클래스 직접 만들기

스프링 데이터가 제공하는 QuerydslRepositorySupport가 지닌 한계를 극복하기 위해 직접 Querydsl 지원 클래스를 만들어보자.

## 장점

- 스프링 데이터가 제공하는 페이징을 편리하게 변환
- 페이징과 카운트 쿼리 분리 가능
- 스프링 데이터 Sort 지원
- select(), selectFrom()으로 시작 가능
- EntityManager, QueryFactory 제공

## Querydsl4RepositorySupport

```
package study.querydsl.repository.support;

import com.querydsl.core.types.EntityPath;
import com.querydsl.core.types.Expression;
import com.querydsl.core.types.dsl.PathBuilder;
import com.querydsl.jpa.impl.JPAQuery;
import com.querydsl.jpa.impl.JPAQueryFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.support.JpaEntityInformation;
import org.springframework.data.jpa.repository.support.JpaEntityInformationSupport;
```

```

import org.springframework.data.jpa.repository.support.Querydsl;
import org.springframework.data.querydsl.SimpleEntityPathResolver;
import org.springframework.data.repository.support.PageableExecutionUtils;
import org.springframework.stereotype.Repository;
import org.springframework.util.Assert;

import javax.annotation.PostConstruct;
import javax.persistence.EntityManager;
import java.util.List;
import java.util.function.Function;

/**
 * Querydsl 4.x 버전에 맞춘 Querydsl 지원 라이브러리
 *
 * @author Younghan Kim
 * @see
 org.springframework.data.jpa.repository.support.QuerydslRepositorySupport
 */
@Repository
public abstract class Querydsl4RepositorySupport {

    private final Class domainClass;
    private Querydsl querydsl;
    private EntityManager entityManager;
    private JPAQueryFactory queryFactory;

    public Querydsl4RepositorySupport(Class<?> domainClass) {
        Assert.notNull(domainClass, "Domain class must not be null!");
        this.domainClass = domainClass;
    }

    @Autowired
    public void setEntityManager(EntityManager entityManager) {
        Assert.notNull(entityManager, "EntityManager must not be null!");

        JpaEntityInformation entityInformation =
JpaEntityInformationSupport.getEntityInformation(domainClass, entityManager);
        SimpleEntityPathResolver resolver = SimpleEntityPathResolver.INSTANCE;
        EntityPath path = resolver.createPath(entityInformation.getJavaType());
        this.entityManager = entityManager;
        this.querydsl = new Querydsl(entityManager, new
PathBuilder<>(path.getType(), path.getMetadata()));
        this.queryFactory = new JPAQueryFactory(entityManager);
    }

```

```

    }

    @PostConstruct
    public void validate() {
        Assert.notNull(entityManager, "EntityManager must not be null!");
        Assert.notNull(querydsl, "Querydsl must not be null!");
        Assert.notNull(queryFactory, "QueryFactory must not be null!");
    }

    protected JPAQueryFactory getQueryFactory() {
        return queryFactory;
    }

    protected Querydsl getQuerydsl() {
        return querydsl;
    }

    protected EntityManager getEntityManager() {
        return entityManager;
    }

    protected <T> JPAQuery<T> select(Expression<T> expr) {
        return getQueryFactory().select(expr);
    }

    protected <T> JPAQuery<T> selectFrom(EntityPath<T> from) {
        return getQueryFactory().selectFrom(from);
    }

    protected <T> Page<T> applyPagination(Pageable pageable,
Function<JPAQueryFactory, JPAQuery> contentQuery) {
        JPAQuery jpaQuery = contentQuery.apply(getQueryFactory());
        List<T> content = getQuerydsl().applyPagination(pageable,
jpaQuery).fetch();
        return PageableExecutionUtils.getPage(content, pageable,
jpaQuery::fetchCount);
    }

    protected <T> Page<T> applyPagination(Pageable pageable,
Function<JPAQueryFactory, JPAQuery> contentQuery, Function<JPAQueryFactory,
JPAQuery> countQuery) {
        JPAQuery jpaContentQuery = contentQuery.apply(getQueryFactory());
        List<T> content = getQuerydsl().applyPagination(pageable,

```

```

jpaContentQuery).fetch();

        JPAQuery countResult = countQuery.apply(getQueryFactory());
        return PageableExecutionUtils.getPage(content, pageable,
countResult::fetchCount);
    }

}

```

### Querydsl4RepositorySupport 사용 코드

```

package study.querydsl.repository;

import com.querydsl.core.types.dsl.BooleanExpression;
import com.querydsl.jpa.impl.JPAQuery;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.support.PageableExecutionUtils;
import org.springframework.stereotype.Repository;
import study.querydsl.dto.MemberSearchCondition;
import study.querydsl.entity.Member;
import study.querydsl.repository.support.Querydsl4RepositorySupport;

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;

@Repository
public class MemberTestRepository extends Querydsl4RepositorySupport {

    public MemberTestRepository() {
        super(Member.class);
    }

    public List<Member> basicSelect() {
        return select(member)
            .from(member)
            .fetch();
    }
}

```

```

public List<Member> basicSelectFrom() {
    return selectFrom(member)
        .fetch();
}

public Page<Member> searchPageByApplyPage(MemberSearchCondition condition,
Pageable pageable) {

    JPAQuery<Member> query = selectFrom(member)
        .leftJoin(member.team, team)
        .where(usernameEq(condition.getUsername()),
            teamNameEq(condition.getTeamName()),
            ageGoe(condition.getAgeGoe()),
            ageLoe(condition.getAgeLoe()));

    List<Member> content = getQuerydsl().applyPagination(pageable, query)
        .fetch();

    return PageableExecutionUtils.getPage(content, pageable,
query::fetchCount);
}

public Page<Member> applyPagination(MemberSearchCondition condition,
Pageable pageable) {
    return applyPagination(pageable, contentQuery -> contentQuery
        .selectFrom(member)
        .leftJoin(member.team, team)
        .where(usernameEq(condition.getUsername()),
            teamNameEq(condition.getTeamName()),
            ageGoe(condition.getAgeGoe()),
            ageLoe(condition.getAgeLoe())));
}

public Page<Member> applyPagination2(MemberSearchCondition condition,
Pageable pageable) {
    return applyPagination(pageable, contentQuery -> contentQuery
        .selectFrom(member)
        .leftJoin(member.team, team)
        .where(usernameEq(condition.getUsername()),
            teamNameEq(condition.getTeamName()),
            ageGoe(condition.getAgeGoe()),
            ageLoe(condition.getAgeLoe())),
        countQuery -> countQuery

```

```
        .selectFrom(member)
        .leftJoin(member.team, team)
        .where(usernameEq(condition.getUsername()),
                teamNameEq(condition.getTeamName()),
                ageGoe(condition.getAgeGoe()),
                ageLoe(condition.getAgeLoe()))
    );
}
```

```
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);
}
```

```
}
```