

3장. 스프링부트 JPA



ORM 하이버네이트



Spring Boot

스프링과 JPA

▪ 스프링과 JPA

데이터 베이스 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 JAP(Java Persistence API)** 이다
다시 말하면 ORM 기술을 Java 언어에 맞도록 스펙으로 정리한 것이라 할 수 있다..

JAP가 제공하는 인터페이스를 이용하여 데이터베이스를 처리하면 실제로는 JPA를 구현한 구현체가 동작하는 것이다.

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



스프링과 JPA

- SQL을 직접 다루는 기술

애플리케이션은 사용자가 입력한 데이터나 운용 과정에서 생성된 데이터를 재사용하기 위해 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.



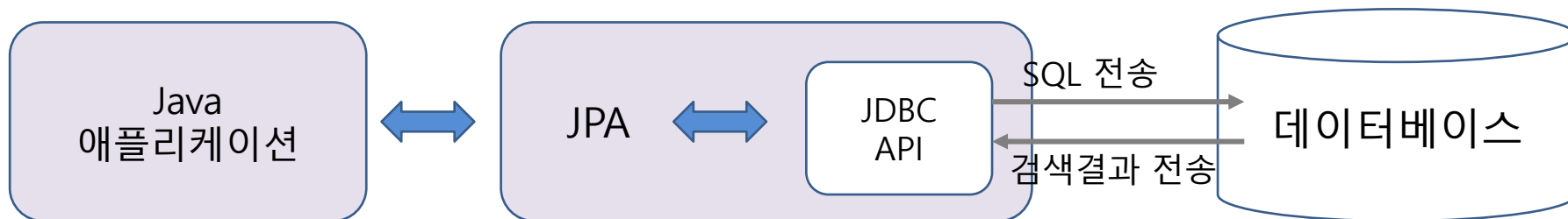
스프링과 JPA

▪ 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.



스프링과 JPA

▪ H2 데이터 베이스 사용하기



H2 Database Engine

Welcome to H2, the Java SQL database. The main features of H2 are:

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- Browser based Console application
- Small footprint: around 2.5 MB jar file size

Download

Version 2.1.214 (2022-06-13)





-  [Windows Installer \(6.7 MB\)](#)
-  [All Platforms \(zip, 9.5 MB\)](#)
- [All Downloads](#)

Support

[Stack Overflow \(tag H2\)](#)

[Google Group](#)

For non-technical issues, use:
dbsupport at h2database.com

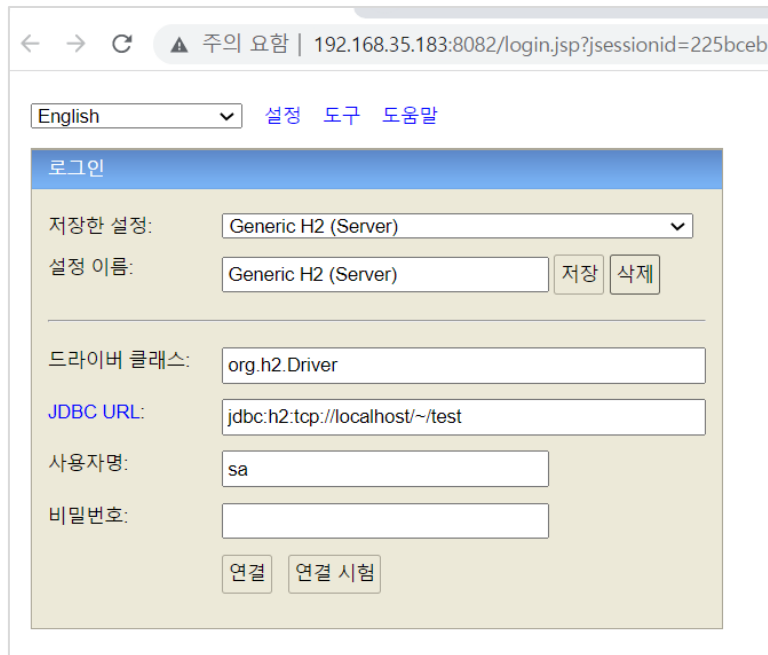
| Dev > h2-2022-06-13 > h2 > bin | |
|--|--------------------|
| 이름 | 수정한 날짜 |
|  h2.bat | 2022-06-13 오후 9:34 |
|  h2.sh | 2022-06-13 오후 9:34 |
|  h2-2.1.214.jar | 2022-06-13 오후 9:34 |
|  h2w.bat | 2022-06-13 오후 9:34 |

H2 설치 폴더 -> bin -> h2w.bat 파일 실행



스프링과 JPA

▪ H2 데이터 베이스 사용하기



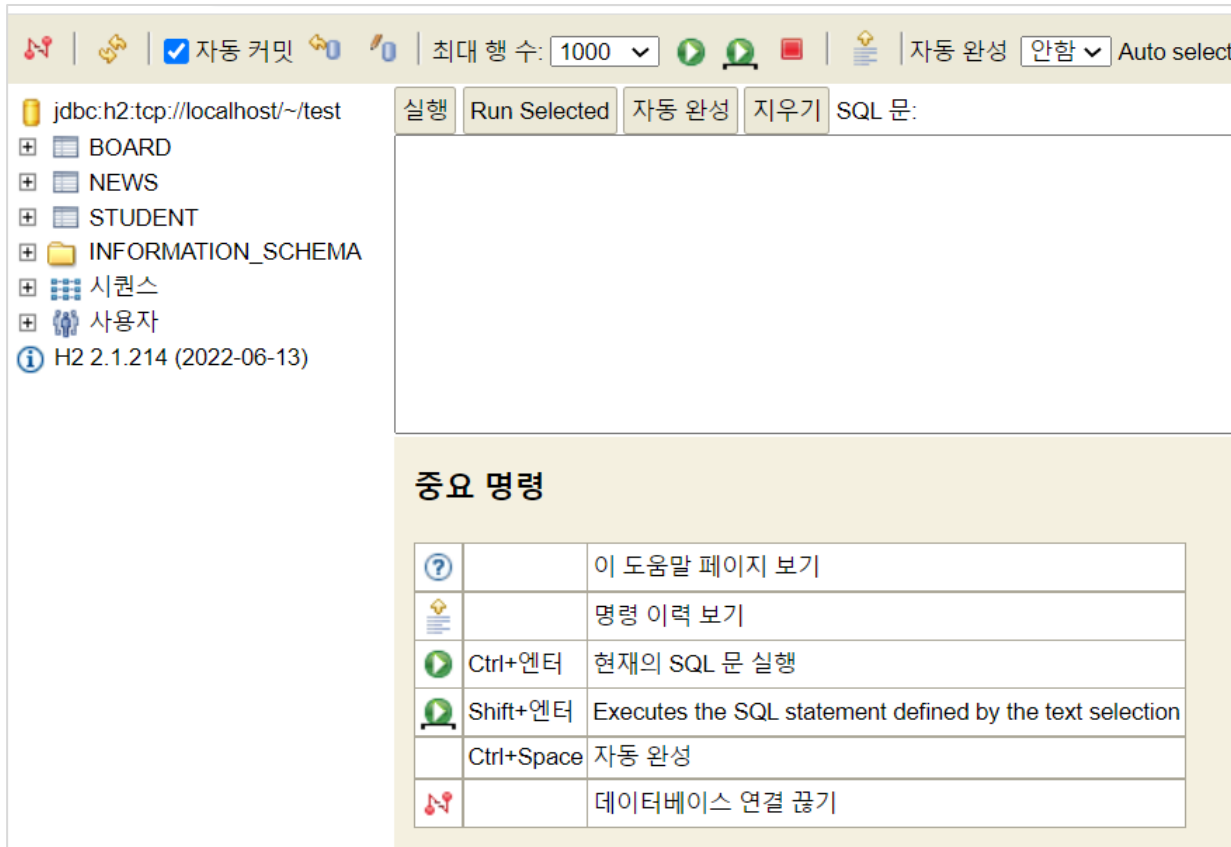
The screenshot shows the H2 database web interface. At the top, there's a navigation bar with 'English' and links for '설정' (Settings), '도구' (Tools), and '도움말' (Help). Below this is a '로그인' (Login) section. It contains several fields: '저장한 설정:' (Saved settings) with a dropdown menu showing 'Generic H2 (Server)'; '설정 이름:' (Settings name) with a text input field also containing 'Generic H2 (Server)' and buttons for '저장' (Save) and '삭제' (Delete); '드라이버 클래스:' (Driver class) with a text input field containing 'org.h2.Driver'; 'JDBC URL:' with a text input field containing 'jdbc:h2:tcp://localhost/~ /test'; '사용자명:' (Username) with a text input field containing 'sa'; and '비밀번호:' (Password) with an empty text input field. At the bottom of the login section are buttons for '연결' (Connect) and '연결 시험' (Test connection).

