Spring Boot Application

스프링 부트 응용

연관관계 매핑

한국기술교육대학교
온라인평생교육원

학습내용

- 단방향 연관관계 매핑
- 양방향 연관관계 매핑

학습목표

- 연관 매핑을 설정하여 데이터베이스 조인을 처리할 수 있다.
- 연관된 엔티티들에 대해 단방향/양방향 매핑을 설정할 수 있다.

단방향 연관관계 매핑

1 연관 매핑

〈1〉 연관 매핑이란?

- 테이블의 연관관계와 엔티티의 연관관계를 어노테이션 같은 메타데이터를 통해 매핑하는 것

1 연관 매핑

〈1〉 연관 매핑이란?

- 관계형 데이터베이스는 정규화를 통해 데이터를 여러 테이블에 나누어 저장하고 조인으로 연관된 데이터를 처리함

단방향 연관관계 매핑

1 연관 매핑

〈1〉 연관 매핑이란?

- 데이터베이스 테이블은 FK를 기반으로 관계를 표현하지만 객체는 참조 변수를 사용함

1 연관 매핑

〈1〉 연관 매핑이란?

- 관계형 데이터베이스에서 FK 기반의 관계를 객체의 참조변수 기반의 관계로 매핑하는 것이 연관 매핑의 핵심임

단방향 연관관계 매핑

1 연관 매핑

〈2〉 연관 매핑 기준

- 연관 매핑의 종류를 구분하는 기준

용어	설명
방향 (Direction)	• 단방향과 양방향이 있음 • 게시판(Board) 객체가 참조 변수를 통해 회원(Member) 객체를 참조하면 단방향, 회원 객체도 게시판 객체를 참조한다면 양방향이 됨 • 중요한 것은 방향은 객체에만 존재하고 테이블은 항상 양방향이라는 것
다중성 (Multiplicity)	• 다대일(N:1), 일대다(1:N), 일대일(1:1), 다대다(N:M)가 있음 • 회원이 여러 개의 게시 글을 소유한다면 회원과 게시판은 일대다(1:N) 관계임 • 반대로 게시판(Board) 입장에서 보면 게시판과 회원은 다대일(N:1) 관계가 됨

2 단방향 연관 매핑

〈1〉 다대일(N:1) 관계

- 데이터 모델링에서 가장 많이 사용하는 관계
- 연관 매핑에서 다대일 단방향 연관관계 매핑만 이해하면 나머지 매핑도 쉽게 이해할 수 있음

단방향 연관관계 매핑

2 단방향 연관 매핑

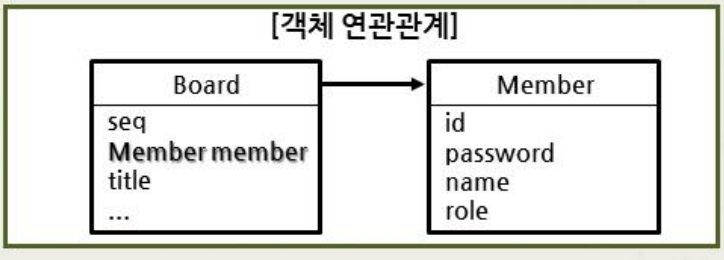
〈1〉 다대일(N:1) 관계

다대일 연관 매핑을 이해하기 위한 조건	
1	게시판(Board)과 회원(Member)이 있음
2	여러 개의 게시 글은 한 명의 회원이 작성할 수 있음
3	게시판과 회원은 다대일 관계임
4	게시글을 통해서 회원 정보를 조회할 수는 있지만 반대는 안됨

