querydsl-김영한

인프런 - 실전! Querydsl

2025-07-05 - Querydsl 설정

1. 학습 주제

- Spring Boot 프로젝트에 QueryDSL을 설정하고 적용하는 방법 학습

2. 주요 개념 요약

항목	설명
Q 클래스	컴파일 시 생성되는 QueryDSL용 타입 클래스 (예: `QHello`)
build.gradle clean 설정	`clean` 작업 시 `/src/main/generated` 디렉터리를 자동 삭제
Q타입 생성 시점	`build` 또는 `compileJava` 단계에서 자동 생성됨
Q타입 생성 확인	`study/querydsl/entity/QHello.java` 파일이 생성되었는지 확인

3. 실습 코드

3-1. build.gradle` (Spring Boot 3.x 이상 기준)

1. dependency 추가

```
// QueryDSL 라이브러리 의존성 추가
implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'

// QueryDSL Annotation Processor 설정
annotationProcessor "com.querydsl:querydsl-
```

```
apt:${dependencyManagement.importedProperties['querydsl.version']}:jakart
a"
annotationProcessor "jakarta.annotation:jakarta.annotation-api"
annotationProcessor "jakarta.persistence:jakarta.persistence-api"
```

2. clean 작업 시 Q클래스 디렉터리 삭제 설정

```
clean {
    delete file('src/main/generated')
}
```

---

4. 마무리

- QueryDSL 적용을 위해서는 annotation processor 설정과 빌드 경로 정리가 필수
- Q타입 클래스가 정상적으로 생성되지 않으면 IDE에서 인식 오류가 발생할 수 있으므로 build 작업 확인 필요
- /src/main/generated 경로는 .gitignore에 등록하여 소스 관리 대상에서 제외하는 것이 일반적

2025-07-16 - 기본 문법

1. 학습 주제

- QueryDSL 기본 문법 학습
  - 검색 조건
  - 정렬 (Sort)
  - 페이징
  - 집계 (Group By)
  - 조인 (Join, On, Fetch Join)
  - 서브쿼리
  - Case 문
  - 상수 및 문자열 결합

2. 주요 개념 요약

항목	설명
JPAQueryFactory	`EntityManager`로부터 생성되며, 스레드 간 공유해도 무방함 (트랜잭션마다 별도의 영속성 컨텍스트를 제공하므로 안전함)

항목	설명
Q 클래스	별칭이 없을 경우 클래스명과 동일한 기본 별칭 사용
fetch()	결과를 `List`로 반환, 결과가 없으면 빈 리스트 반환
fetchOne()	결과가 하나일 때 반환, 없으면 `null`, 두 개 이상이면 `NonUniqueResultException` 발생
fetchResults()	페이징 정보 포함 (`total count` 쿼리 자동 실행)
fetchCount()	`count` 쿼리 실행 (성능 문제 있음)
fetchFirst()	`limit(1).fetchOne()`과 동일
selectFrom()	`select()`와 `from()`을 결합한 축약 표현
where(...)	파라미터 여러 개 전달 시 `AND` 조건으로 연결됨
Tuple	집계 함수나 복합 선택 시 사용하는 반환 타입
join(...)	대상 엔티티, 별칭을 지정하여 조인, 자동으로 ID 기준 연결
getPersistenceUnitUtil().isLoaded(x)	로딩 여부 확인
JPAExpressions	서브쿼리 사용 시 필요한 클래스
from절 서브쿼리 제한	`FROM` 절의 서브쿼리는 JPQL/QueryDSL 모두 미지원
Expressions.constant(x)	상수 값 삽입 시 사용 (쿼리에는 포함되지 않고 조회 결과에 포함)
concat	문자열 결합 시 사용

3. 실습 코드

3-1. Q-Type 사용법

```
QMember q1 = new QMember("m");          // 별칭 지정
QMember q2 = QMember.member;            // 기본 Q 인스턴스 사용
```

1. JPQL 쿼리 확인 방법

```
spring.jpa.properties.hibernate.use_sql_comments: true
```

3-2. 기본 검색 조건

표현식	의미
eq("member1")	=
ne("member1")	!=
not()	부정

표현식	의미
isNotNull()	null 여부
in(10, 20)	포함 여부
notIn(...)	포함되지 않음
between(10, 20)	범위
goe(30) / gt(30)	≥ / >
loe(30) / lt(30)	≤ / <
like("member%")	패턴
contains("mem")	포함 여부
startsWith("mem")	시작 문자열 확인

```
Member result = queryFactory
    .selectFrom(member)
    .where(
        member.username.eq("member1"),
        member.age.eq(10)
    )
    .fetchOne();
```

