

## 3장. Spring Data JPA



*JPA – mysql, junit*



Spring Boot

# 스프링과 JPA

## ■ 스프링과 JPA

데이터베이스 연동에 사용되는 기술은 전통적인 JDBC, 스프링 MyBatis, 하이버네이트 ORM(Object Relational Mapping)에 이르기 까지 다양하다.

이 중에서 하이버네이트 ORM은 애플리케이션에서 사용하는 SQL까지도 프레임워크에서 제공하기 때문에 개발자가 처리해야 할 일들을 많이 줄여준다.

ORM이란 "객체지향 구조를 관계형 구조로 매핑"하는 기술이다.

이런 **ORM을 보다 쉽게 사용할 수 있도록 표준화 시킨 것이 JPA(Java Persistence API)**이다. 다시 말하면 ORM 기술을 Java 언어에 맞도록 스펙으로 정리한 것이라 할 수 있다..

JPA 구현체는 하이버네이트, EclipseLink, DataNucleus 등 여러가지가 있는데 **스프링 부트**에서는 기본적으로 **하이버네이트를 JPA 구현체**로 이용한다.



# 스프링과 JPA

## ▪ SQL을 직접 다루는 기술

애플리케이션은 사용자가 입력한 데이터나 운용 과정에서 생성된 데이터를 재사용 하기위해 데이터베이스 같은 저장공간에 저장해야 한다. 이 때 SQL이 사용된다.

### 오라클 DB 테이블

```
CREATE TABLE board(  
    seq NUMBER(5) PRIMARY KEY,  
    title VARCHAR2(200),  
    writer VARCHAR2(20),  
    content VARCHAR2(2000),  
    regdate DATE DEFAULT SYSDATE,  
    cnt NUMBER(5) DEFAULT 0  
);
```

### VO 클래스

```
@Getter  
@Setter  
public class BoardVO {  
    private int seq;  
    private String title;  
    private String writer;  
    private String content;  
    private Date createDate;  
    private int cnt = 0;  
}
```

// 글 등록

```
INSERT INTO board(seq, title, writer, content) VALUES(seq.nextval, ?, ?, ?, ?);
```

//글 목록 조회

```
SELECT * FROM board;
```



# 스프링과 JPA

- SQL을 직접 다루지 않는 기술

BoardVO를 Map에 저장하고 관리했을때의 CRUD 코드

```
Map<String, BoardVO> boardList = new HashMap<>();

BoardVO board = new BoardVO();
board.setSeq(1);
board.setTitle("테스트 제목...");
board.setWriter("테스터");
board.setContent("테스트 내용입니다...");
board.setCreateDate(new Date());
board.setCnt(0);

//게시글 등록
boardList.put("board", board);
```

BoardVO 객체를 Map에 저장했기 때문에 소스 어디에도 BOARD 테이블(DB)과 관련된 SQL이 사용되지 않는다.



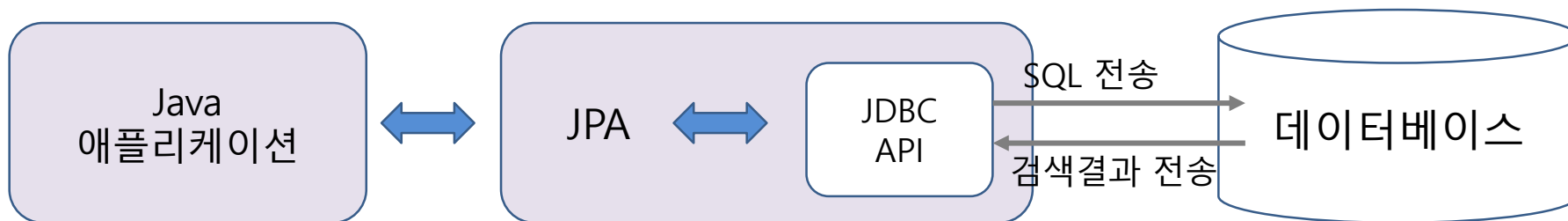
# 스프링과 JPA

## ■ JPA 동작 원리

JPA는 자바 객체를 컬렉션에 저장하고 관리하는 것과 비슷한 개념이다.

하지만 결국 컬렉션에 저장된 객체를 테이블의 로우와 매핑하기 위해서는 누군가가 JDBC API를 이용해서 실질적인 연동 작업을 처리해야 한다.

JPA는 자바 애플리케이션과 JDBC 사이에 존재하면서 JDBC의 복잡한 절차를 대신 처리해 준다.



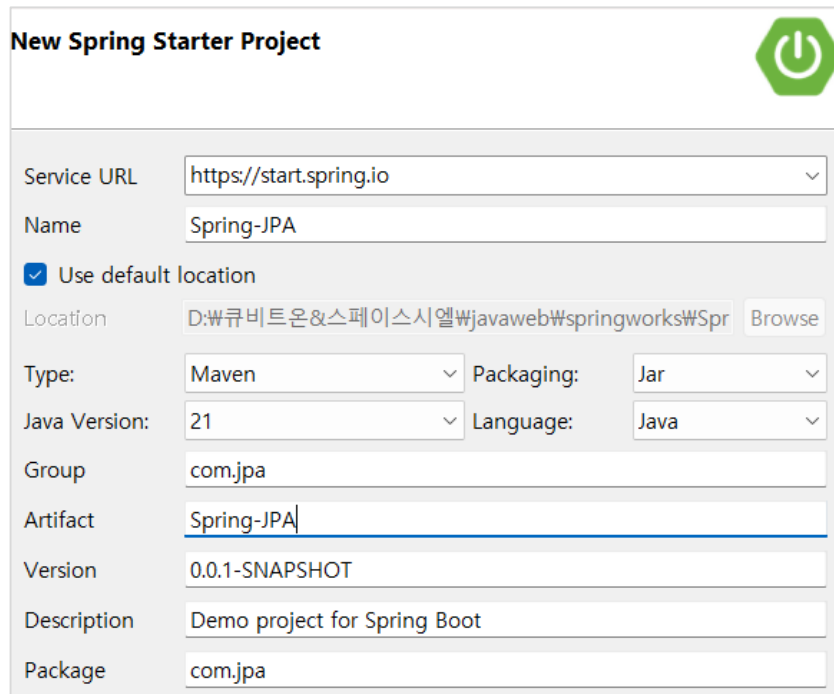
JPA가 데이터베이스 연동에 사용되는 코드 뿐만 아니라 SQL까지도 제공한다.

테이블과 VO 클래스 이름을 똑같이 매핑하고, 테이블의 칼럼을 VO 클래스의 멤버 변수와 매핑하여 동작한다.