| 항목 | 설정 값 |
|-------------|---------------------------------|
| driverClass | org.h2.Driver |
| jdbc url | jdbc:h2:tcp://localhost/~ /test |
| 사용자명 | sa |
| 비밀번호 | 없음 |



스프링과 JPA

▪ H2 데이터 베이스 사용하기



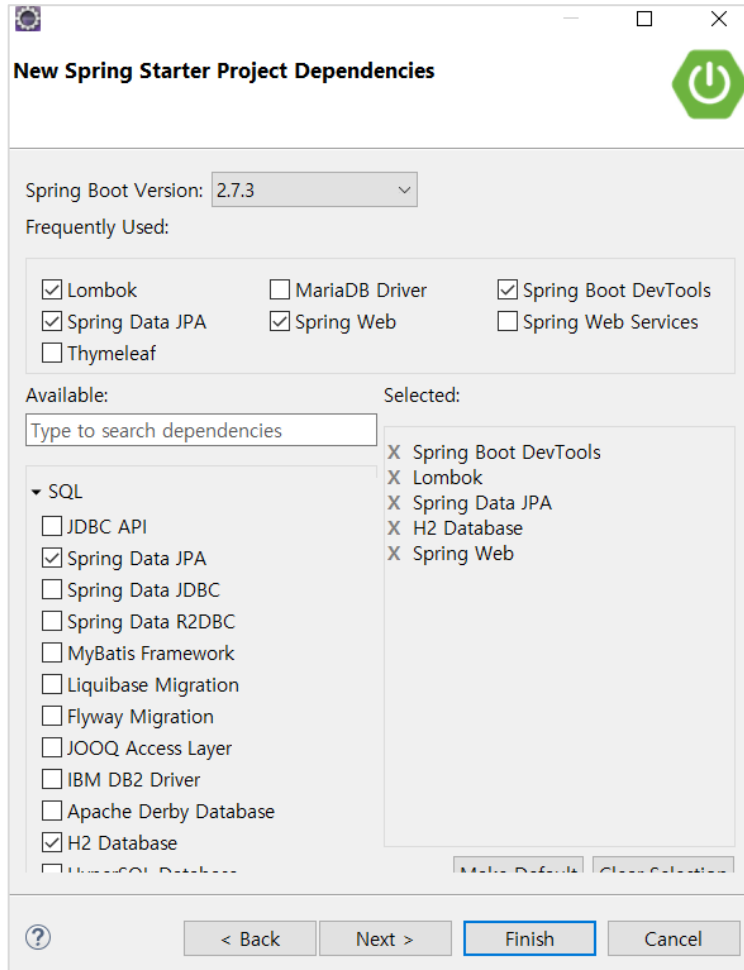
The screenshot shows the H2 Database console interface. The top toolbar includes icons for connection, schema, auto-commit, and execution, along with a dropdown for '최대 행 수' (Maximum rows) set to 1000. The left sidebar displays the database structure: jdbc:h2:tcp://localhost/~/.test, BOARD, NEWS, STUDENT, INFORMATION_SCHEMA, 시퀀스 (Sequences), 사용자 (Users), and H2 2.1.214 (2022-06-13). The main area contains buttons for '실행' (Execute), 'Run Selected', '자동 완성' (Auto complete), and '지우기' (Clear), followed by a text input for 'SQL 문:' (SQL statement). Below this is a section titled '중요 명령' (Important commands) with a table of shortcuts.

| Icon | Shortcut | Description |
|------|------------|--|
| ? | | 이 도움말 페이지 보기 |
| 📄 | | 명령 이력 보기 |
| ▶ | Ctrl+엔터 | 현재의 SQL 문 실행 |
| ▶ | Shift+엔터 | Executes the SQL statement defined by the text selection |
| | Ctrl+Space | 자동 완성 |
| 🔌 | | 데이터베이스 연결 끊기 |



스프링 데이터 JPA

■ 프로젝트 생성 및 기본 설정



The image shows the 'New Spring Starter Project Dependencies' dialog box. It has a title bar with a green power icon. The 'Spring Boot Version' is set to 2.7.3. Under 'Frequently Used', the following dependencies are checked: Lombok, Spring Data JPA, Thymeleaf, MariaDB Driver, Spring Web, and Spring Boot DevTools. The 'Available' list on the left includes SQL-related dependencies, with 'Spring Data JPA' and 'H2 Database' checked. The 'Selected' list on the right includes Spring Boot DevTools, Lombok, Spring Data JPA, H2 Database, and Spring Web. At the bottom, there are buttons for '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

New Spring Starter Project Dependencies

Spring Boot Version: 2.7.3

Frequently Used:

- ☒ Lombok
- ☐ MariaDB Driver
- ☒ Spring Boot DevTools
- ☒ Spring Data JPA
- ☒ Spring Web
- ☐ Spring Web Services
- ☐ Thymeleaf

Available:

Type to search dependencies

- SQL
 - ☐ JDBC API
 - ☒ Spring Data JPA
 - ☐ Spring Data JDBC
 - ☐ Spring Data R2DBC
 - ☐ MyBatis Framework
 - ☐ Liquibase Migration
 - ☐ Flyway Migration
 - ☐ JOOQ Access Layer
 - ☐ IBM DB2 Driver
 - ☐ Apache Derby Database
 - ☒ H2 Database
 - ☐ Microsoft SQL Database

Selected:

- X Spring Boot DevTools
- X Lombok
- X Spring Data JPA
- X H2 Database
- X Spring Web

< Back Next > Finish Cancel

[New] -> [Spring Starter Project]

Name – SpringJPA,

Packing – Jar

Java Version – 8

Group – com.boot,

Package – com.boot

- 모듈 추가

DevTools, Lombok, Spring Web,
Spring Data JPA, H2 Database



스프링 데이터 JPA

프로젝트 생성 및 기본 설정

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

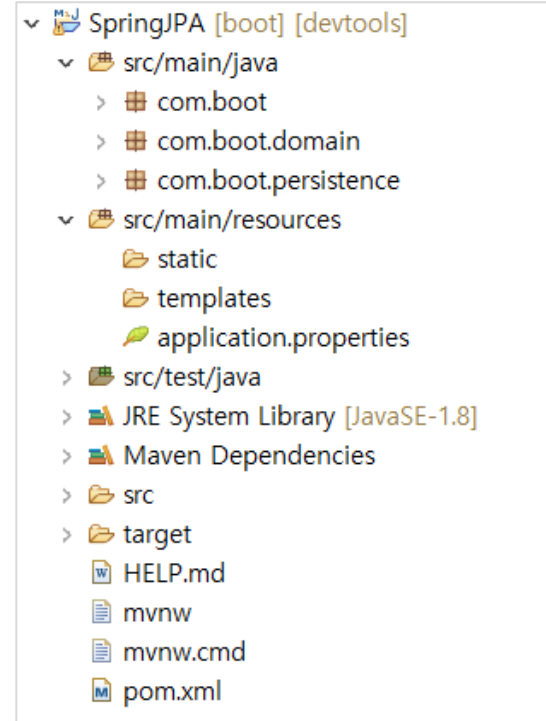
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