3-3. 정렬

항목	설명
`desc()` , `asc()`	일반 정렬
`nullsLast()` , `nullsFirst()`	null 데이터 순서 부여

```
List<Member> result = queryFactory
    .selectFrom(member)
    .orderBy(member.age.desc(), member.username.asc().nullsLast())
    .fetch();
```

3-4. 페이징

```
QueryResults<Member> paged = queryFactory
    .selectFrom(member)
    .offset(1)
    .limit(2)
    .fetchResults();
```

3-5. 집합 함수 및 GroupBy

1. 집합함수 사용 예제

```
List<Tuple> result = queryFactory
    .select(
        member.count(),
        member.age.sum(),
        member.age.avg(),
        member.age.max(),
        member.age.min()
    )
    .from(member)
    .fetch();
```

2. GroupBy 사용

```
List<Tuple> groupResult = queryFactory
    .select(team.name, member.age.avg())
    .from(member)
    .join(member.team, team)
    .groupBy(team.name)
    .fetch();
```

3-6. 조인

1. 조인(Join)

항목	설명
`join()` , `innerJoin()`	내부 조인(inner join)
`leftJoin()`	left 외부 조인(left outer join)
`rightJoin()`	rigth 외부 조인(rigth outer join)

```
List<Member> result = queryFactory
    .selectFrom(member)
    .join(member.team, team)
    .where(team.name.eq("teamA"))
    .fetch();
```

2. 세타 조인 (비연관 조인)

- 외부 조인 불가능

```
List<Member> result = queryFactory
    .select(member)
```

```
    .from(member, team)
    .where(member.username.eq(team.name))
    .fetch();
```

3-7. ON절 조인

항목	설명
일반조인	leftJoin(member.team, team)
on조인	from(member).leftJoin(team).on(xxx)

1. 조인 대상 필터링

- inner join일 경우에는 where절에 on의 조건을 넣는것 과 동일함

```
List<Tuple> result = queryFactory
    .select(member, team)
    .from(member)
    .leftJoin(member.team, team).on(team.name.eq("teamA"))
    .fetch();
```

2. 연관관계 없는 외부 조인

```
List<Tuple> result = queryFactory
    .select(member, team)
    .from(member)
    .leftJoin(team).on(member.username.eq(team.name))
    .fetch();
```

3-8. Fetch Join

```
Member result = queryFactory
    .selectFrom(member)
    .join(member.team, team).fetchJoin()
    .where(member.username.eq("member1"))
    .fetchOne();
```

3-9. 서브쿼리

1. 조건절에서 서브쿼리

```
QMember memberSub = new QMember("memberSub");

List<Member> result = queryFactory
```

```
    .selectFrom(member)
    .where(member.age.eq(
        JPAExpressions
            .select(memberSub.age.max())
            .from(memberSub)
    ))
    .fetch();
```

2. SELECT절에서 서브쿼리

- **FROM절의 서브쿼리는 QueryDSL에서도 불가능**
  - 해결 방법:
    1. 서브쿼리를 JOIN으로 대체
    2. 쿼리를 분리하여 애플리케이션에서 조합
    3. Native SQL 사용

```
QMember memberSub = new QMember("memberSub"); // member는 이미 사용하고 있으므로 중복 회피
QMember memberSub = new QMember("memberSub");

List<Tuple> fetch = queryFactory
    .select(
        member.username,
        JPAExpressions
            .select(memberSub.age.avg())
            .from(memberSub)
    )
    .from(member)
    .fetch();
```

3-10. CASE 문

- select, 조건절(where), order by에서 사용가능

1. 단순 CASE

```
List<String> result = queryFactory
    .select(
        member.age
            .when(10).then("열살")
            .when(20).then("스무살")
            .otherwise("기타")
    )
    .from(member)
    .fetch();
```

2. 복잡한 조건 CASE

```
List<String> result = queryFactory
    .select(
        new CaseBuilder()
            .when(member.age.between(0, 20)).then("0~20살")
            .when(member.age.between(21, 30)).then("21~30살")
            .otherwise("기타")
    )
    .from(member)
    .fetch();
```