# 스프링 데이터 JPA

## ■ 프로젝트 생성 및 기본 설정



The image shows the 'New Spring Starter Project' dialog box in an IDE. It contains the following fields and settings:

- Service URL:** `https://start.spring.io`
- Name:** `Spring-JPA`
- ☒ **Use default location**
- Location:** `D:\₩큐비트온&스페이스시엘₩javaweb₩springworks₩Spr` (with a 'Browse' button)
- Type:** `Maven`
- Packaging:** `Jar`
- Java Version:** `21`
- Language:** `Java`
- Group:** `com.jpa`
- Artifact:** `Spring-JPA`
- Version:** `0.0.1-SNAPSHOT`
- Description:** `Demo project for Spring Boot`
- Package:** `com.jpa`

[New] -> [Spring Starter Project]

Name – Spring-JPA

Type - Maven

Packing – Jar

Java Version – 21

Group – com.jpa


Package –  
com.springboot.jpa



# 스프링 데이터 JPA

## ■ 프로젝트 생성 및 기본 설정

### New Spring Starter Project Dependencies



Spring Boot Version:

Frequently Used:

<input checked="" type="checkbox"/> Lombok	<input checked="" type="checkbox"/> MySQL Driver	<input checked="" type="checkbox"/> Spring Boot DevTools
<input checked="" type="checkbox"/> Spring Data JPA	<input checked="" type="checkbox"/> Spring Web	<input checked="" type="checkbox"/> Thymeleaf

Available:

Type to search dependencies

- ▶ AI
- ▶ Developer Tools
- ▶ Google Cloud
- ▶ I/O

Selected:

- X Spring Boot DevTools
- X Lombok
- X Spring Data JPA
- X MySQL Driver
- X Thymeleaf
- X Spring Web



# 스프링과 JPA

- JPA 환경 설정 – pom.xml

```
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
    <groupId>com.mysql</groupId>  
    <artifactId>mysql-connector-j</artifactId>  
    <scope>runtime</scope>  
</dependency>
```





# 스프링 데이터 JPA

## ▪ Mysql 설정

톰캣 서버를 실행할때 데이터 베이스 설정이 되어 있지 않으면 서버 오류가 발생하므로 mysql의 datasource를 설정함

Description:

Failed to configure a DataSource: 'url' attribute is not specified

Reason: Failed to determine a suitable driver class

application.properties

# DataSource 설정 - mysql

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/bootdb?serverTime=Asia/Seoul

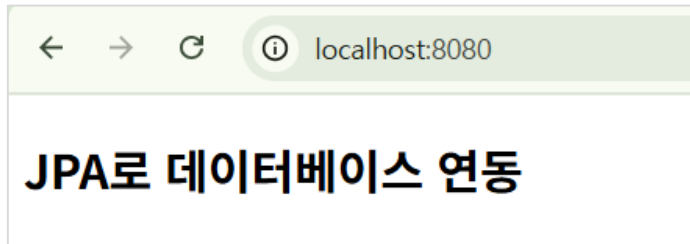
spring.datasource.username=bootuser

spring.datasource.hikari.password=pwboot



# 스프링 데이터 JPA

- 홈페이지 화면 띄우기



home.html

```
@Controller
public class HomeController {

    @GetMapping("/")
    public String Home() {
        return "home";
    }
}
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>home page</title>
</head>
<body>
    <h2>JPA로 데이터베이스 연동 </h2>
</body>
</html>
```



# 스프링 데이터 JPA

- JPA를 활용한 데이터 관리

JPA로 데이터베이스 연동하여 데이터를 생성하고 조회한다.  
먼저 JPA 기본 설정을 해야 함.

```
# JPA 설정
```

```
spring.jpa.hibernate.ddl-auto=create
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.format_sql=true
```

```
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```



# 스프링 데이터 JPA

## ■ 엔티티 매핑하기

어노테이션	의미
@Entity	@Entity가 설정된 클래스를 엔티티라 하며, 기본적으로 클래스 이름과 동일한 테이블과 매핑된다.
@Table	엔티티 이름과 매핑될 테이블 이름이 다른 경우 name 속성을 사용하여 매핑한다. 엔티티 이름과 테이블 이름이 동일하면 생략해도 됨
@Id	테이블의 기본 키를 매핑한다.
@GeneratedValue	@Id가 선언된 필드에 기본 키 값을 자동으로 할당한다.



# 스프링 데이터 JPA

## ▪ Board 엔티티 매핑하기

```
@Builder
@ToString
@Setter
@Getter
@Entity
public class Board {
    @Id //기본키(Primary Key) 설정
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id; //자동 순번

    @Column(nullable=false) //필수 입력
    private String title; //글 제목

    @Column(length=20, nullable=false)
    private String writer; //글쓴이

    @Column(length=4000, nullable=false)
    private String content; //글 내용

    @CreationTimestamp
    private Timestamp createDate; //작성일
}
```



# 스프링 데이터 JPA

- Board 엔티티 매핑하기

```
2025-10-10T14:53:01.397+09:00 INFO 13720 --- [Spring-JPA]
Hibernate:
    drop table if exists board
Hibernate:
    create table board (
        id integer not null auto_increment,
        create_date datetime(6),
        writer varchar(20) not null,
        content varchar(4000) not null,
        title varchar(255) not null,
        primary key (id)
    ) engine=InnoDB
```



# 스프링 데이터 JPA

- Mysql WorkBench - DB 확인

```
1
2 • desc board;
3
4 • select * from board;
5
```

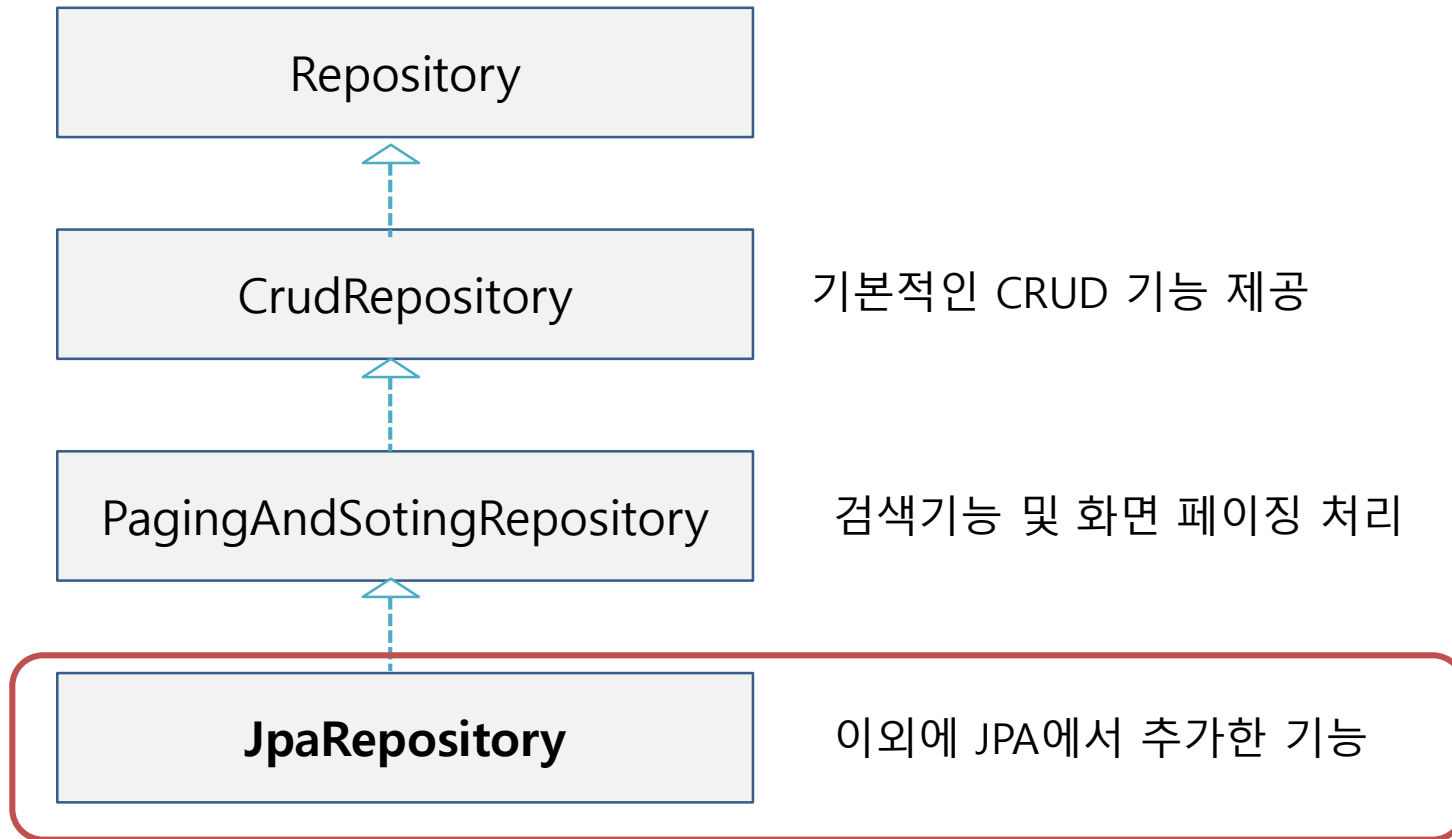
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Field	Type	Null	Key	Default	Extra
▶	id	int	NO	PRI	NULL	auto_increment
	create_date	datetime(6)	YES		NULL	
	writer	varchar(20)	NO		NULL	
	content	varchar(4000)	NO		NULL	
	title	varchar(255)	NO		NULL	



# 스프링 데이터 JPA

- 리포지터리 작성하기 - Repository 인터페이스





# 스프링 데이터 JPA

- 리포지터리 작성하기

**JpaRepository<T, ID>**

T : 엔티티의 클래스 타입

ID : 식별자(PK) 타입(@Id로 매핑한 식별자 변수의 자료형)

별도의 구현 클래스를 만들지 않고 인터페이스만 정의함으로써 기능을 사용할 수 있음

