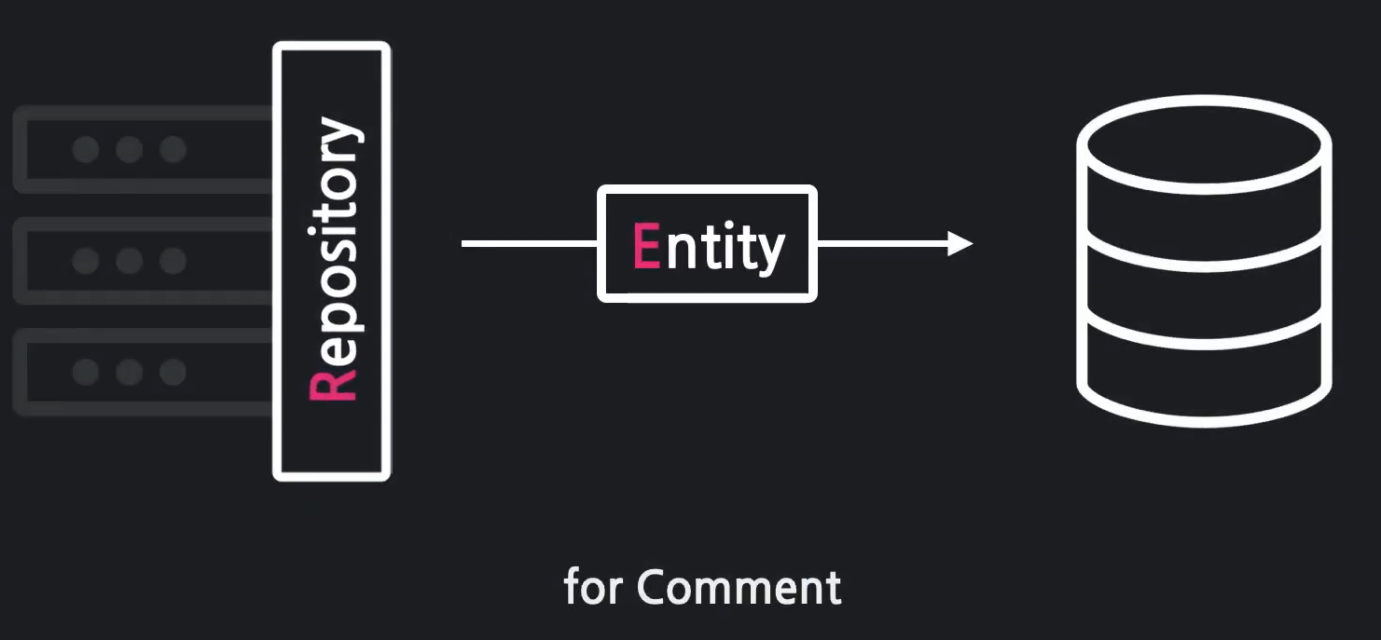
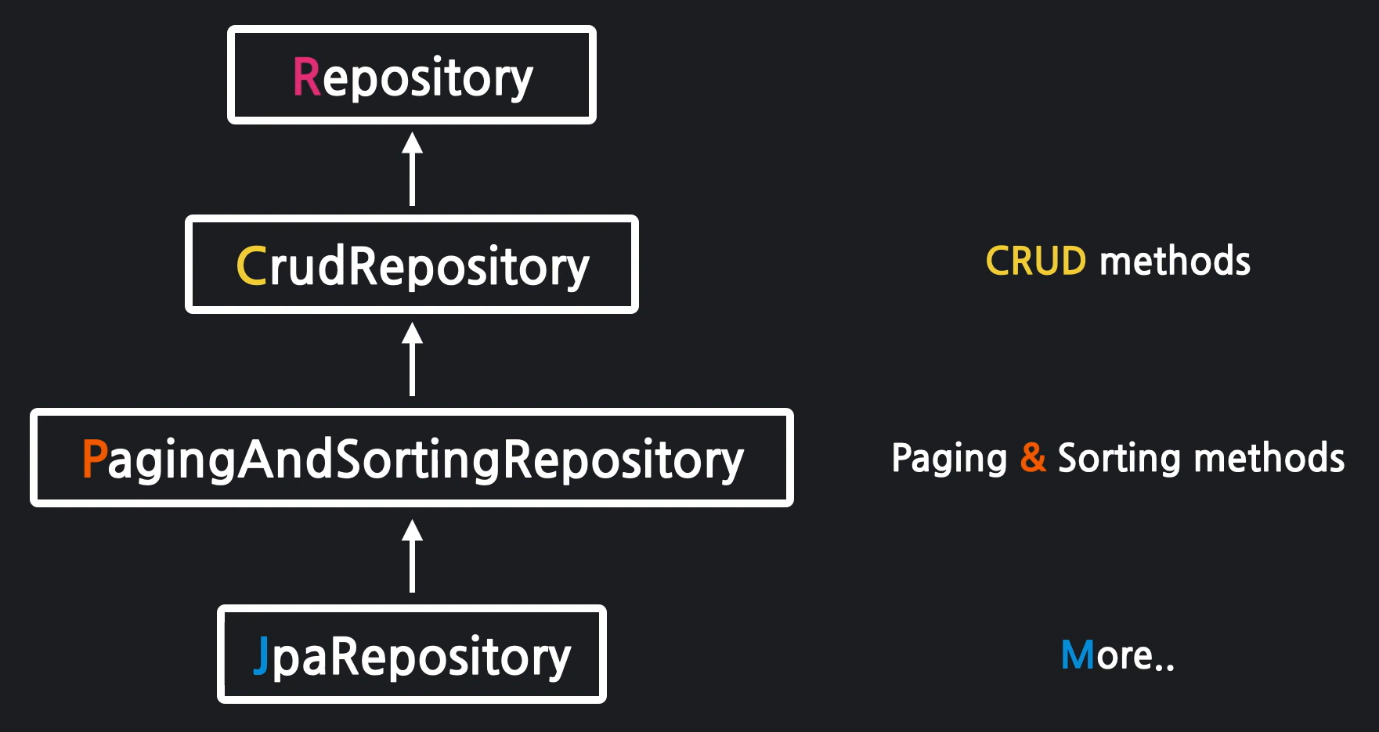
댓글 엔티티와 리파지터리(feat. 데이터의 관계)