3-11. 상수 및 문자열 결합

1. 상수 표현

```
Tuple result = queryFactory
    .select(member.username, Expressions.constant("A"))
    .from(member)
    .fetchFirst();
```

2. 문자열 결합

```
String result = queryFactory
    .select(member.username.concat("_").concat(member.age.stringValue()))
    .from(member)
    .where(member.username.eq("member1"))
    .fetchOne();
```

4. 마무리

- QueryDSL은 JPQL의 단점을 보완하여 **타입 안전성, 자동 완성, 가독성 향상** 등의 장점을 제공
- JPAQueryFactory를 중심으로 Q타입 객체를 활용하여 동적 쿼리, 조인, 서브쿼리, 정렬, 페이징 등 다양한 기능을 직관적으로 구현
- fetch(), fetchOne(), fetchResults() 등 다양한 조회 메서드를 상황에 맞게 사용할 수 있으며, Tuple, Case, Expressions 등을 통해 복잡한 쿼리도 유연하게 작성 가능
- **FROM절의 서브쿼리**는 JPQL과 동일하게 **지원되지 않으며**, 이 경우는 조인으로 우회하거나, 쿼리를 분리하거나, Native SQL을 사용하는 방식으로 해결
- QueryDSL을 실무에 효과적으로 적용하기 위해서는 문법 숙지는 물론이고 **제약 사항과 대안 전략**에 대한 이해가 함께 필요

2025-07-19 - 중급문법

1. 학습 주제

- 중급 QueryDSL 문법 학습
  - 프로젝션 이해 및 DTO 반환 방식
  - 동적 쿼리 처리 방식
  - 수정 및 삭제 벌크 연산 처리
  - SQL Function 호출 방법

2. 주요 개념 요약

항목	설명
프로젝션	select 대상 지정. 단일이면 단일 타입 반환, 복수면 `Tuple`, DTO 등으로 반환
DTO 반환 방식	`Projections.bean()` : setter 기반 `Projections.constructor()` : 생성자 기반 `Projections.fields()` : 필드 직접 접근
별칭이 다를 경우	`ExpressionUtils.as()` 또는 `.as("alias")` 를 통해 명시적 별칭 설정
동적 쿼리 처리	`BooleanBuilder` 또는 `where()` 다중 파라미터 활용, null은 자동 무시됨
수정·삭제 벌크 연산	`add`, `multiply` 등 함수 제공. **실행 후 영속성 컨텍스트 초기화 필수**
SQL Function 호출	`Expressions.stringTemplate()` 사용. ANSI 표준 함수는 내장되어 있음

3. 실습 코드

3-1. 프로젝션 예제

```
// 단일 컬럼 조회
List<String> result = queryFactory
    .select(member.username)
    .from(member)
    .fetch();

// 다중 컬럼 조회 - Tuple
List<Tuple> result = queryFactory
    .select(member.username, member.age)
    .from(member)
    .fetch();
```

3-2. DTO.반환

```
// JPA - 생성자 기반 방식만 지원
List<MemberDto> result = em.createQuery(
```

```
    "select new study.querydsl.dto.MemberDto(m.username, m.age) " +
    "from Member m", MemberDto.class)
    .getResultList();

// QueryDSL - setter 접근
List<MemberDto> result = queryFactory
    .select(Projections.bean(MemberDto.class,
        member.username,
        member.age))
    .from(member)
    .fetch();

// 필드 접근 + 별칭
List<MemberDto> result = queryFactory
    .select(Projections.fields(MemberDto.class,
        member.username,
        ExpressionUtils.as(
            JPAExpressions
                .select(memberSub.age.max())
                .from(memberSub),
            "age")))
    .from(member)
    .fetch();

// 생성자 접근
List<MemberDto> result = queryFactory
    .select(Projections.constructor(MemberDto.class,
        member.username,
        member.age))
    .from(member)
    .fetch();

// @QueryProjection 활용 시
// MemberDto.java
public class MemberDto {
    private String username;
    private int age;

    @QueryProjection
    public MemberDto(String username, int age) {
        this.username = username;
        this.age = age;
    }
}

// DTO 반환
List<MemberDto> result = queryFactory
    .select(new QMemberDto(member.username, member.age))
    .from(member)
    .fetch();
```