```
import org.springframework.data.jpa.repository.JpaRepository;

import com.jpa.model.Board;

//JpaRepository를 상속받은 BoardRepository 인터페이스
public interface BoardRepository extends JpaRepository<Board, Integer>{

}
```



# 스프링 데이터 JPA

## ▪ JPA를 활용한 CRUD 테스트

작업	메서드
INSERT	save(엔티티 객체)
SELECT	findById(키 타입), get()
	findAll() - 목록
UPDATE	save(엔티티 객체)
DELETE	deleteById(키 타입), delete(객체 타입)



# 스프링 데이터 JPA

- JPA를 활용한 CRUD 테스트

- 실행전 테이블 자동생성 기능을 update로 변경

```
# JPA 설정
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```



# 스프링 데이터 JPA

## ▪ JUnit 도구를 활용한 테스트

- **JUnit**은 자바(Java)를 위한 단위 테스트 프레임워크로, 코드의 특정 부분(함수, 메소드, 클래스 등)이 의도한 대로 작동하는지 검증한다.
- 테스트 시간 단축, 쉬운 코드 변경, 안정성 향상에 도움을 준다.
- @Test와 같은 어노테이션을 사용하여 테스트를 간결하게 작성한다.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

```
▼ 📁 src/test/java
  ▼ 📁 com.jpa
    > 📄 BoardRepositoryTest.java
    > 📄 QueryMethodTest.java
    > 📄 SpringJpaApplicationTests.java
```



# 스프링 데이터 JPA

- 등록 기능 테스트 – save() 메서드 사용

```
@Slf4j
@SpringBootTest
public class BoardRepositoryTest {

    @Autowired
    private BoardRepository repository;

    //게시글 생성
    @Test
    public void insertBoard() {
        Board board = new Board();

        board.setTitle("가입 인사");
        board.setWriter("박신입");
        board.setContent("안녕하세요~ 방가방가");
        board.setCreateDate(new Timestamp(System.currentTimeMillis()));

        repository.save(board);

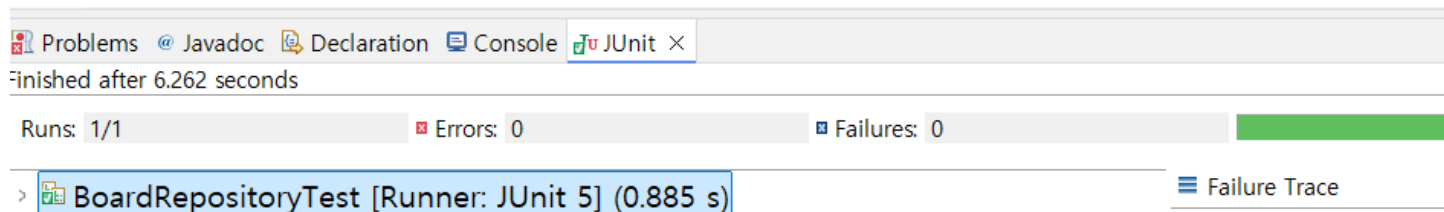
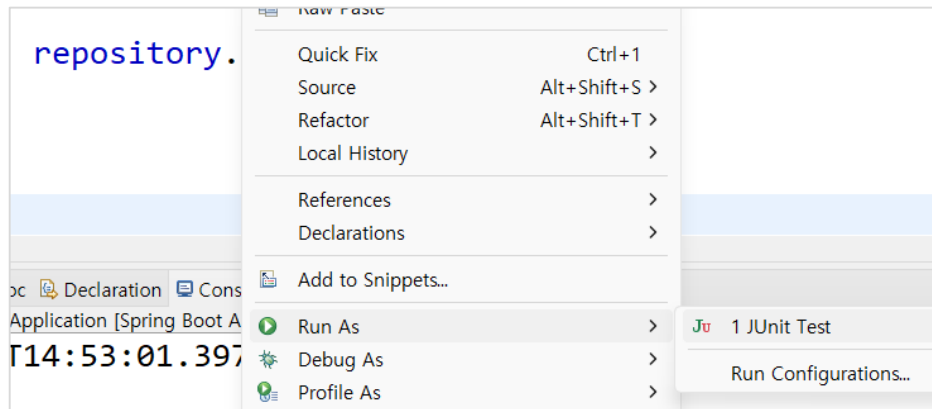
        Log.info("board:" + board);
    }
}
```



# 스프링 데이터 JPA

- 등록 기능 테스트 – save() 메서드 사용

실행: Run As > JUnit Test



```
: board:Board(id=1, title=가입 인사, writer=박신입, content=안녕하세요~ 방가방가,  
: Closing JPA EntityManagerFactory for persistence unit 'default'  
: HikariPool-1 - Shutdown initiated...  
: HikariPool-1 - Shutdown completed.
```



# 스프링 데이터 JPA

- 등록 기능 테스트 – save() 메서드 사용

롬복 @Builder 사용

```
@Builder //junit 테스트에서 사용
@ToString
@Setter
@Getter
@Entity
public class Board {
    @Id //기본키(Primary Key) 설정
```

```
Board board = Board.builder()
    .title("안녕하세요")
    .writer("이신입")
    .content("잘 부탁드립니다..")
    .createDate(new Timestamp(System.currentTimeMillis()))
    .build();

repository.save(board);

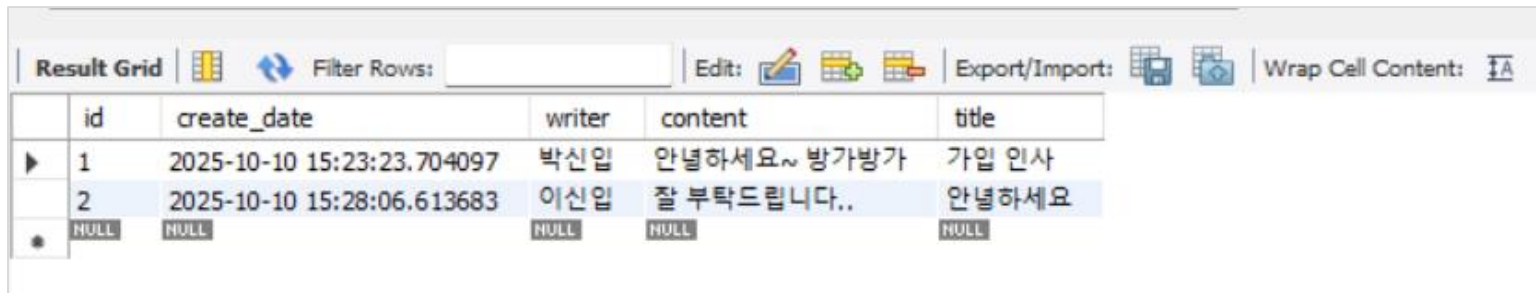
Log.info("board:" + board);
```