2 단방향 연관 매핑

〈2〉 다대일(N:1) 관계 이해

- 주어진 조건을 토대로 작성한 클래스 다이어그램



단방향 연관관계 매핑

2 단방향 연관 매핑

〈2〉 다대일(N:1) 관계 이해

- 주어진 조건을 토대로 작성한 클래스 다이어그램

• 게시판(Board) 객체는 참조 변수(Board.member)를 통해 회원(Member) 객체와 관계를 맺음

2 단방향 연관 매핑

〈2〉 다대일(N:1) 관계 이해

- 주어진 조건을 토대로 작성한 클래스 다이어그램

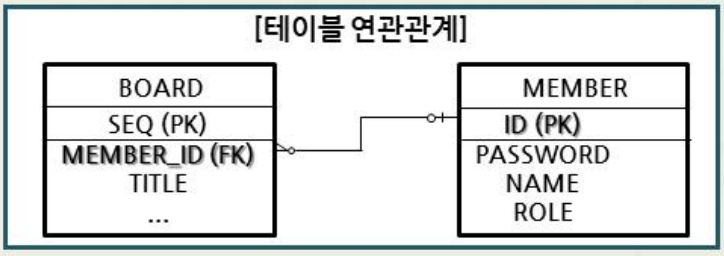
• 게시판과 회원은 단방향으로 게시판은 회원을 참조함
• 반대로 회원은 게시판에 대한 참조 정보를 가지지 않음

단방향 연관관계 매핑

2 단방향 연관 매핑

〈2〉 다대일(N:1) 관계 이해

- 데이터베이스 ERD로 표현



2 단방향 연관 매핑

〈2〉 다대일(N:1) 관계 이해

- 데이터베이스 ERD로 표현

- 게시판(BOARD) 테이블은 외래 키(MEMBER_ID)를 통해 회원(MEMBER)과 관계를 맺음
- 게시판 테이블과 회원 테이블은 하나의 외래 키를 기반으로 양방향 관계가 성립함

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

1 참조되는 엔티티 작성

- 회원(Member) 엔티티는 Board에 의해 단방향으로 참조되는 객체이므로 별다른 설정이 필요 없음

```
package com.mycompany.entity;

import javax.persistence.Entity;

@Data
@Entity
public class Member {
    @Id
    private String id;
    private String password;
    private String name;
    private String role;
}
```

3 다대일 연관 매핑 설정

1 참조되는 엔티티 작성

- Member 엔티티를 이용하여 CRUD기능을 처리하는 MemberRepository 인터페이스도 작성함

```
package com.mycompany.persistence;

import org.springframework.data.repository.CrudRepository;

@Repository
public interface MemberRepository extends CrudRepository<Member, String> {
}
```