스프링 데이터 JPA

- JPA 기본 설정

```
# DataSource Setting
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:tcp://localhost/~ /test
spring.datasource.username=sa
spring.datasource.password=

# JPA Setting
spring.jpa.hibernate.ddl-auto=create
spring.jpa.generate-ddl=false
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.format_sql=true
```

application.properties



스프링 데이터 JPA

- 엔티티 매핑과 리포지터리 작성

| 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

- 엔티티 매핑과 리포지터리 작성

- (1) 엔티티 클래스 매핑

```
@ToString
@Setter
@Getter
@Entity
public class Board {

    @Id@GeneratedValue
    private Long seq;

    private String title;
    private String writer;
    private String content;

    private Date createDate;
    private Long cnt;
}
```



스프링 데이터 JPA

- 엔티티 매핑과 리포지터리 작성

(2) 테이블 생성 확인 – 앱 실행

```
@SpringBootApplication
public class SpringJpaApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringJpaApplication.class, args);
    }
}
```

```
# WebApplication Type Setting
# spring.main.web-application-type=none
spring.main.web-application-type=servlet

## Server Setting(포트번호 변경)
server.port = 8181
```



스프링 데이터 JPA

- 엔티티 매핑과 리포지터리 작성

- (2) 테이블 생성 확인

```
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:
```

```
create table board (
  seq bigint not null,
  cnt bigint,
  content varchar(255),
  create_date timestamp,
  title varchar(255),
  writer varchar(255),
  primary key (seq)
)
```



스프링 데이터 JPA

■ 엔티티 매핑과 리포지터리 작성

(2) 테이블 생성 확인

The screenshot shows the H2 database console interface. The left pane displays the database structure for 'jdbc:h2:top://localhost/~test'. The tables listed are BOARD, NEWS, STUDENT, INFORMATION_SCHEMA, 시퀀스, and 사용자. The 'NEWS' table is highlighted with a red dashed box. The right pane shows the SQL editor with buttons for '실행' (Execute), 'Run Selected', '자동 완성' (Auto Complete), '지우기' (Clear), and 'SQL 문:' (SQL Statement). Below the editor is a '중요 명령' (Important Commands) section with a table of shortcuts.

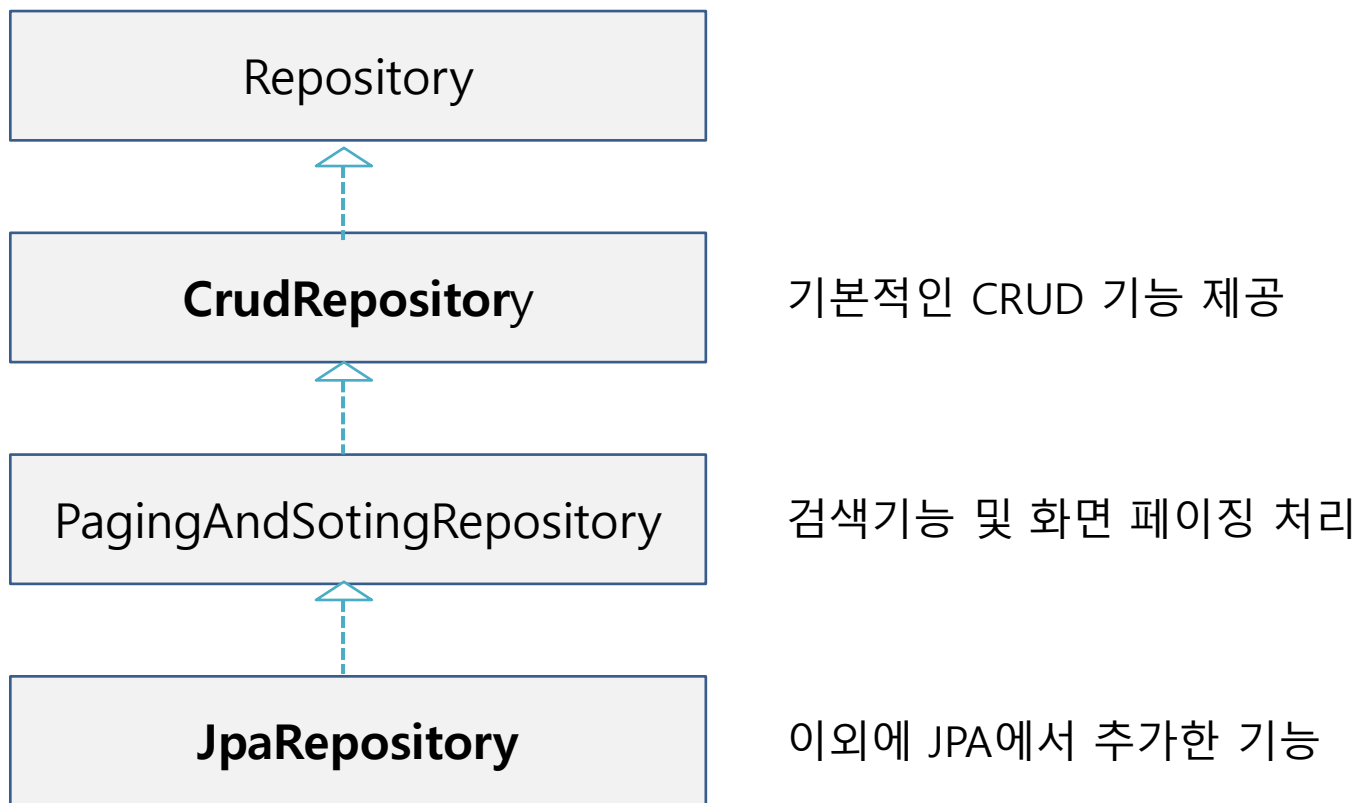
| Icon | Shortcut | Description |
|------|------------|--|
| ? | | 이 도움말 페이지 보기 |
| 📋 | | 명령 이력 보기 |
| ▶ | Ctrl+엔터 | 현재의 SQL 문 실행 |
| ▶ | Shift+엔터 | Executes the SQL statement defined by the text selection |
| | Ctrl+Space | 자동 완성 |
| 🔌 | | 데이터베이스 연결 끊기 |



스프링 데이터 JPA

- 엔티티 매핑과 리포지터리 작성

(3) Repository 인터페이스 작성



스프링 데이터 JPA

- 엔티티 매핑과 리포지터리 작성

- (3) Repository 인터페이스 작성

CrudRepository<T, ID>

T : 엔티티의 클래스 타입

ID : 식별자(PK) 타입(@Id로 매핑한 식별자 변수의 자료형)

```
package com.boot.persistence;

import org.springframework.data.repository CrudRepository;

public interface BoardRepository extends CrudRepository<Board, Long>{

}
```

별도의 구현 클래스를 만들지 않고 인터페이스만 정의함으로써 기능을 사용할 수 있음



스프링 데이터 JPA

- 테스트 코드를 통한 CRUD

| 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

- CRUD 기능 테스트

- (1) 등록 기능 테스트

- 실행전 테이블 자동생성 기능을 update로 변경
 - 실행전 H2 DB 접속을 반드시 해야함

```
# JPA Setting
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=false
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.format_sql=true
```



스프링 데이터 JPA

- CRUD 기능 테스트

- (1) 등록 기능 테스트 – save() 메서드 사용

```
@SpringBootTest
public class BoardRepositoryTest {

    @Autowired
    private BoardRepository boardRepo;

    @Test
    public void testInsertBoard() {
        Board board = new Board();
        board.setTitle("첫 번째 게시글");
        board.setWriter("테스터");
        board.setContent("등록이 잘 되네요..");
        board.setCreateDate(new Date());
        board.setCnt(0L);

        boardRepo.save(board); //save() 메서드로 DB에 저장함
    }
}
```