# 스프링 데이터 JPA

- 등록 기능 테스트 – save() 메서드 사용

Mysql WorkBench에서 Select



	id	create_date	writer	content	title
▶	1	2025-10-10 15:23:23.704097	박신입	안녕하세요~ 방가방가	가입 인사
	2	2025-10-10 15:28:06.613683	이신입	잘 부탁드립니다..	안녕하세요
*	NULL	NULL	NULL	NULL	NULL





# 스프링 데이터 JPA

- 목록 조회 기능 테스트 – findAll() 사용

```
//게시글 목록
@Test
public void getBoardList() {
    List<Board> boardList = repository.findAll();

    //기본 생성자 오류 해결 - Board에 NoArgsConstructor
    /*for(Board board : boardList)
        log.info(board.toString());*/
    boardList.forEach(board -> Log.info(" " + board));
}
```

```
Board(id=1, title=가입 인사, writer=박신입, content=안녕하세요~ 방가방가
Board(id=2, title=안녕하세요, writer=이신입, content=잘 부탁드립니다..,
Closing JPA EntityManagerFactory for persistence unit 'default'
HikariPool-1 - Shutdown initiated...
```



# 스프링 데이터 JPA

- 상세 조회 기능 테스트 – findById(seq) 사용

```
//게시글 상세
@Test
public void getBoard() {
    //2번째 게시글 가져오기
    Board board = repository.findById(2).get();
    Log.info(board.toString());
}
```



# 스프링 데이터 JPA

- 글 수정 - findById(seq)로 조회후 save()로 재등록

```
//게시글 수정
@Test
public void updateBoard() {
    Log.info("** 2번 게시글 조회 **");
    Board board = repository.findById(2).get();

    Log.info("** 2번 게시글 제목 수정 **");
    board.setTitle("제목을 수정합니다.");

    //수정후 저장
    repository.save(board);
}
```

```
Board(id=2, title=제목을 수정합니다., writer=이신입, content=잘 부탁드립니다.
Closing JPA EntityManagerFactory for persistence unit 'default'
HikariPool-1 - Shutdown initiated...
```



# 스프링 데이터 JPA

- 글 삭제 기능 테스트 – deleteById(seq) 사용

```
//게시글 삭제
@Test
public void deleteBoard() {
    Log.info("1번째 게시글 삭제");
    repository.deleteById(1);
}
```

```
Hibernate:
delete
from
    board
where
    id=?
```

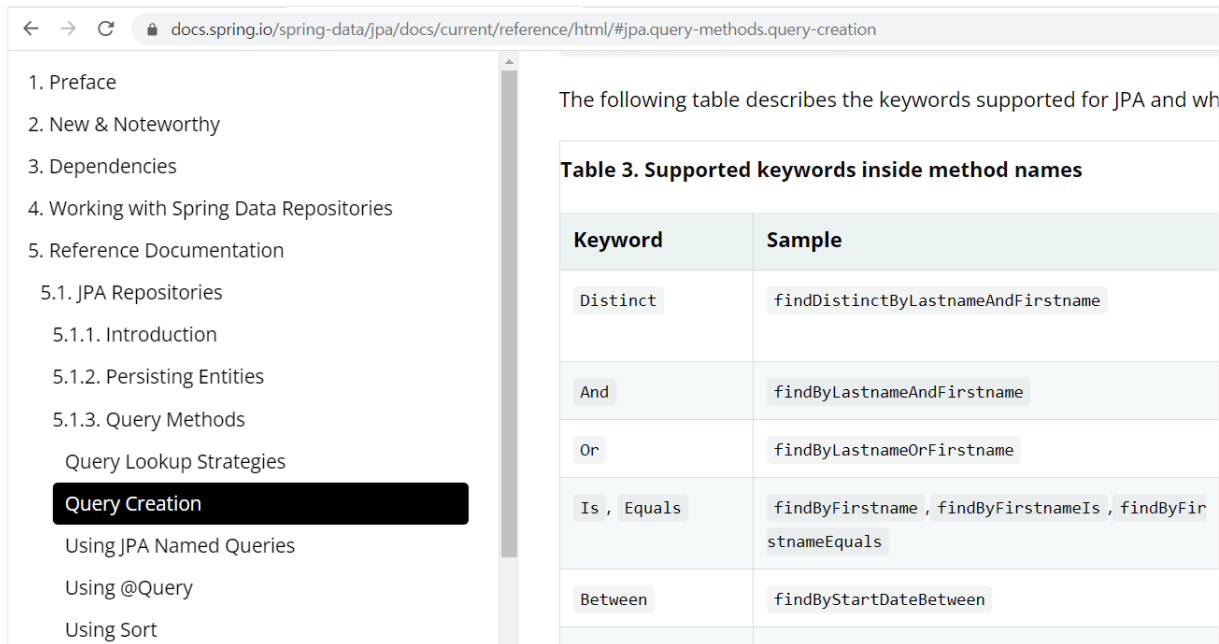


# 쿼리 메서드(Query Method)

## ■ 쿼리 메서드란?

메서드의 이름으로 필요한 쿼리를 만들어 주는 기능을 한다.

findBy..., getBy... 등으로 메서드의 이름을 시작하고 칼럼과 키워드를 연결하는 것으로 메서드를 작성한다. (예, findByAgeOrderByNameDesc() )



The screenshot shows the Spring Data JPA documentation page. The left sidebar contains a table of contents with 'Query Creation' highlighted. The main content area shows a table titled 'Table 3. Supported keywords inside method names'.

Keyword	Sample
Distinct	findDistinctByLastnameAndFirstname
And	findByLastnameAndFirstname
Or	findByLastnameOrFirstname
Is , Equals	findByFirstname , findByFirstnameIs , findByFirstnameEquals
Between	findByStartDateBetween

Spring Data  
JPA Document



# 쿼리 메서드(Query Method)

- 쿼리 메서드 사용하기

find + 엔티티 이름 + By + 변수 이름 (엔티티 이름은 생략 가능)

예) findBoardByTitle() : Board 엔티티에서 title 변수 값만 조회한다.

```
//JpaRepository를 상속받은 BoardRepository 인터페이스
public interface BoardRepository extends JpaRepository<Board, Integer>{
    //쿼리 메서드 - 글 제목 검색
    List<Board> findByTitle(String searchKeyword);
}
```



# 쿼리 메서드(Query Method)

- 쿼리 메서드 사용하기

```
@Slf4j //로그 출력
@SpringBootTest
public class QueryMethodTest {

    @Autowired
    private BoardRepository repository;

    @BeforeEach
    public void dataPrepare() {
        for(int i=1; i<=200; i++) {
            Board board = new Board();
            board.setTitle("테스트 제목 " + i);
            board.setWriter("테스터");
            board.setContent("테스트 내용 " + i);
            board.setCreateDate(new Timestamp(System.currentTimeMillis()));

            repository.save(board);
        }
    }
}
```