3-3.동적 쿼리 처리

```
// 1. BooleanBuilder 사용
BooleanBuilder builder = new BooleanBuilder();
if (usernameCond != null) {
    builder.and(member.username.eq(usernameCond));
}
if (ageCond != null) {
    builder.and(member.age.eq(ageCond));
}
return queryFactory
    .selectFrom(member)
    .where(builder)
    .fetch();

// 2. where 절 다중 파라미터 활용
return queryFactory
    .selectFrom(member)
    .where(usernameEq(usernameCond), ageEq(ageCond))
    .fetch();

private BooleanExpression usernameEq(String usernameCond) {
    return usernameCond != null ? member.username.eq(usernameCond) :
null;
}
private BooleanExpression ageEq(Integer ageCond) {
    return ageCond != null ? member.age.eq(ageCond) : null;
}
```

3-4.수정, 삭제 벌크 연산

```
// 나이 1씩 증가
long count = queryFactory
    .update(member)
    .set(member.age, member.age.add(1))
    .execute();

// 이후 반드시 영속성 컨텍스트 초기화 필요
em.flush();
em.clear();
```

3-5.SQL function 호출

```
// username 내 'member' 문자열을 'M'으로 대체
String result = queryFactory
```

```
    .select(Expressions.stringTemplate(
        "function('replace', {0}, {1}, {2})",
        member.username, "member", "M"))
    .from(member)
    .fetchFirst();
```

4. 마무리

- 실무에서 DTO 반환 시 성능과 명확성을 위해 @QueryProjection과 Projections.constructor()를 선호
- 동적 쿼리 시 조건 누락을 막고 가독성을 높이기 위해 BooleanExpression 메서드 분리 권장
- 벌크 연산 이후에는 꼭 em.flush() + em.clear() 호출할 것

2025-07-20 - 실무활용

1. 학습 주제

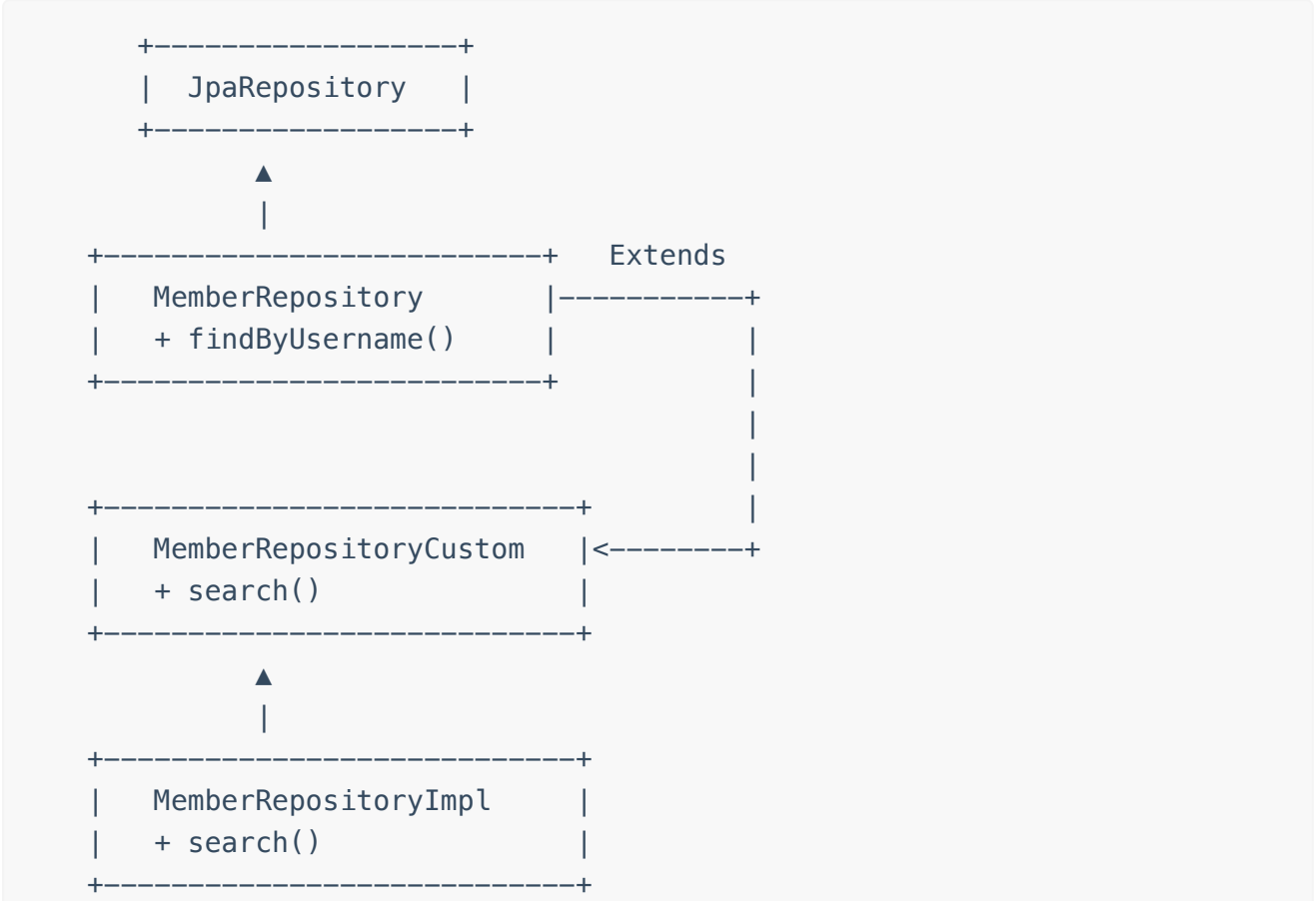
- Spring Data JPA에서 사용자 정의 리포지토리를 활용한 실무 개발 전략
  - 사용자 정의 리포지토리의 설계 및 구현 방식
  - QueryDSL과 결합한 동적 쿼리 처리
  - PageableExecutionUtils를 활용한 효율적인 페이징 처리
  - 정렬(Sort) 기능의 한계와 실무적인 대응 방안