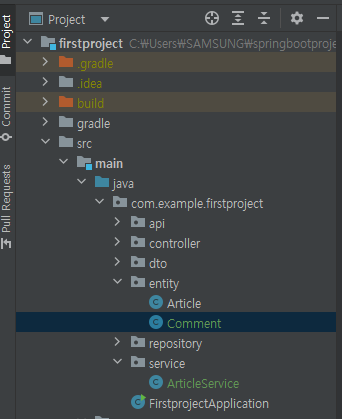
**PK, FK**



**JPA 리파지터리**

CrudRepository 말고 이를 확장한 JpaRepository를 사용해보자.  


**댓글 엔티티**



package com.example.firstproject.entity;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.ToString;

import javax.persistence.\*;

@Entity

@Getter

@ToString

@AllArgsConstructor

@NoArgsConstructor

public class Comment {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@ManyToOne // 해당 댓글 엔티티 여러개가 하나의 Article에 연관된다!!

@JoinColumn(name = "article\_id") // article 테이블의 id를 가져올떄 name

private Article article; // 댓글의 부모 게시글

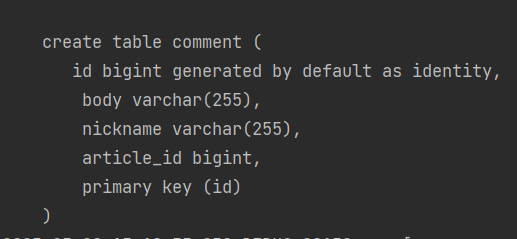
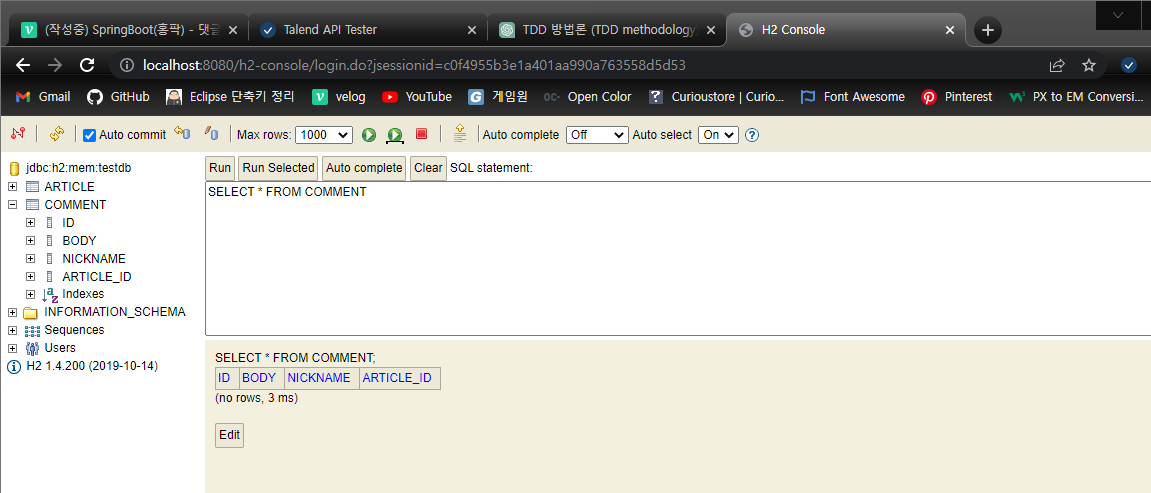
@Column

private String nickname;

@Column

private String body;

}

서버 돌리면 콘솔에서 table 생성 쿼리 확인 가능.  
  
DB에서 확인 가능.  


**더미 데이터 추가하기.**

data.sql에 댓글 데이터 추가.

INSERT INTO article(title, content) VALUES('가가가가', '1111');

INSERT INTO article(title, content) VALUES('나나나나', '2222');

INSERT INTO article(title, content) VALUES('다다다다', '3333');