# 쿼리 메서드(Query Method)

- 쿼리 메서드 사용하기

```
@Test
public void testFindByTitle() {
    //findByTitle(keyword) 사용
    List<Board> boardList = boardRepo.findByTitle("테스트 제목 10");

    Log.info("검색 결과");
    for(Board board : boardList) {
        Log.info("--->" + board.toString());
    }
}
```

```
: 검색 결과
: --->Board(id=10, title=테스트 제목 10, writer=테스터, content=테스트 내용 10,
```





# 쿼리 메서드(Query Method)

- LIKE 연산자 사용하기

게시글 내용에 특정 단어가 포함된 목록을 검색하려면 LIKE 연산자와 더불어 Containing 키워드를 사용한다.

```
public interface BoardRepository extends JpaRepository<Board, Integer>{  
    //쿼리 메서드 - 글 제목 검색  
    List<Board> findByTitle(String searchKeyword);  
  
    //특정 단어가 포함된 글 내용 검색  
    List<Board> findByContentContaining(String searchKeyword);  
}
```



# 쿼리 메서드(Query Method)

- LIKE 연산자 사용하기

```
@Test
public void testFindByContentContaining() {
    List<Board> boardList = repository.findByContentContaining("17");

    Log.info("검색 결과");
    for(Board board : boardList) {
        Log.info("--->" + board.toString());
    }
}
```

where 절에 like 연산자가 사용됨

```
Hibernate:
select
    b1_0.id,
    b1_0.content,
    b1_0.create_date,
    b1_0.title,
    b1_0.writer
from
    board b1_0
where
    b1_0.content like ? escape '\\'
```



# 쿼리 메서드(Query Method)

- LIKE 연산자 사용하기

```
: 검색 결과
: --->Board(id=17, title=테스트 제목 17, writer=테스터, c
: --->Board(id=117, title=테스트 제목 117, writer=테스터,
: --->Board(id=170, title=테스트 제목 170, writer=테스터,
: --->Board(id=171, title=테스트 제목 171, writer=테스터,
: --->Board(id=172, title=테스트 제목 172, writer=테스터,
: --->Board(id=173, title=테스트 제목 173, writer=테스터,
: --->Board(id=174, title=테스트 제목 174, writer=테스터,
: --->Board(id=175, title=테스트 제목 175, writer=테스터,
: --->Board(id=176, title=테스트 제목 176, writer=테스터,
: --->Board(id=177, title=테스트 제목 177, writer=테스터,
: --->Board(id=178, title=테스트 제목 178, writer=테스터,
: --->Board(id=179, title=테스트 제목 179, writer=테스터,
```



# 쿼리 메서드(Query Method)

- 여러 조건 사용하기

제목 혹은 내용에 특정 검색어가 포함된 게시 글 목록을 검색하는 경우 'OR' 키워드를 결합하여 사용한다.

```
public interface BoardRepository extends JpaRepository<Board, Integer>{  
    //쿼리 메서드 - 글 제목 검색  
    List<Board> findByTitle(String searchKeyword);  
  
    //특정 단어가 포함된 글 내용 검색  
    List<Board> findByContentContaining(String searchKeyword);  
  
    //제목 또는 내용에 특정 단어가 포함된 목록 검색  
    List<Board> findByTitleContainingOrContentContaining(String title,  
                                                         String content);  
}
```



# 쿼리 메서드(Query Method)

- 여러 조건 사용하기

```
@Test
public void testFindByTitleContainingOrContentContaining() {
    List<Board> boardList =
        boardRepo.findByTitleContainingOrContentContaining("17", "18");

    Log.info("검색 결과");
    for(Board board : boardList) {
        Log.info("--->" + board.toString());
    }
}
```

```
select
    b1_0.id,
    b1_0.content,
    b1_0.create_date,
    b1_0.title,
    b1_0.writer
from
    board b1_0
where
    b1_0.title like ? escape '\\'
    or b1_0.content like ? escape '\\'
```

where 절에 두개의 조건이 OR  
연산으로 결합됨



# 쿼리 메서드(Query Method)

- 데이터 정렬하기

데이터를 정렬해서 조회하기 위해서는 "OrderBy" + 변수 + "Asc Or Desc"를 이용함.

게시글 제목에 특정 단어가 포함된 글 목록을 내림차순으로 조회하기

```
//글 제목에 특정 단어가 포함된 글 목록을 내림차순으로 검색  
List<Board> findByIdContainingOrderByIdDesc(String searchKeyword);
```



# 쿼리 메서드(Query Method)

- 데이터 정렬하기

```
@Test
public void testFindByTitleContainingOrderByIdDesc() {
    List<Board> boardList =
        repository.findByTitleContainingOrderByIdDesc("18");

    Log.info("검색 결과");
    for(Board board : boardList) {
        Log.info("--->" + board.toString());
    }
}
```

```
select
    b1_0.id,
    b1_0.content,
    b1_0.create_date,
    b1_0.title,
    b1_0.writer
from
    board b1_0
where
    b1_0.title like ? escape '\\'
order by
    b1_0.id desc
```

where 절에 order by 절이 추가되었고 id가 내림차순으로 정렬되어 있음



# 쿼리 메서드(Query Method)

- 데이터 정렬하기

```
: 검색 결과
: --->Board(id=189, title=테스트 제목 189, writer=테스터, content=테스트 내용 189,
: --->Board(id=188, title=테스트 제목 188, writer=테스터, content=테스트 내용 188,
: --->Board(id=187, title=테스트 제목 187, writer=테스터, content=테스트 내용 187,
: --->Board(id=186, title=테스트 제목 186, writer=테스터, content=테스트 내용 186,
: --->Board(id=185, title=테스트 제목 185, writer=테스터, content=테스트 내용 185,
: --->Board(id=184, title=테스트 제목 184, writer=테스터, content=테스트 내용 184,
: --->Board(id=183, title=테스트 제목 183, writer=테스터, content=테스트 내용 183,
: --->Board(id=182, title=테스트 제목 182, writer=테스터, content=테스트 내용 182,
: --->Board(id=181, title=테스트 제목 181, writer=테스터, content=테스트 내용 181,
: --->Board(id=180, title=테스트 제목 180, writer=테스터, content=테스트 내용 180,
: --->Board(id=118, title=테스트 제목 118, writer=테스터, content=테스트 내용 118,
: --->Board(id=18, title=테스트 제목 18, writer=테스터, content=테스트 내용 18, cre
```





# 쿼리 메서드(Query Method)

- 검색 및 페이지 처리와 정렬

모든 쿼리 메소드는 마지막 파라미터로 페이징 처리를 위한 Pageable 인터페이스와 정렬을 처리하는 Sort 인터페이스를 추가할 수 있다.

