## 14장 완성 코드

### entity/Comment.java

**package** com.example.firstproject.entity;  
  
**import** jakarta.persistence.\*;  
**import** lombok.AllArgsConstructor;  
**import** lombok.Getter;  
**import** lombok.NoArgsConstructor;  
**import** lombok.ToString;  
  
@Entity *// 해당 클래스가 엔티티임을 선언, 클래스 필드를 바탕으로 DB에 테이블 생성*@Getter *// 각 필드 값을 조회할 수 있는 getter 메서드 자동 생성*@ToString *// 모든 필드를 출력할 수 있는 toString 메서드 자동 생성*@AllArgsConstructor *// 모든 필드를 매개변수로 갖는 생성자 자동 생성*@NoArgsConstructor *// 매겨변수가 아예 없는 기본 생성자 자동 생성***public class** Comment {  
 @Id *// 대표키 지정* @GeneratedValue(strategy= GenerationType.***IDENTITY***) *// DB가 자동으로 1씩 증가* **private** Long **id**; *// 대표키* @ManyToOne *// 이 엔티티(Comment)와 부모 엔티티(Article)를 다대일 관계로 설정* @JoinColumn(name=**"article\_id"**) *// 외래키 생성, Article 엔티티의 기본키(id)와 매핑* **private** Article **article**; *// 해당 댓글의 부모 게시글* @Column *// 해당 필드를 테이블의 속성으로 매핑* **private** String **nickname**; *// 댓글을 단 사람* @Column *// 해당 필드를 테이블의 속성으로 매핑* **private** String **body**; *// 댓글 본문*}

### resources/data.sql

**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'**);

### repository/CommentRepository.java

**package** com.example.firstproject.repository;  
  
**import** com.example.firstproject.entity.Comment;  
**import** org.springframework.data.jpa.repository.JpaRepository;  
**import** org.springframework.data.jpa.repository.Query;  
  
**import** java.util.List;  
  
**public interface** CommentRepository **extends** JpaRepository<Comment, Long> {  
 *// 특정 게시글의 모든 댓글 조회* @Query(value = **"SELECT \* FROM comment WHERE article\_id = :articleId"**,  
 nativeQuery = **true**) *// value 속성에 실행하려는 쿼리 작성* List<Comment> findByArticleId(Long articleId);  
  
 *// 특정 닉네임의 모든 댓글 조회* List<Comment> findByNickname(String nickname);  
}

### resources/META-INF/orm.xml

*<?*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>

### test/…/repository/CommentRepositoryTest.java

**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"**);  
 }  
 }  
  
 @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!"**);  
 }  
 }  
}

### 셀프체크 정답

#### test/…/repository/CommentRepositoryTest.java

**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 비어있어야 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 비어있어야 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!"**);  
 }  
 }  
}