-- article 테이블에 데이터 추가

INSERT INTO article(title, content) VALUES('당신의 인생 영화는?', '댓글 고');

INSERT INTO article(title, content) VALUES('당신의 소울 푸드는?', '댓글 고고');

INSERT INTO article(title, content) VALUES('당신의 취미는?', '댓글 고고고');

-- 4번 게시글의 댓글 추가

INSERT INTO comment(article\_id, nickname, body) VALUES(4, 'Park', '굿 윌 헌팅');

INSERT INTO comment(article\_id, nickname, body) VALUES(4, 'Kim', '아이 엠 샘');

INSERT INTO comment(article\_id, nickname, body) VALUES(4, 'Choi', '쇼생크 탈출');

-- 5번 게시글의 댓글 추가

INSERT INTO comment(article\_id, nickname, body) VALUES(5, 'Park', '치킨');

INSERT INTO comment(article\_id, nickname, body) VALUES(5, 'Kim', '샤브샤브');

INSERT INTO comment(article\_id, nickname, body) VALUES(5, 'Choi', '초밥');

-- 6번 게시글의 댓글 추가

INSERT INTO comment(article\_id, nickname, body) VALUES(6, 'Park', '조깅');

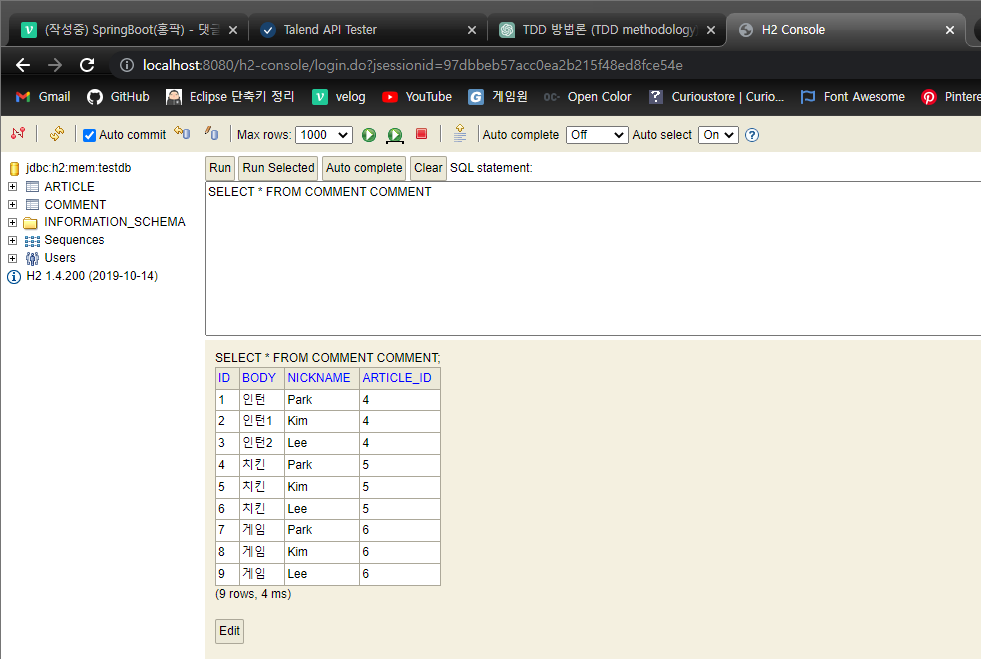
INSERT INTO comment(article\_id, nickname, body) VALUES(6, 'Kim', '유튜브 시청');