## (1) 페이지 처리

한 화면에 5개의 데이터를 보여주기로 하고 첫 페이지에 해당하는 0번부터 열 개의 데이터만 조회하기

## (2) 정렬 처리

페이지 처리 시 Sort 클래스를 사용한다.

```
//제목 검색어가 포함된 게시글 목록을 페이지 처리하여 조회  
List<Board> findByTitleContaining(String searchKeyword, Pageable paging);
```



# 쿼리 메서드(Query Method)

- 검색 및 페이지 처리와 정렬

```
@Test
public void testFindByTitleContaining() {
    //0->첫 페이지 번호(pageNumber), 1 -> 둘째 페이지 번호
    //10->데이터 개수(pageSize)
    //Default - 오름차순 정렬
    //Pageable paging = PageRequest.of(1, 10);

    //내림차순 정렬
    Pageable paging = PageRequest.of(1, 10, Sort.Direction.DISC, "id");

    List<Board> boardList =
        repository.findByTitleContaining("제목", paging);

    Log.info("검색 결과");
    for(Board board : boardList) {
        Log.info("--->" + board.toString());
    }
}
```



# 쿼리 메서드(Query Method)

- 검색 및 페이지 처리와 정렬

```
select
    b1_0.id,
    b1_0.content,
    b1_0.create_date,
    b1_0.title,
    b1_0.writer
from
    board b1_0
where
    b1_0.title like ? escape '\\'
order by
    b1_0.id desc
limit
    ?, ?
```

mysql 사용함으로써 where 절에 limit  
예약어가 사용됨



# 쿼리 메서드(Query Method)

- 검색 및 페이지 처리와 정렬

```
: --->Board(id=11, title=테스트 제목 11, writer=테스터,  
: --->Board(id=12, title=테스트 제목 12, writer=테스터,  
: --->Board(id=13, title=테스트 제목 13, writer=테스터,  
: --->Board(id=14, title=테스트 제목 14, writer=테스터,  
: --->Board(id=15, title=테스트 제목 15, writer=테스터,  
: --->Board(id=16, title=테스트 제목 16, writer=테스터,  
: --->Board(id=17, title=테스트 제목 17, writer=테스터,  
: --->Board(id=18, title=테스트 제목 18, writer=테스터,  
: --->Board(id=19, title=테스트 제목 19, writer=테스터,  
: --->Board(id=20, title=테스트 제목 20, writer=테스터,
```

오름차순 정렬(2 페이지)

```
: --->Board(id=190, title=테스트 제목 190, writer=테스터,  
: --->Board(id=189, title=테스트 제목 189, writer=테스터,  
: --->Board(id=188, title=테스트 제목 188, writer=테스터,  
: --->Board(id=187, title=테스트 제목 187, writer=테스터,  
: --->Board(id=186, title=테스트 제목 186, writer=테스터,  
: --->Board(id=185, title=테스트 제목 185, writer=테스터,  
: --->Board(id=184, title=테스트 제목 184, writer=테스터,  
: --->Board(id=183, title=테스트 제목 183, writer=테스터,  
: --->Board(id=182, title=테스트 제목 182, writer=테스터,  
: --->Board(id=181, title=테스트 제목 181, writer=테스터,
```

내림차순 정렬(2 페이지)



# 쿼리 메서드(Query Method)

- 검색 및 페이지 처리와 정렬

(3) Page<T> 사용하기

List<T> 대신 **Page<T>**를 사용하면 다양한 기능을 추가로 이용할 수 있다.

```
//제목 검색어가 포함된 게시글 목록을 페이지 처리하여 조회 - Page<T> 사용
Page<Board> findByTitleContaining(String searchKeyword, Pageable paging);
```



# 쿼리 메서드(Query Method)

- 검색 및 페이지 처리와 정렬

```
@Test
public void testFindByTitleContaining() {
    Pageable paging = PageRequest.of(1, 10);

    //내림차순 정렬
    //Pageable paging = PageRequest.of(1, 10, Sort.Direction.DESC, "id");

    //페이지 정보 객체 생성
    Page<Board> pageInfo =
        repository.findByTitleContaining("제목", paging);

    Log.info("PAGE SIZE: " + pageInfo.getSize()); //10(페이지당 데이터수)
    Log.info("TOTAL PAGES: " + pageInfo.getTotalPages()); //전체 페이지수
    Log.info("TOTAL COUNT: " + pageInfo.getTotalElements()); //전체 데이터수

    //글 목록 반환 객체 생성
    List<Board> boardList = pageInfo.getContent();

    Log.info("검색 결과");
    for(Board board : boardList) {
        Log.info("--->" + board.toString());
    }
}
```



# 쿼리 메서드(Query Method)

- 검색 및 페이지 처리와 정렬

```
: PAGE SIZE: 10
: TOTAL PAGES: 20
: TOTAL COUNT: 200
: 검색 결과
: --->Board(id=11, title=테스트 제목 11, writer=테스터,
: --->Board(id=12, title=테스트 제목 12, writer=테스터,
: --->Board(id=13, title=테스트 제목 13, writer=테스터,
: --->Board(id=14, title=테스트 제목 14, writer=테스터,
: --->Board(id=15, title=테스트 제목 15, writer=테스터,
: --->Board(id=16, title=테스트 제목 16, writer=테스터,
: --->Board(id=17, title=테스트 제목 17, writer=테스터,
: --->Board(id=18, title=테스트 제목 18, writer=테스터,
: --->Board(id=19, title=테스트 제목 19, writer=테스터,
: --->Board(id=20, title=테스트 제목 20, writer=테스터,
```



# 연관 관계 매핑

## ● 연관 관계 매핑

관계형 데이터베이스에서 테이블 하나로 애플리케이션에서 사용하는 모든 데이터를 관리하는 것은 불가능하다. 따라서 관련된 데이터를 여러 테이블에 나누어 저장하고 테이블을 조인하여 데이터를 처리한다.

결국 테이블의 연관관계를 엔티티의 연관 관계로 매핑해야 하는데, 중요한 것은 테이블은 PK와 FK를 기반으로 연관 관계를 맺지만 객체는 참조 변수를 통해 연관 관계를 맺기 때문에 테이블의 연관과 엔티티의 연관이 정확하게 일치하지 않는다는 것이다.