스프링 데이터 JPA

▪ CRUD 기능 테스트

(1) 등록 기능 테스트

```
Hibernate:
    call next value for hibernate_sequence
Hibernate:
    insert
    into
        board
    (cnt, content, create_date, title, writer, seq)
    values
        (?, ?, ?, ?, ?, ?)
```

The screenshot shows a database client interface with a tree view on the left and a SQL editor/execution pane on the right. The tree view shows a connection to 'jdbc:h2:tcp://localhost/~:/test' with a schema named 'H2 2.1.214 (2022-06-13)'. The SQL editor contains the query 'SELECT * FROM board;'. The execution results are displayed in a table below the editor.

| SEQ | CNT | CONTENT | CREATE_DATE | TITLE | WRITER |
|-----|-----|-------------|-------------------------|----------|--------|
| 1 | 0 | 등록이 잘 되네요.. | 2022-09-17 11:18:55.119 | 첫 번째 게시글 | 테스터 |



스프링 데이터 JPA

- CRUD 기능 테스트

(2) 상세 조회 기능 테스트 – findById(seq) 사용

```
//상세 조회
@Test
public void testGetBoard() {
    Board board = boardRepo.findById(1L).get();
    Log.info(board.toString());
}
```

```
Board(seq=1, title=첫 번째 게시글, writer=테스터, content=등록이 잘 되네요..,
Closing JPA EntityManagerFactory for persistence unit 'default'
HikariPool-1 - Shutdown initiated...
```



스프링 데이터 JPA

- CRUD 기능 테스트

(3) 글 수정 기능 테스트 – findById(seq)로 조회후 save()로 재등록

```
//글 수정
@Test
public void testUpdateBoard() {
    Log.info("2번 게시글 조회");
    //2번 게시글 가져옴
    Board board = boardRepo.findById(2L).get();

    Log.info("2번 게시글 제목 수정");
    board.setTitle("제목을 수정합니다");
    //수정후 저장
    boardRepo.save(board);
}
```

```
Hibernate:
    update
      board
    set
      cnt=?,
      content=?,
      create_date=?,
      title=?,
      writer=?
    where
      seq=?
```

select * from board;

| SEQ | CNT | CONTENT | CREATE_DATE | TITLE | WRITER |
|-----|-----|-------------|-------------------------|-----------|--------|
| 1 | 0 | 등록이 잘 되네요.. | 2022-09-17 16:02:13.393 | 첫 번째 게시글 | 테스터 |
| 2 | 0 | 등록이 잘 되네요.. | 2022-09-17 16:02:42.378 | 제목을 수정합니다 | 테스터 |

(2 행, 1 ms)



스프링 데이터 JPA

- CRUD 기능 테스트

- (4) 글 삭제 기능 테스트 – deleteById(seq) 사용

```
//글 삭제
@Test
public void testDeleteBoard() {

    Log.info("1번 게시글 삭제");

    boardRepo.deleteById(1L);
}
```

Hibernate:
delete
from
board
where
seq=?

select * from board;

| SEQ | CNT | CONTENT | CREATE_DATE | TITLE | WRITER |
|-----|-----|-------------|-------------------------|-----------|--------|
| 2 | 0 | 등록이 잘 되네요.. | 2022-09-17 16:02:42.378 | 제목을 수정합니다 | 테스터 |

(1 row, 2 ms)

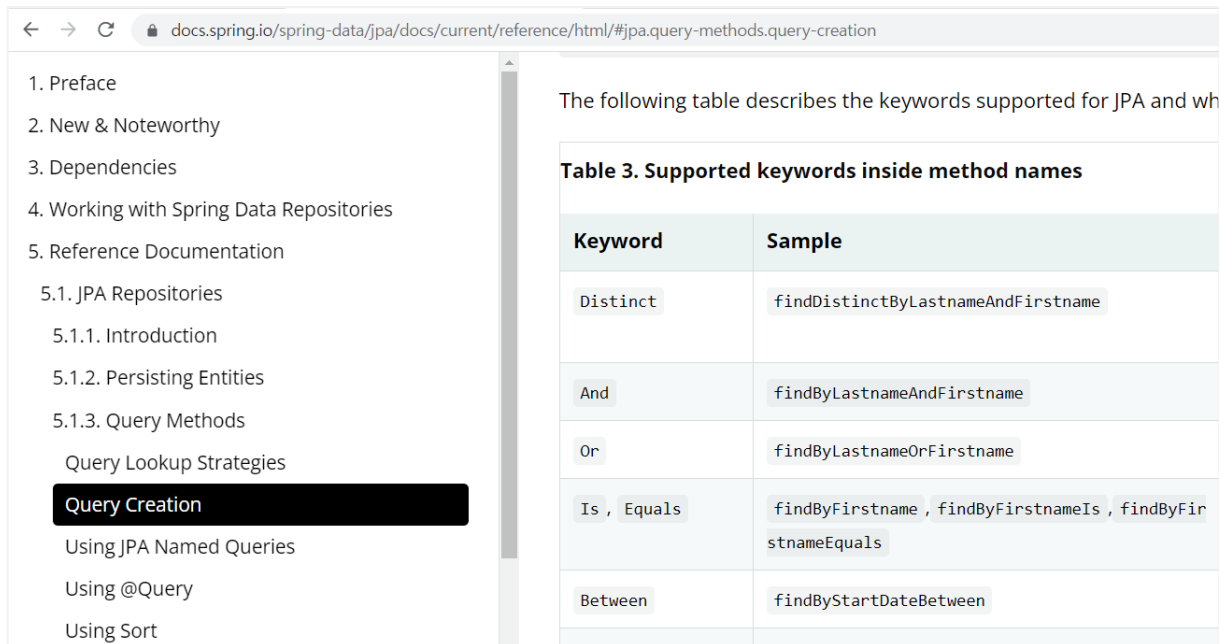


스프링 데이터 JPA

- 쿼리 메소드란?

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



스프링 데이터 JPA

- 쿼리 메소드 사용하기

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

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

```
public interface BoardRepository extends CrudRepository<Board, Long>{  
  
    //쿼리 메소드 - 글 제목 검색  
    List<Board> findByTitle(String searchKeyword);  
  
}
```



스프링 데이터 JPA

- 쿼리 메소드 사용하기

```
@Slf4j
@SpringBootTest
public class QueryMethodTest {

    @Autowired
    private BoardRepository boardRepo;

    /*BeforeEach의 dataPrepare()는 테스트 메소드가
    실행되기 전에 동작함.. 데이터 200개 저장*/
    @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 Date());
            board.setCnt(0L);

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



스프링 데이터 JPA

- 쿼리 메소드 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(seq=12, title=테스트 제목10, writer=테스터, content=테스트 내용 10,



스프링 데이터 JPA

- 쿼리 메소드 사용하기