INSERT INTO comment(article\_id, nickname, body) VALUES(6, 'Choi', '독서');

-- 11장 셀프체크 데이터 추가

INSERT INTO coffee(name, price) VALUES('아메리카노', '4500');

INSERT INTO coffee(name, price) VALUES('라떼', '5000');

INSERT INTO coffee(name, price) VALUES('카페 모카', '5500');

**댓글 Repository**

**CommentRepository**

extends를 CrudRepository말고 JpaRepository를 사용하고 특정 쿼리를 실행하는 메소드를 추가하자.  
특정 쿼리를 실행하는 메소드를 작성하는 두가지 방법이 있다.  
1. @Query(직접 쿼리문 작성)  
2. XML 파일 이용

**1.@Query**

@Query : 특정 쿼리를 수행시키고 싶을때사용  
nativeQuery = true : true로 해줘야 해당 SQL문이 실행된다.

package com.example.firstproject.repository;

public interface CommentRepository extends JpaRepository<Comment, Long> {

// 특정 게시글의 모든 댓들 조회

@Query(value =

"SELECT \* " +

"FROM comment " +

"WHERE article\_id = :articleId",

nativeQuery = true)

List<Comment> findByArticleId(Long articleId);

}

**2.네이티브 쿼리 XML**

public interface CommentRepository extends JpaRepository<Comment, Long> {

// 1. @Query

// 특정 게시글의 모든 댓들 조회

@Query(value =

"SELECT \* " +

"FROM comment " +

"WHERE article\_id = :articleId",

nativeQuery = true)

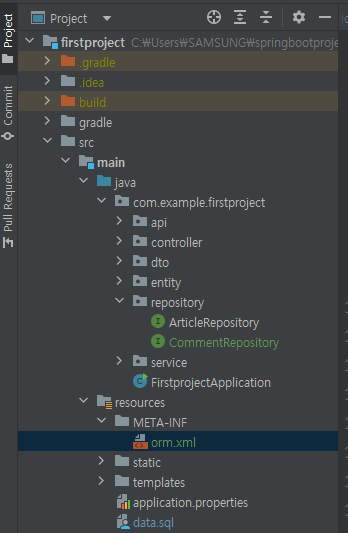
List<Comment> findByArticleId(Long articleId);

// 2. XML

// 특정 닉네임의 모든 댓글 조회

List<Comment> findByNickname(String nickname);

}

  
xml 파일 생성.  
설정 내용 구글링: "orm natice query orm.xml example"

* <named-native-query  
  name="Comment.findByNickname" :  
  Comment안에 findByNickname 메소드가 아래 <query>를 실행하고

리턴타입은 result-class="com.example.firstproject.entity.Comment"> 즉 Comment 엔티티를 반환하겠다.라는 뜻

<?xml version="1.0" encoding="utf-8" ?>

<entity-mappings xmlns="https://jakarta.ee/xml/ns/persistence/orm"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence/orm

https://jakarta.ee/xml/ns/persistence/orm/orm\_3\_0.xsd"

version="3.0">

<named-native-query

name="Comment.findByNickname"

result-class="com.example.firstproject.entity.Comment">

<query>

<![CDATA[

SELECT

\*

FROM

comment

WHERE

nickname = :nickname

]]>

</query>

</named-native-query>

</entity-mappings>

**레퍼지토리 Test**

package com.example.firstproject.repository;

import com.example.firstproject.entity.Article;

import com.example.firstproject.entity.Comment;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

import java.util.Arrays;

import java.util.List;

import static org.junit.jupiter.api.Assertions.\*;