## ● 연관관계 매핑 분류

- 단방향 매핑 – 일대일 단방향, 다대일 단방향, 일대다 단방향
- 양방향 매핑 – 양방향 매핑



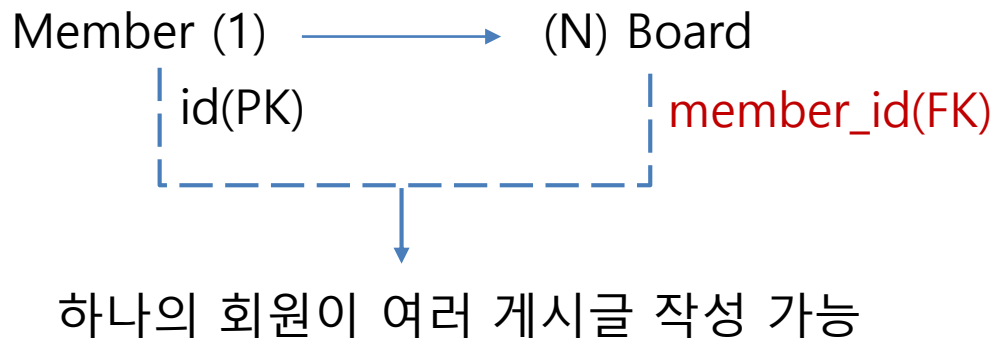


# 연관 관계 매핑

## ● 연관 관계 매핑

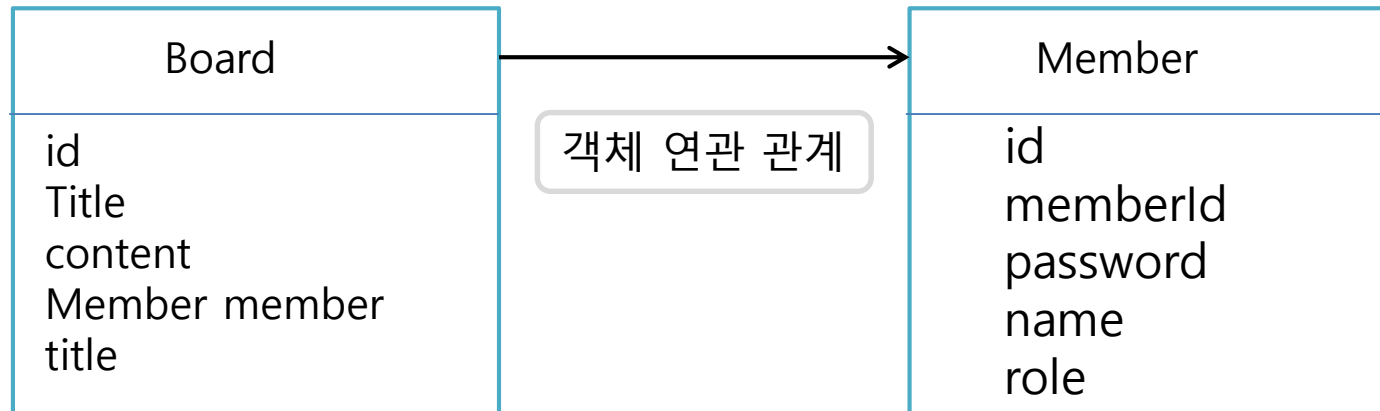
### (1) 다대일(N:1) 단방향 매핑하기

- 게시판과 회원이 있다
- 한명의 회원은 여러 개의 게시글을 작성할 수 있다
- 게시판과 회원은 다대일 관계이다.
- 게시글을 통해서 게시글을 작성한 회원 정보를 조회할 수 있다 .



# 연관 관계 매핑

## ● 다대일(N:1) 단방향 매핑하기



- 게시판 객체(Board)는 참조 변수(Board.member)를 통해 회원 객체(Member)와 관계를 맺는다.
- 게시판 객체와 회원 객체는 단방향 관계로서 게시판은 Board.member 변수를 통해 회원 정보를 알 수 있지만, 반대로 회원은 게시판에 대한 참조 변수를 가지지 않기 때문에 게시판 정보를 알 수 없다.



# 연관 관계 매핑

## ● Member 엔티티

```
@ToString
@Setter
@Getter
@Entity
public class Member {
    @Id //PRIMARY KEY(기본키)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id; //번호

    @Column(unique = true)
    private String memberId; //회원 아이디

    @Column(nullable = false)
    private String password; //비밀번호

    @Column(nullable = false)
    private String name; //이름

    private String role; //권한(USER, ADMIN)
}
```



# 연관 관계 매핑

- Member 테이블

```
create table member (  
    id int not null auto_increment,  
    member_id varchar(255) not null unique,  
    password varchar(255) not null,  
    name varchar(255) not null,  
    role varchar(255),  
    primary key (id)  
);
```



# 연관 관계 매핑

- 리포지터리 인터페이스

MemberRepository.java

```
import org.springframework.data.jpa.repository.JpaRepository;  
  
import com.springboot.entity.Member;  
  
public interface MemberRepository extends JpaRepository<Member, Integer>{  
  
}
```

BoardRepository.java

```
import com.springboot.entity.Board;  
  
public interface BoardRepository extends JpaRepository<Board, Integer>{  
  
}
```



# 연관 관계 매핑

## ● 연관 관계 매핑 테스트

```
@SpringBootTest
public class RelationMappingTest {

    @Autowired
    private MemberRepository memberRepo;

    @Test
    public void insertData() {
        //일반 회원
        Member member1 = new Member();
        member1.setMemberId("member1");
        member1.setPassword("m1111");
        member1.setName("뽀로로");
        member1.setRole("User");
        memberRepo.save(member1);

        //관리자
        Member member2 = new Member();
        member2.setMemberId("member2");
        member2.setPassword("m2222");
        member2.setName("아기상어");
        member2.setRole("ADMIN");
        memberRepo.save(member2);
    }
}
```



# 연관 관계 매핑

## ● Board 엔티티

```
@Entity
public class Board {
    @Id //PRIMARY KEY(기본키) - 없으면 에러발생
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id; //번호

    @Column(nullable = false)
    private String title; //글 제목

    @Column(length = 2000, nullable = false)
    private String content; //글 내용

    @CreationTimestamp //시간 자동 저장
    private Timestamp createDate; //작성일

    //다대일 연관 관계 매핑(Board(다) : Member(일))
    //FetchType.LAZY - 게시글 조회시 로딩
    //오류 발생시 - FetchType.EAGER 사용
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn() //조인 - 참조 관계
    private Member member;
}
```



# 연관 관계 매핑

## ● Board 엔티티

- 다대일 관계를 설정하기 위해 Member 타입의 member 변수를 추가했고, @ManyToOne 어노테이션을 사용함
- @JoinColumn은 name 속성을 통해 외래키 칼럼을 매핑함
- fetch 전략중 EAGER(즉시로딩)는 회원 정보를 조회할 때 게시판 정보도 같이 조회할 때 사용하고, LAZY(지연로딩)은 즉시 로딩하지 않고 필요할 때만 조회한다.(LAZY 권장)





# 연관 관계 매핑

## ● Board 테이블

```
create table board (  
    id int not null auto_increment,  
    title varchar(255) not null,  
    content varchar(2000) not null,  
    created_date datetime(6),  
    member_id int,  
    primary key (id),  
    constraint fk_board_member foreign key (member_id)  
    references member (id)  
);
```



# 연관 관계 매핑

- 연관 관계 매핑 테스트

```
@SpringBootTest
public class RelationMappingTest {

    @Autowired
    private MemberRepository memberRepo;

    @Autowired
    private BoardRepository boardRepo;

    @Test
    public void insertData() {
        //일반 회원
        Member member1 = new Member();
        member1.setMemberId("member1");
        member1.setPassword("m1111");
        member1.setName("뽀로로");
        member1.setRole("User");
        memberRepo.save(member1);
    }
}
```



# 연관 관계 매핑

```
//관리자
Member member2 = new Member();
member2.setMemberId("member2");
member2.setPassword("m2222");
member2.setName("아기상어");
member2.setRole("ADMIN");
memberRepo.save(member2);

for(int i = 1; i <= 3; i++) {
    Board board = new Board();
    board.setTitle("뽀로로가 등록한 게시글 " + i);
    board.setContent("뽀로로가 등록한 게시글 내용 " + i);
    board.setMember(member1);
    boardRepo.save(board);
}

for(int i = 1; i <= 3; i++) {
    Board board = new Board();
    board.setTitle("아기상어가 등록한 게시글 " + i);
    board.setContent("아기상어가 등록한 게시글 내용 " + i);
    board.setMember(member2);
    boardRepo.save(board);
}
```



# 연관 관계 매핑

- MySQL DB

id	member_id	name	password	role
1	member1	뽀로로	m1111	User
2	member2	아기상어	m2222	ADMIN

id	member_id	created_date	content	title
1	1	2025-10-22 21:06:59.292583	뽀로로가 등록한 게시물 내용 1	뽀로로가 등록한 게시물 1
2	1	2025-10-22 21:06:59.309022	뽀로로가 등록한 게시물 내용 2	뽀로로가 등록한 게시물 2
3	1	2025-10-22 21:06:59.317079	뽀로로가 등록한 게시물 내용 3	뽀로로가 등록한 게시물 3
4	2	2025-10-22 21:06:59.324728	아기상어가 등록한 게시물 내용 1	아기상어가 등록한 게시물 1
5	2	2025-10-22 21:06:59.332897	아기상어가 등록한 게시물 내용 2	아기상어가 등록한 게시물 2
6	2	2025-10-22 21:06:59.341042	아기상어가 등록한 게시물 내용 3	아기상어가 등록한 게시물 3