```
select * from board;
```

| SEQ | CNT | CONTENT | CREATE_DATE | TITLE | WRITER |
|-----|-----|-----------|-------------------------|-----------|--------|
| 3 | 0 | 테스트 내용 1 | 2022-09-22 04:53:54.084 | 테스트 제목 1 | 테스터 |
| 4 | 0 | 테스트 내용 2 | 2022-09-22 04:53:54.162 | 테스트 제목 2 | 테스터 |
| 5 | 0 | 테스트 내용 3 | 2022-09-22 04:53:54.169 | 테스트 제목 3 | 테스터 |
| 6 | 0 | 테스트 내용 4 | 2022-09-22 04:53:54.174 | 테스트 제목 4 | 테스터 |
| 7 | 0 | 테스트 내용 5 | 2022-09-22 04:53:54.178 | 테스트 제목 5 | 테스터 |
| 8 | 0 | 테스트 내용 6 | 2022-09-22 04:53:54.182 | 테스트 제목 6 | 테스터 |
| 9 | 0 | 테스트 내용 7 | 2022-09-22 04:53:54.185 | 테스트 제목 7 | 테스터 |
| 10 | 0 | 테스트 내용 8 | 2022-09-22 04:53:54.189 | 테스트 제목 8 | 테스터 |
| 11 | 0 | 테스트 내용 9 | 2022-09-22 04:53:54.191 | 테스트 제목 9 | 테스터 |
| 12 | 0 | 테스트 내용 10 | 2022-09-22 04:53:54.194 | 테스트 제목 10 | 테스터 |
| 13 | 0 | 테스트 내용 11 | 2022-09-22 04:53:54.197 | 테스트 제목 11 | 테스터 |
| 14 | 0 | 테스트 내용 12 | 2022-09-22 04:53:54.2 | 테스트 제목 12 | 테스터 |
| 15 | 0 | 테스트 내용 13 | 2022-09-22 04:53:54.202 | 테스트 제목 13 | 테스터 |



스프링 데이터 JPA

- 쿼리 메소드 사용하기

- (1) LIKE 연산자 사용하기

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

```
public interface BoardRepository extends CrudRepository<Board, Long>{  
  
    //쿼리 메소드 - 글 제목 검색  
    List<Board> findByTitle(String searchKeyword);  
  
    //특정 단어가 포함된 목록 검색  
    List<Board> findByContentContaining(String searchKeyword);  
}
```



스프링 데이터 JPA

- 쿼리 메소드 사용하기

```
Hibernate:
select
    board0_.seq as seq1_0_,
    board0_.cnt as cnt2_0_,
    board0_.content as content3_0_,
    board0_.create_date as create_d4_0_,
    board0_.title as title5_0_,
    board0_.writer as writer6_0_
from
    board board0_
where
    board0_.content like ? escape ?
```

where 절에 like 연산자가 사용됨

검색 결과

```
--->Board(seq=19, title=테스트 제목 17, writer=테스터, content=테스트 내용 17, cre
--->Board(seq=119, title=테스트 제목 117, writer=테스터, content=테스트 내용 117,
--->Board(seq=172, title=테스트 제목 170, writer=테스터, content=테스트 내용 170,
--->Board(seq=173, title=테스트 제목 171, writer=테스터, content=테스트 내용 171,
--->Board(seq=174, title=테스트 제목 172, writer=테스터, content=테스트 내용 172,
--->Board(seq=175, title=테스트 제목 173, writer=테스터, content=테스트 내용 173,
--->Board(seq=176, title=테스트 제목 174, writer=테스터, content=테스트 내용 174,
--->Board(seq=177, title=테스트 제목 175, writer=테스터, content=테스트 내용 175,
--->Board(seq=178, title=테스트 제목 176, writer=테스터, content=테스트 내용 176,
--->Board(seq=179, title=테스트 제목 177, writer=테스터, content=테스트 내용 177,
--->Board(seq=180, title=테스트 제목 178, writer=테스터, content=테스트 내용 178,
--->Board(seq=181, title=테스트 제목 179, writer=테스터, content=테스트 내용 179,
```



스프링 데이터 JPA

- 쿼리 메소드 사용하기

- (2) 여러 조건 사용하기

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

```
public interface BoardRepository extends CrudRepository<Board, Long>{

    //쿼리 메소드 - 글 제목 검색
    List<Board> findByTitle(String searchKeyword);

    //특정 단어가 포함된 목록 검색
    List<Board> findByContentContaining(String searchKeyword);

    //제목 또는 내용에 특정 단어가 포함된 목록 검색
    List<Board> findByTitleContainingOrContentContaining(String title,
                                                         String content);
}
```



스프링 데이터 JPA

- 쿼리 메소드 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
    board0_.seq as seq1_0_,
    board0_.cnt as cnt2_0_,
    board0_.content as content3_0_,
    board0_.create_date as create_d4_0_,
    board0_.title as title5_0_,
    board0_.writer as writer6_0_
from
    board board0_
where
    board0_.title like ? escape ?
    or board0_.content like ? escape ?
```

where 절에 두개의 조건이 OR
연산으로 결합됨



스프링 데이터 JPA

- 쿼리 메소드 사용하기

- (3) 데이터 정렬하기

데이터를 정렬해서 조회하기 위해서는 "OrderBy" + 변수 + "Asc Or Desc"를 이용함
게시글 제목에 특정단어가 포함된 글 목록을 내림차순으로 조회하기

```
public interface BoardRepository extends CrudRepository<Board, Long>{  
  
    //글 제목에 특정 단어가 포함된 글 목록을 내림차순으로 조회  
    List<Board> findByTitleContainingOrderBySeqDesc(String searchKeyword);  
  
}
```



스프링 데이터 JPA

- 쿼리 메소드 사용하기

```
@Test
public void testFindByTitleContainingOrderBySeqDesc() {
    List<Board> boardList =
        boardRepo.findByTitleContainingOrderBySeqDesc("18");

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

```
select
    board0_.seq as seq1_0_,
    board0_.cnt as cnt2_0_,
    board0_.content as content3_0_,
    board0_.create_date as create_d4_0_,
    board0_.title as title5_0_,
    board0_.writer as writer6_0_
from
    board board0_
where
    board0_.title like ? escape ?
order by
    board0_.seq desc
```

where 절에 order by 절이 추가되었고 seq 가 내림차순으로 정렬되어 있음



스프링 데이터 JPA

- 쿼리 메소드 사용하기

```
: 검색 결과
: --->Board(seq=191, title=테스트 제목 189, writer=테스터,
: --->Board(seq=190, title=테스트 제목 188, writer=테스터,
: --->Board(seq=189, title=테스트 제목 187, writer=테스터,
: --->Board(seq=188, title=테스트 제목 186, writer=테스터,
: --->Board(seq=187, title=테스트 제목 185, writer=테스터,
: --->Board(seq=186, title=테스트 제목 184, writer=테스터,
: --->Board(seq=185, title=테스트 제목 183, writer=테스터,
: --->Board(seq=184, title=테스트 제목 182, writer=테스터,
: --->Board(seq=183, title=테스트 제목 181, writer=테스터,
: --->Board(seq=182, title=테스트 제목 180, writer=테스터,
: --->Board(seq=120, title=테스트 제목 118, writer=테스터,
: --->Board(seq=20, title=테스트 제목 18, writer=테스터, cc
n : Closing JPA EntityManagerFactory for persistence uni
```



스프링 데이터 JPA

- 페이징과 정렬 처리하기

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

(1) 페이징 처리

한 화면에 열 개의 데이터를 보여주기로 하고 첫 페이지에 해당하는 1번부터 열 개의 데이터만 조회하기

```
public interface BoardRepository extends CrudRepository<Board, Long>{

    //글 제목에 특정 단어가 포함된 글 목록을 내림차순으로 조회
    List<Board> findByTitleContainingOrderBySeqDesc(String searchKeyword);

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



스프링 데이터 JPA

- 페이징과 정렬 처리하기

```
@Test
public void testFindByTitleContaining() {
    //0은 페이지번호(실제 1페이지임), 10은 데이터의 개수
    Pageable paging = PageRequest.of(0, 10);

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

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

```
select
    board0_.seq as seq1_0_,
    board0_.cnt as cnt2_0_,
    board0_.content as content3_0_,
    board0_.create_date as create_d4_0_,
    board0_.title as title5_0_,
    board0_.writer as writer6_0_
from
    board board0_
where
    board0_.title like ? escape ? limit ?
```

H2 데이터베이스를 사용함으로써 where 절에 limit 예약어가 사용됨



스프링 데이터 JPA

- 페이징과 정렬 처리하기