2. 주요 개념 요약

항목	설명
사용자 정의 리포지토리 사용법	1. 사용자 정의 리포지토리 인터페이스 작성 2. 사용자 정의 리포지토리 구현 3. Spring Data 리포지토리에서 사용자 정의 리포지토리 인터페이스 상속
PageableExecutionUtils	Spring Boot 2.6 이상부터는 `PageableExecutionUtils.getPage()` 로 사용 (이전 버전은 패키지명이 다름)
fetchResults() / fetchCount()	`.fetchResults()` 는 count 쿼리와 content를 동시에 조회하지만 Spring Boot 2.6 이상에서는 deprecated 예정 → 별도로 `.fetch()` 와 `.fetchCount()` 또는 `.fetchOne()` 조합 권장
Spring Data 정렬	복잡한 정렬이 필요한 경우 Pageable의 Sort 사용이 제한적이므로 직접 정렬 조건을 파라미터로 받아 Querydsl에서 처리

3. 실습 코드

3-1. 사용자 정의 리파지토리 구성



3-2. 사용자 정의 리파지토리 구현

[1] 사용자 정의 인터페이스 작성

```
public interface MemberRepositoryCustom {
    List<MemberTeamDto> search(MemberSearchCondition condition);
}
```

[2] 사용자 정의 인터페이스 구현

```
public class MemberRepositoryImpl implements MemberRepositoryCustom {

    private final JPAQueryFactory queryFactory;

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

    @Override
    public List<MemberTeamDto> search(MemberSearchCondition condition) {
        // 구현 생략
```

```
    }
}
```

[3] 스프링 데이터 리포지토리에 사용자 정의 인터페이스 상속

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

3-3. Querydsl 페이징 연동

[1] 전체 카운트를 함께 조회하는 방법 (deprecated 예정)

```
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);
```

> ✅ 주의: fetchResults()는 Spring Boot 2.6 이상에서 deprecated 예정입니다.