@DataJpaTest

class CommentRepositoryTest {

@Autowired

CommentRepository commentRepository;

@Test

@DisplayName("특정 게시글의 모든 댓글 조회")

void findByArticleId() {

/\* Case 1: 4번 게시글의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

Long articleId = 4L;

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByArticleId(articleId);

// 3. 예상 데이터

Article article = new Article(4L, "당신의 인생 영화는?", "댓글 고");

Comment a = new Comment(1L, article, "Park", "굿 윌 헌팅");

Comment b = new Comment(2L, article, "Kim", "아이 엠 샘");

Comment c = new Comment(3L, article, "Choi", "쇼생크 탈출");

List<Comment> expected = Arrays.asList(a, b, c);

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(), "4번 글의 모든 댓글을 출력!");

}

/\* Case 2: 1번 게시글의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

Long articleId = 1L;

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByArticleId(articleId);

// 3. 예상 데이터

Article article = new Article(1L, "가가가가", "1111");

List<Comment> expected = Arrays.asList();

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(), "1번 글은 댓글이 없음");

}

/\* Case 3: 9번 게시글의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

Long articleId = 9L;

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByArticleId(articleId);

// 3. 예상 데이터

Article article = null;

List<Comment> expected = Arrays.asList();

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(),

"9번 글 자체가 없으므로 댓글은 비어 있어야 함");

}

/\* Case 4: 999번 게시글의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

Long articleId = 999L;

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByArticleId(articleId);

// 3. 예상 데이터

Article article = null;

List<Comment> expected = Arrays.asList();

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(),

"999번 글 자체가 없으므로, 댓글은 비어 있어야 함");

}

/\* Case 5: -1번 게시글의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

Long articleId = -1L;

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByArticleId(articleId);

// 3. 예상 데이터

Article article = null;

List<Comment> expected = Arrays.asList();

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(),

"-1번 글 자체가 없으므로, 댓글은 비어 있어야 함");

}

}

@Test

@DisplayName("특정 닉네임의 모든 댓글 조회")

void findByNickname() {

/\* Case 1: "Park"의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

String nickname = "Park";

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByNickname(nickname);

// 3. 예상 데이터

Comment a = new Comment(1L, new Article(4L, "당신의 인생 영화는?", "댓글 고"),

nickname, "굿 윌 헌팅");

Comment b = new Comment(4L, new Article(5L, "당신의 소울 푸드는?", "댓글 고고"),

nickname, "치킨");

Comment c = new Comment(7L, new Article(6L, "당신의 취미는?", "댓글 고고고"),

nickname, "조깅");

List<Comment> expected = Arrays.asList(a, b, c);

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(), "Park의 모든 댓글을 출력!");

}

/\* Case 2: "Kim"의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

String nickname = "Kim";

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByNickname(nickname);

// 3. 예상 데이터

Comment a = new Comment(2L, new Article(4L, "당신의 인생 영화는?",

"댓글 고"), nickname, "아이 엠 샘");

Comment b = new Comment(5L, new Article(5L, "당신의 소울 푸드는?",

"댓글 고고"), nickname, "샤브샤브");

Comment c = new Comment(8L, new Article(6L, "당신의 취미는?",

"댓글 고고고"), nickname, "유튜브 시청");

List<Comment> expected = Arrays.asList(a, b, c);

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(),

"Kim의 모든 댓글을 출력!");

}

/\* Case 3: null의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

String nickname = null;

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByNickname(nickname);

// 3. 예상 데이터

List<Comment> expected = Arrays.asList();

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(),

"null의 모든 댓글을 출력!");

}

/\* Case 4: ""의 모든 댓글 조회 \*/

{

// 1. 입력 데이터 준비

String nickname = "";

// 2. 실제 데이터

List<Comment> comments = commentRepository.findByNickname(nickname);

// 3. 예상 데이터

List<Comment> expected = Arrays.asList();

// 4. 비교 및 검증

assertEquals(expected.toString(), comments.toString(),

"\"\"의 모든 댓글을 출력!");

}

}

}