```
: 검색 결과
: --->Board(seq=3, title=테스트 제목 1, writer=테스터, content=테스트 내용 1,
: --->Board(seq=4, title=테스트 제목 2, writer=테스터, content=테스트 내용 2,
: --->Board(seq=5, title=테스트 제목 3, writer=테스터, content=테스트 내용 3,
: --->Board(seq=6, title=테스트 제목 4, writer=테스터, content=테스트 내용 4,
: --->Board(seq=7, title=테스트 제목 5, writer=테스터, content=테스트 내용 5,
: --->Board(seq=8, title=테스트 제목 6, writer=테스터, content=테스트 내용 6,
: --->Board(seq=9, title=테스트 제목 7, writer=테스터, content=테스트 내용 7,
: --->Board(seq=10, title=테스트 제목 8, writer=테스터, content=테스트 내용 8,
: --->Board(seq=11, title=테스트 제목 9, writer=테스터, content=테스트 내용 9,
: --->Board(seq=12, title=테스트 제목 10, writer=테스터, content=테스트 내용 1
```



스프링 데이터 JPA

- 페이징과 정렬 처리하기

- (2) 정렬 처리

페이징 처리 시 Sort 클래스를 사용한다.

```
@Test
public void testFindByTitleContaining() {
    //0은 페이지번호(실제 1페이지임), 10은 데이터의 개수
    //Pageable paging = PageRequest.of(0, 10);

    //Sort 클래스를 사용하여 첫 페이지를 내림차순 정렬함
    Pageable paging = PageRequest.of(0, 10, Sort.Direction.DESC, "seq");

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

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



스프링 데이터 JPA

■ 페이징과 정렬 처리하기

```
select
    board0_.seq as seq1_0_,
    board0_.cnt as cnt2_0_,
    board0_.content as content3_0_,
    board0_.create_date as create_d4_0_,
    board0_.title as title5_0_,
    board0_.writer as writer6_0_
from
    board board0_
where
    board0_.title like ? escape ?
order by
    board0_.seq desc limit ?
```

: 검색 결과

```
: --->Board(seq=202, title=테스트 제목 200, writer=테스터, content=테스트 내용 200,
: --->Board(seq=201, title=테스트 제목 199, writer=테스터, content=테스트 내용 199,
: --->Board(seq=200, title=테스트 제목 198, writer=테스터, content=테스트 내용 198,
: --->Board(seq=199, title=테스트 제목 197, writer=테스터, content=테스트 내용 197,
: --->Board(seq=198, title=테스트 제목 196, writer=테스터, content=테스트 내용 196,
: --->Board(seq=197, title=테스트 제목 195, writer=테스터, content=테스트 내용 195,
: --->Board(seq=196, title=테스트 제목 194, writer=테스터, content=테스트 내용 194,
: --->Board(seq=195, title=테스트 제목 193, writer=테스터, content=테스트 내용 193,
: --->Board(seq=194, title=테스트 제목 192, writer=테스터, content=테스트 내용 192,
: --->Board(seq=193, title=테스트 제목 191, writer=테스터, content=테스트 내용 191,
```



스프링 데이터 JPA

- 페이징과 정렬 처리하기

- (3) Page<T> 사용하기

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

```
public interface BoardRepository extends CrudRepository<Board, Long>{  
  
    //제목 검색어가 포함된 게시글 목록을 검색하되 페이징 처리하여 조회  
    //List<Board> findByTitleContaining(String searchKeyword, Pageable paging);  
  
    Page<Board> findByTitleContaining(String searchKeyword, Pageable paging);  
}
```



스프링 데이터 JPA

- 페이징과 정렬 처리하기

```
@Test
public void testFindByTitleContaining() {
    Pageable paging = PageRequest.of(0, 10, Sort.Direction.DESC, "seq");

    Page<Board> pageInfo = boardRepo.findByTitleContaining("제목", paging);

    System.out.println("PAGE SIZE : " + pageInfo.getSize());
    System.out.println("TOTAL PAGES : " + pageInfo.getTotalPages());
    System.out.println("TOTAL COUNT : " + pageInfo.getTotalElements());
    System.out.println("NEXT : " + pageInfo.nextPageable());

    List<Board> boardList = pageInfo.getContent();

    System.out.println("검색 결과");
    for(Board board : boardList) {
        System.out.println("--->" + board.toString());
    }
}
```



스프링 데이터 JPA

- 페이징과 정렬 처리하기

```
Hibernate:
  select
    count(board0_.seq) as col_0_0_
  from
    board board0_
  where
    board0_.title like ? escape ?
PAGE SIZE : 10
TOTAL PAGES : 20
TOTAL COUNT : 200
NEXT : Page request [number: 1, size 10, sort: seq: DESC]
```



스프링 데이터 JPA

@Query 어노테이션

조금 복잡한 쿼리를 사용한다거나 연관관계에 기반한 조인(JOIN) 검색을 처리하기 위해 JPQL(Java Persistence Query Language)을 사용해야 하는데 이때 @Query()에 파라미터로 설정한다.

(1) 위치기반 파라미터 사용하기

```
public interface BoardRepository extends CrudRepository<Board, Long>{  
  
    //제목에 특정 단어가 포함된 글 목록을 내림차순으로 조회  
    @Query("SELECT b FROM Board b WHERE b.title LIKE %?1% ORDER BY b.seq DESC")  
    List<Board> queryAnnotationTest1(String searchKeyword);  
}
```

'?1'은 첫번째 파라미터를 의미한다. 매개변수로 받은 searchKeyword가 첫 번째 파라미터 값으로 바인딩 된다.



스프링 데이터 JPA

@Query 어노테이션

```
@Slf4j
@SpringBootTest
public class QueryAnnotationTest {

    @Autowired
    private BoardRepository boardRepo;

    @Test
    public void testQueryAnnotationTest1() {
        List<Board> boardList =
            boardRepo.queryAnnotationTest1("테스트 제목 10");

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



스프링 데이터 JPA

@Query 어노테이션

검색 결과

```
--->Board(seq=109, title=테스트 제목 109, writer=테스터, content=테스트 내용 109,  
--->Board(seq=108, title=테스트 제목 108, writer=테스터, content=테스트 내용 108,  
--->Board(seq=107, title=테스트 제목 107, writer=테스터, content=테스트 내용 107,  
--->Board(seq=106, title=테스트 제목 106, writer=테스터, content=테스트 내용 106,  
--->Board(seq=105, title=테스트 제목 105, writer=테스터, content=테스트 내용 105,  
--->Board(seq=104, title=테스트 제목 104, writer=테스터, content=테스트 내용 104,  
--->Board(seq=103, title=테스트 제목 103, writer=테스터, content=테스트 내용 103,  
--->Board(seq=102, title=테스트 제목 102, writer=테스터, content=테스트 내용 102,  
--->Board(seq=101, title=테스트 제목 101, writer=테스터, content=테스트 내용 101,  
--->Board(seq=100, title=테스트 제목 100, writer=테스터, content=테스트 내용 100,  
--->Board(seq=10, title=테스트 제목 10, writer=테스터, content=테스트 내용 10, cre
```



스프링 데이터 JPA

@Query 어노테이션

(2) 이름기반 파라미터 사용하기

```
public interface BoardRepository extends CrudRepository<Board, Long>{

    //제목에 특정 단어가 포함된 글 목록을 내림차순으로 조회(위치 기반 파라미터)
    /*@Query("SELECT b FROM Board b WHERE b.title LIKE ??1% ORDER BY b.seq DESC")
    List<Board> queryAnnotationTest1(String searchKeyword);*/