[2] 데이터와 카운트를 별도 조회하는 방법 (권장)

```
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.count())
        .from(member)
        .leftJoin(member.team, team)
        .where(
            usernameEq(condition.getUsername()),
            teamNameEq(condition.getTeamName()),
            ageGoe(condition.getAgeGoe()),
            ageLoe(condition.getAgeLoe()))
        .fetchOne();

    return PageableExecutionUtils.getPage(content, pageable, () -> total);
```

3-4. 변경된 카운트 쿼리 방식 (Spring Boot 2.6 이상 기준)

```
Long totalCount = queryFactory
    // .select(Wildcard.count) // select count(*)
    .select(member.count())     // select count(member.id)
    .from(member)
    .fetchOne();
```

4. 마무리

- 사용자 정의 리포지토리는 **확장성과 재사용성이 뛰어나며,** Querydsl과 결합 시 **강력한 동적 쿼리 기능**을 제공함.
- 페이징 처리 시에는 **Spring Boot 최신 버전에 맞추어 .fetchResults() 대신 PageableExecutionUtils 사용이 권장**되지만, **정렬(Sort) 기능의 제약을 고려하여 추가적인 처리 로직이 필요**

2025-07-20 - 스프링 데이터 JPA가 제공하는 Querydsl 기능

1. 학습 주제

- Spring Data JPA에서 제공하는 QueryDSL 관련 기능 학습
  - `QuerydslPredicateExecutor`
  - Querydsl Web 통합
  - `QuerydslRepositorySupport`
  - 사용자 정의 Querydsl 지원 클래스 생성 및 확장

2. 주요 개념 요약

항목	설명
QuerydslPredicateExecutor	묵시적 조인은 가능하지만 **Left Join 불가능**. 클라이언트 코드가 Querydsl에 직접 의존하게 되며, **실무 적용에는 한계**가 명확함. 하지만 Pageable과 Sort는 기본적으로 지원됨.
Querydsl Web	컨트롤러 단에서 Querydsl Predicate를 직접 바인딩하여 조건을 처리할 수 있음. 단순한 조건만 처리 가능하며, 커스터마이징이 어렵고 **명시적이지 않음**. Controller가 Querydsl에 직접 의존.
QuerydslRepositorySupport	`getQuerydsl().applyPagination()`을 통해 Spring Data의 Pageable을 Querydsl 쿼리에 적용 가능. 하지만 **Sort는 제대로 지원되지 않으며**, Querydsl 3.x에 맞춰져 있어 **4.x의 `JPAQueryFactory` 사용이 불가능**.
Querydsl 지원 클래스 직접 생성	위의 한계를 극복하기 위한 **직접 구현 클래스**. `JPAQueryFactory` 기반에서 동작하며, **페이징/카운트 쿼리 분리 지원**, `select()` 또는 `selectFrom()` 부터 시작 가능. Spring Data JPA Sort도 지원. 실무에서 추천되는 방식.

3. 실습 코드

3-1. `QuerydslPredicateExecutor` 적용 예시

```
// 리포지토리 정의
public interface MemberRepository extends JpaRepository<Member, Long>,
    QuerydslPredicateExecutor<Member> {
}

// 사용 예시
Iterable<Member> result = memberRepository.findAll(
    member.age.between(10, 40)
    .and(member.username.eq("member1"))
);
```

3-2. 사용자 정의 지원 클래스: `Querydsl4RepositorySupport`

```
@Repository
public 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<?> query = contentQuery.apply(getQueryFactory());
        List<T> content = getQuerydsl().applyPagination(pageable,
query).fetch();
        return PageableExecutionUtils.getPage(content, pageable,
query::fetchCount);
    }

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

3-3. Querydsl4RepositorySupport 사용 예제

```
public Page<Member> applyPagination2(MemberSearchCondition condition,
Pageable pageable) {
    return applyPagination(
        pageable,
        queryFactory -> queryFactory
            .selectFrom(member)
            .leftJoin(member.team, team)
            .where(
                usernameEq(condition.getUsername()),
                teamNameEq(condition.getTeamName()),
                ageGoe(condition.getAgeGoe()),
                ageLoe(condition.getAgeLoe())
            ),
        countFactory -> countFactory
            .select(member.id)
            .from(member)
```

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

4. 마무리

- Spring Data JPA가 제공하는 기본 Querydsl 기능은 간단한 조건 처리에는 유용하지만, **복잡한 실무 요건을 만족하기에는 한계**
- QuerydslRepositorySupport는 기본 지원을 제공하지만 **Sort 미지원, JPAQueryFactory 미적용** 등의 문제가 존재
- 실무에서는 Querydsl4RepositorySupport와 같은 **직접 구현한 지원 클래스를 통해 페이징, 정렬, 카운트 쿼리 분리 등을 세밀하게 제어**하는 방식이 권장