# 연관 관계 매핑

## ● 다대일(N:1) 연관관계 테스트 하기

```
//특정 게시글을 작성한 회원 조회
//트랜잭션이 닫힌 후 Lazy 로딩이 되어 오류 발생함 - @Transactional 사용
@Transactional
@Test
public void testSelect() {
    Board board = boardRepo.findById(5).get();

    Log.info("[ " + board.getId() + "번 게시글 정보 ]");
    Log.info("제목\t: " + board.getTitle());
    Log.info("글쓴이\t: " + board.getMember().getName());
    Log.info("내용\t: " + board.getContent());
    Log.info("권한\t: " + board.getMember().getRole());
}
```

[ 5번 게시글 정보 ]

제목 : 아기상어가 등록한 게시글 2

글쓴이 : 아기상어

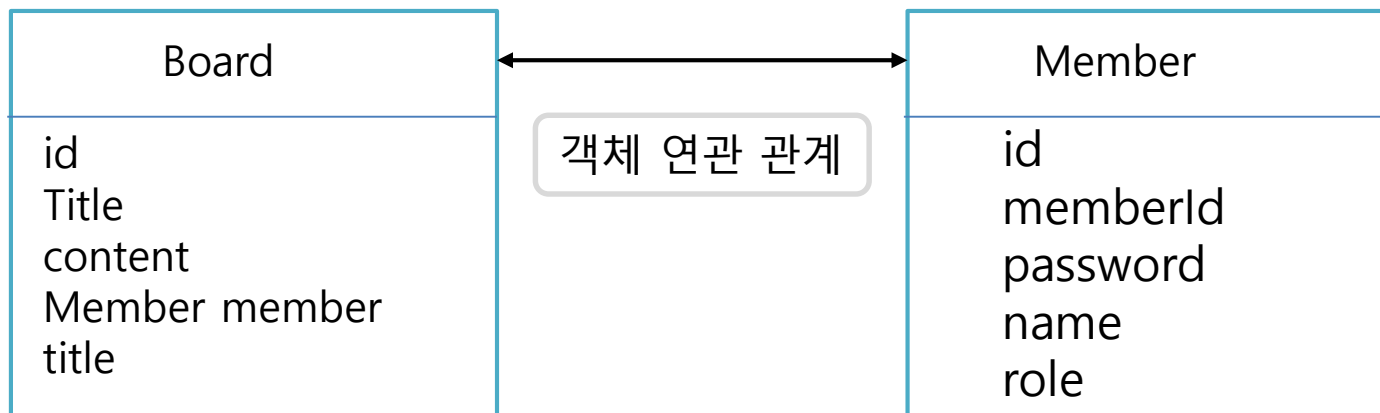
내용 : 아기상어가 등록한 게시글 내용 2

권한 : ADMIN



# 연관 관계 매핑

## ● 양방향 매핑 설정하기



- 지금까지 게시판에서 회원으로만 접근하는 다대일 단방향 매핑을 테스트 했다.
- 이번에는 반대 방향인 회원에서 게시판 정보를 접근할 수 있도록 관계를 추가하여 양방향 연관관계 매핑을 테스트 해본다.
- 회원은 게시판과 일대다 관계이다. 일대다 관계는 하나의 객체가 여러 객체와 연관 관계를 맺을 수 있으므로 당연히 **List 같은 컬렉션**을 사용해야 한다.



# 연관 관계 매핑

## ● 양방향 매핑 설정하기 – Member 엔티티

```
@ToString(exclude = "boards") //순환 참조 오류 방지
@Setter
@Getter
@Entity
public class Member {
    @Id //PRIMARY KEY(기본키)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id; //번호

    @Column(unique = true)
    private String memberId; //회원 아이디

    @Column(nullable = false)
    private String password; //비밀번호

    @Column(nullable = false)
    private String name; //이름

    private String role; //권한(USER, ADMIN)

    // Member(일) : Board(다)와 연관 매핑 :
    @OneToMany(mappedBy = "member", cascade = CascadeType.ALL)
    private List<Board> boards = new ArrayList<>();
}
```



# 연관 관계 매핑

## ● 양방향 매핑 설정하기 – Board 엔티티

```
@ToString(exclude = "member") //순환 참조 오류 방지
@Setter
@Getter
@Entity
public class Board {
    @Id //PRIMARY KEY(기본키) - 없으면 에러발생
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id; //번호

    @Column(nullable = false)
    private String title; //글 제목

    @Column(length = 2000, nullable = false)
    private String content; //글 내용

    @CreationTimestamp //시간 자동 저장
    private Timestamp createDate; //작성일

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn() //조인 - 참조 관계
    private Member member;
}
```





# 연관 관계 매핑

- 양방향 매핑 설정하기

Hibernate:

```
create table board (  
    id integer not null auto_increment,  
    member_id integer,  
    created_date datetime(6),  
    content varchar(2000) not null,  
    title varchar(255) not null,  
    primary key (id)  
) engine=InnoDB
```

Hibernate:

```
create table member (  
    id integer not null auto_increment,  
    member_id varchar(255),  
    name varchar(255) not null,  
    password varchar(255) not null,  
    role varchar(255),  
    primary key (id)  
) engine=InnoDB
```



# 연관 관계 매핑

- 양방향 매핑 설정하기

```
Hibernate:
    alter table member
        add constraint UK4rw879c4q7wrgi3v64cy73b17 unique (member_id)
Hibernate:
    alter table board
        add constraint FKsds8ox89wwf6aihinar49rmfy
        foreign key (member_id)
        references member (id)
```



# 연관 관계 매핑

## ● 양방향 매핑 설정

어노테이션	의미
OneToMany(mappedBy="member")	Board 엔티티의 member 필드에 의해 매핑됨 (연관관계의 주인은 Board)
cascade = CascadeType.ALL	회원이 저장/삭제될 때 관련 게시글도 함께 처리
@ToString(exclude = "boards")	Lombok 순환참조(무한 루프) 방지



# 연관 관계 매핑

- 양방향 매핑 테스트

```
//한명의 회원이 작성한 모든 게시글 조회
@Transactional
@Test
public void testGetBoardList() {
    //member1이 회원 가져오기
    Member member = memberRepo.findById("member1");

    Log.info(member.getName() + "이 작성한 게시글 목록");

    //member1이 작성한 모든 게시글 가져오기
    List<Board> boardList = member.getBoards();
    for(Board board : boardList)
        Log.info(board.toString());
}
```



# 연관 관계 매핑

- 양방향 매핑 테스트 오류

- ✓ 순환 참조 오류 방지

- ```
@ToString(exclude = "boards") //순환 참조 오류 방지
```

- ```
@ToString(exclude = "member") //순환 참조 오류 방지
```



# 연관 관계 매핑

## ● 양방향 매핑 테스트 결과

```
Board(id=1, title=뿌로로가 등록한 게시글 1, content=뿌로로가 등록한 게시글 내용 1,  
Board(id=2, title=뿌로로가 등록한 게시글 2, content=뿌로로가 등록한 게시글 내용 2,  
Board(id=3, title=뿌로로가 등록한 게시글 3, content=뿌로로가 등록한 게시글 내용 3,
```