    //제목에 특정 단어가 포함된 글 목록을 내림차순으로 조회(이름 기반 파라미터)
    @Query("SELECT b FROM Board b WHERE b.title LIKE %:searchKeyword% "
        + "ORDER BY b.seq DESC")
    List<Board> queryAnnotationTest1(@Param("searchKeyword") String searchKeyword);
```

@Query에 '?1' 대신에 ':searchKeyword'로 수정하여 설정한다.

또한 @Param 어노테이션을 추가하여 2개 이상의 파라미터를 사용할때 오류가 없도록 한다.



스프링 데이터 JPA

@Query 어노테이션

(3) 페이징 및 정렬 처리하기

```
//글 번호를 기준으로 내림차순 정렬하기.  
@Query("SELECT b FROM Board b ORDER BY b.seq DESC")  
List<Board> queryAnnotationTest2(Pageable paging);
```

```
@Test  
public void testQueryAnnotationTest2() {  
    //1페이지를 게시글 5개 내림차순으로 조회  
    Pageable paging = PageRequest.of(0, 5, Sort.Direction.DISC, "seq");  
    List<Board> boardList = boardRepo.queryAnnotationTest2(paging);  
  
    Log.info("검색 결과");  
    for(Board board : boardList) {  
        Log.info("--->" + board.toString());  
    }  
}
```



스프링 데이터 JPA

@Query 어노테이션

(3) 페이징 및 정렬 처리하기

검색 결과

```
--->Board(seq=200, title=테스트 제목 200, writer=테스터, content=테스트 내용 200,  
--->Board(seq=199, title=테스트 제목 199, writer=테스터, content=테스트 내용 199,  
--->Board(seq=198, title=테스트 제목 198, writer=테스터, content=테스트 내용 198,  
--->Board(seq=197, title=테스트 제목 197, writer=테스터, content=테스트 내용 197,  
--->Board(seq=196, title=테스트 제목 196, writer=테스터, content=테스트 내용 196,
```



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 매핑하기

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



연관 관계 매핑

● 연관 관계 매핑

(1) 다대일(N:1) 단방향 매핑하기

데이터 모델링을 하다 보면 다대일 관계가 가장 많이 등장한다. 즉 다대일 관계가 모든 매핑에서 가장 기본이 되는 것이다.

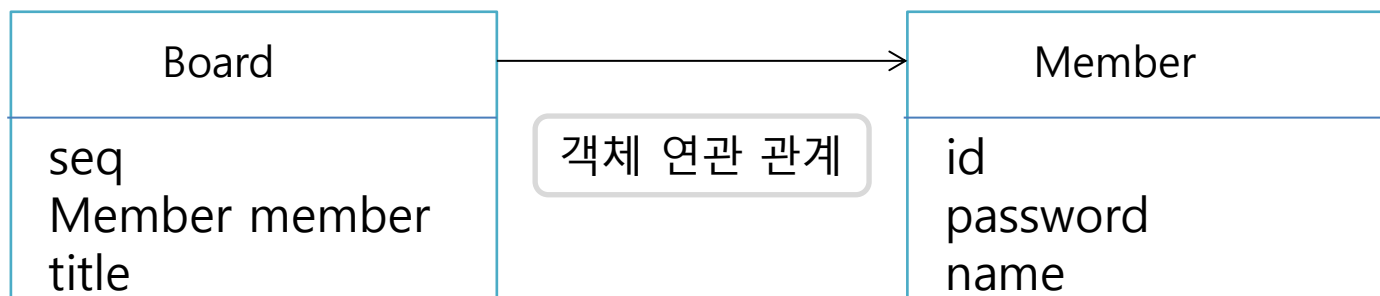
- 게시판과 회원이 있다
- 한명의 회원은 여러 개의 게시글을 작성할 수 있다
- 게시판과 회원은 다대일 관계이다.
- 게시글을 통해서 게시글을 작성한 회원 정보를 조회할 수 있다 .(반대는 안됨)



연관 관계 매핑

- 연관 관계 매핑

- (1) 다대일(N:1) 단방향 매핑하기



게시판 객체(Board)는 참조 변수(Board.member)를 통해 회원 객체(Member)와 관계를 맺는다.

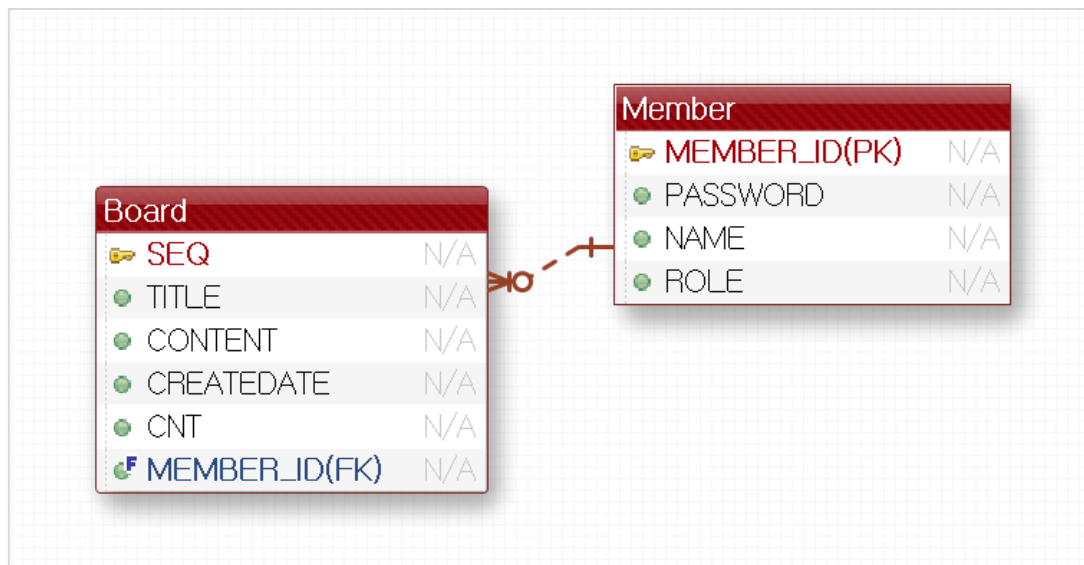
그리고 게시판 객체와 회원 객체는 단방향 관계로서 게시판은 Board.member 변수를 통해 회원 정보를 알 수 있지만 반대로 회원은 게시판에 대한 참조 변수를 가지지 않기 때문에 게시판 정보를 알 수 없다.



연관 관계 매핑

- 연관 관계 매핑

- (1) 다대일(N:1) 단방향 매핑하기



테이블 연관 관계

게시판 테이블(Board)은 MEMBER_ID라는 외래키를 이용하여 회원 테이블(Member)와 연관 관계를 맺는다.

테이블은 객체(엔티티)와 다르게 양방향 관계가 성립한다.

게시판 테이블은 MEMBER_ID 외래 키를 통해서 게시판과 회원 테이블을 조인할 수 있고, 반대로 회원과 게시판도 조인할 수도 있다



연관 관계 매핑

- 연관 관계 매핑

(1) 다대일(N:1) 단방향 매핑하기 – Board 클래스(엔티티)

```
@Entity
public class Board {

    @Id
    @GeneratedValue
    private Long seq;

    private String title;
    //private String writer;
    private String content;

    @Column(updatable=false,
            columnDefinition = "timestamp DEFAULT CURRENT_TIMESTAMP")
    private Date createDate;

    @Column(updatable=false,
            columnDefinition = "bigint DEFAULT 0")
    private Long cnt = 0L;

    @ManyToOne
    @JoinColumn(name="MEMBER_ID")
    private Member member;
}
```

다대일 관계를 설정하기 위해 Member 타입의 member 변수를 추가했고, @ManyToOne 어노테이션을 사용함
@JoinColumn은 name 속성을 통해 외래키 칼럼을 매핑함



연관 관계 매핑

- 연관 관계 매핑

(1) 다대일(N:1) 단방향 매핑하기 – Member 클래스(엔티티)

```
@ToString
@Getter
@Setter
@Entity
public class Member {
    @Id
    @Column(name="MEMBER_ID")
    private String id;

    private String password;

    private String name;

    private String role;
}
```

@Column 어노테이션의 name 속성은 필드와 매핑할 칼럼 이름임.



연관 관계 매핑

(2) 다대일(N:1) 연관 관계 테스트 하기

```
@SpringBootTest
public class ReleationMappingTest {

    @Autowired
    private MemberRepository memberRepo;

    @Autowired
    private BoardRepository boardRepo;

    @Test
    public void testManyToOnInsert() {
        Member member1 = new Member();
        member1.setId("member1");
        member1.setPassword("member111");
        member1.setName("뽀로로");
        member1.setRole("User");
        memberRepo.save(member1);

        Member member2 = new Member();
        member2.setId("member2");
        member2.setPassword("member222");
        member2.setName("아기상어");
        member2.setRole("Admin");
        memberRepo.save(member2);
    }
}
```



연관 관계 매핑

(2) 다대일(N:1) 연관관계 테스트 하기

```
for(int i = 1; i <= 3; i++) {  
    Board board = new Board();  
    board.setMember(member1);  
    board.setTitle("뽀로로가 등록한 게시글" + i);  
    board.setContent("뽀로로가 등록한 게시글 내용" + i);  
    boardRepo.save(board);  
}  
  
for(int i = 1; i <= 3; i++) {  
    Board board = new Board();  
    board.setMember(member2);  
    board.setTitle("아기 상어가 등록한 게시글" + i);  
    board.setContent("아기 상어가 등록한 게시글 내용" + i);  
    boardRepo.save(board);  
}  
}
```



연관 관계 매핑

(2) 다대일(N:1) 연관관계 테스트 하기

| member (2r × 5c) | | | | | |
|------------------|--|------|-----------|-------|---------|
| member_id | | name | password | role | enabled |
| member1 | | 뽀로로 | member111 | User | 0 |
| member2 | | 아기상어 | member222 | Admin | 0 |

| seq | | cnt | content | create_date | title | member_id |
|-----|--|-----|--------------------|---------------------|-----------------|-----------|
| 1 | | 0 | 뽀로로가 등록한 게시물 내용1 | 2022-09-25 16:19:07 | 뽀로로가 등록한 게시물1 | member1 |
| 2 | | 0 | 뽀로로가 등록한 게시물 내용2 | 2022-09-25 16:19:07 | 뽀로로가 등록한 게시물2 | member1 |
| 3 | | 0 | 뽀로로가 등록한 게시물 내용3 | 2022-09-25 16:19:07 | 뽀로로가 등록한 게시물3 | member1 |
| 4 | | 0 | 아기 상어가 등록한 게시물 내용1 | 2022-09-25 16:19:07 | 아기 상어가 등록한 게시물1 | member2 |
| 5 | | 0 | 아기 상어가 등록한 게시물 내용2 | 2022-09-25 16:19:07 | 아기 상어가 등록한 게시물2 | member2 |
| 6 | | 0 | 아기 상어가 등록한 게시물 내용3 | 2022-09-25 16:19:07 | 아기 상어가 등록한 게시물3 | member2 |



연관 관계 매핑

(2) 다대일(N:1) 연관관계 테스트 하기

```
@Test
public void testManyToOneSelect() {
    Board board = boardRepo.findById(5L).get();

    System.out.println("[ " + board.getSeq() + "번 게시글 정보 ]");
    System.out.println("제목\t: " + board.getTitle());
    System.out.println("작성자\t: " + board.getMember().getName());
    System.out.println("내용\t: " + board.getContent());
    System.out.println("권한\t: " + board.getMember().getRole());
}
```

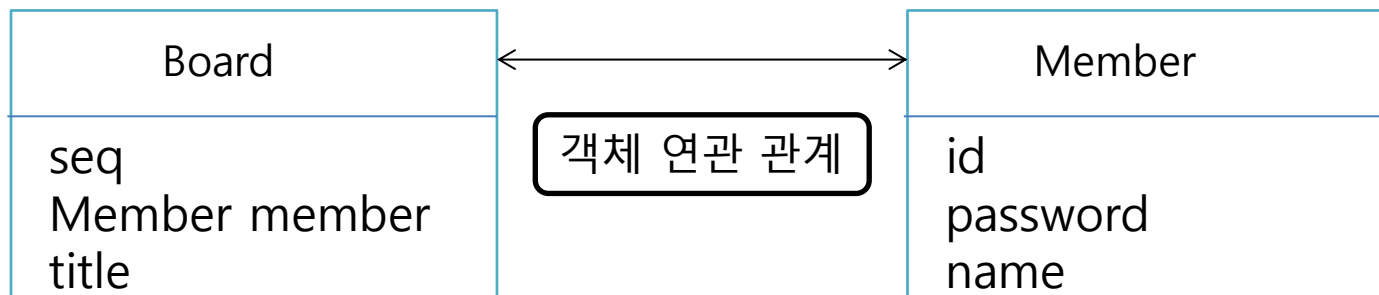
[5번 게시글 정보]

제목 : 아기 상어가 등록한 게시글2
작성자 : 아기상어
내용 : 아기 상어가 등록한 게시글 내용2
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 컬렉션을 사용해야 한다.



연관 관계 매핑

- 양방향 매핑 설정하기

```
@ToString
@Getter
@Setter
@Entity
public class Member {
    @Id
    @Column(name="MEMBER_ID")
    private String id;

    private String password;

    private String name;

    private String role;

    @OneToMany(mappedBy="member", fetch=FetchType.EAGER)
    private List<Board> boardList = new ArrayList<>();
}
```

@OneToMany는 mappedBy 속성과 fetch 속성이 사용되었다.

mappedBy는 연관관계의 주인이 아님을 알려줄 때 사용한다.

즉 연관 관계의 주인은 board이고 member는 주인이 아니라는 뜻임.

fetch 속성은 회원 정보를 조회할 때 게시판 정보도 같이 조회할 것인지를 결정할때는 EAGER 사용하고 LAZY는 연관 엔티티를 실제 사용할때 사용한다.



연관 관계 매핑

- 양방향 매핑 테스트하기

```
@Test
public void testTwoWayMapping() {
    //member1이라는 회원을 조회
    Member member = memberRepo.findById("member1").get();

    System.out.println("=====");
    System.out.println(member.getName() + "가(이) 저장한 게시글 목록");
    System.out.println("=====");
    List<Board> list = member.getBoardList();
    for(Board board : list) {
        System.out.println(board.toString());
    }
}
```

순환 참조 문제로 출력이 안됨

```
=====
뽀로로가(이) 저장한 게시글 목록
=====
2022-09-25 17:49:50.414 IN
```

Failure Trace

java.lang.StackOverflowError

at java.base/java.lang.StringUTF16.checkBoundsOffCount(StringUTF16.java:1636)



연관 관계 매핑

- 양방향 매핑 테스트하기 – 순환 참조 문제 해결

```
@ToString(exclude="member")
@Setter
@Getter
@Entity
public class Board {

    @Id
    @GeneratedValue
    private Long seq;

    private String title;
    //private String writer;
    private String content;
```

```
@ToString(exclude="boardList")
@Getter
@Setter
@Entity
public class Member {
    @Id
    @Column(name="MEMBER_ID")
    private String id;

    private String password;
```



연관 관계 매핑

- 양방향 매핑 테스트하기 - 순환 참조 문제 해결

```
=====
```

```
뽀로로가(이) 저장한 게시물 목록
```

```
=====
```

```
Board(seq=1, title=뽀로로가 등록한 게시물1, content=뽀로로가 등록한 게시물 내용1,
```

```
Board(seq=2, title=뽀로로가 등록한 게시물2, content=뽀로로가 등록한 게시물 내용2,
```

```
Board(seq=3, title=뽀로로가 등록한 게시물3, content=뽀로로가 등록한 게시물 내용3,
```