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

〈2〉 연관 매핑 설정

- Board 엔티티에 연관 매핑을 위한 참조 변수를 선언하고 @ManyToOne과 @JoinColumn을 설정함

```
package com.mycompany.entity;

import java.util.Date;

@Data
@Entity
public class Board {
    @Id
    @GeneratedValue
    private Integer seq;
    private String title;
//  private String writer;
    private String content;
    @Temporal(value = TemporalType.TIMESTAMP)
    private Date createDate;
    private Integer cnt;

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

3 다대일 연관 매핑 설정

package com.mycompany.entity;

import java.util.Date;

@Data
@Entity
public class Board {
    @Id
    @GeneratedValue
    private Integer seq;
    private String title;
// private String writer;
    private String content;
    @Temporal(value = TemporalType.TIMESTAMP)
    private Date createDate;
    private Integer cnt;

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

변수를
umn을 설정함

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

〈3〉 연관 매핑 어노테이션

1 @ManyToOne

- @ManyToOne은 다대일 관계를 설정할 때 다(N)에 해당하는 참조변수에 설정함

3 다대일 연관 매핑 설정

〈3〉 연관 매핑 어노테이션

1 @ManyToOne

- Board 엔티티는 Member 타입의 member 변수를 통해 Member 엔티티를 단방향 참조함

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

③ 연관 매핑 어노테이션

1 @ManyToOne

- @ManyToOne이 가질 수 있는 속성과 의미

속성	기능	기본값
optional	연관된 엔티티의 필수 여부를 결정함	true
fetch	글로벌 페치(Fetch) 전략을 설정함 • EAGER : 연관 엔티티를 즉시 가져옴 • LAZY : 연관 엔티티가 실제 사용될 때 가져옴	@ManyToOne : EAGER @OneToMany : LAZY
cascade	영속성 전이 범위를 설정함 연관 엔티티를 관리할 때 사용함(등록/수정/삭제)	

3 다대일 연관 매핑 설정

③ 연관 매핑 어노테이션

2 @JoinColumn

- 외래 키(Foreign Key) 매핑을 위해 사용함
- 생략 가능

@JoinColumn 생략 → '참조변수 이름'_'기본 키 컬럼 이름' 자동으로 설정

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

〈3〉 연관 매핑 어노테이션

2 @JoinColumn

- 연관 필드인 Board.member를 외래 키(Foreign Key)로 매핑하기 위해 @JoinColumn을 설정함

3 다대일 연관 매핑 설정

〈3〉 연관 매핑 어노테이션

2 @JoinColumn

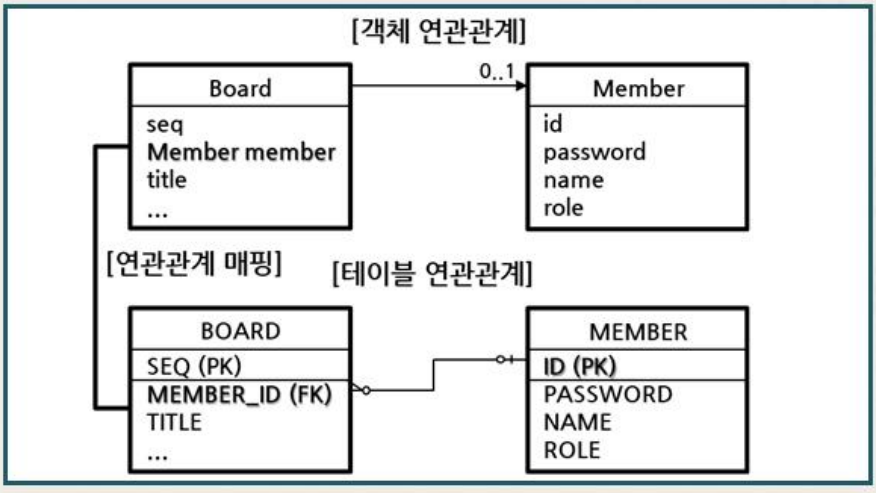
- @JoinColumn은 name 속성을 통해 외래 키 컬럼을 매핑함

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

⟨3⟩ 연관 매핑 어노테이션

2 @JoinColumn



3 다대일 연관 매핑 설정

⟨4⟩ 연관관계 INSERT

- 엔티티를 저장할 때는 연관관계에 있는 엔티티가 반드시 영속 상태에 있어야 함
- 따라서 Member 객체가 먼저 영속 상태로 들어가고 Board 객체가 영속 상태로 전환되어야 함

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

④ 연관관계 INSERT

```
@SpringBootTest
public class RelationMappingTest {
    @Autowired
    private BoardRepository boardRepo;
    @Autowired
    private MemberRepository memberRepo;

    @Test
    public void manyToOneInsert() {
        Member member = new Member();
        member.setId("member01");
        member.setPassword("member111");
        member.setName("둘리");
        member.setRole("User");
        memberRepo.save(member);
        for (int i = 1; i <= 3; i++) {
            Board board = new Board();
            board.setMember(member);
            board.setTitle("둘리가 등록한 게시글 " + i);
            board.setContent("둘리가 등록한 게시글 내용 " + i);
            board.setCreateDate(new Date());
            board.setCnt(0);
            boardRepo.save(board);
        }
    }
}
```

3 다대일 연관 매핑 설정

④ 연관관계 INSERT

- INSERT 실행 결과

실행 | Run Selected | 자동 완성 | 지우기 | SQL 문:

```
SELECT * FROM MEMBER ;
SELECT * FROM BOARD ;
```

SELECT * FROM MEMBER ;

ID	NAME	PASSWORD	ROLE
member01	둘리	member111	User

(1 row, 2 ms)

SELECT * FROM BOARD ;

SEQ	CNT	CONTENT	CREATE_DATE	TITLE	MEMBER_ID
1	0	둘리가 등록한 게시글 내용 1	2021-08-04 19:35:24.333	둘리가 등록한 게시글 1	member01
2	0	둘리가 등록한 게시글 내용 2	2021-08-04 19:35:24.352	둘리가 등록한 게시글 2	member01
3	0	둘리가 등록한 게시글 내용 3	2021-08-04 19:35:24.356	둘리가 등록한 게시글 3	member01

(3 행, 0 ms)

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

⑤ 연관관계 SELECT

- 연관관계에 있는 엔티티 조회

> • Board 엔티티에 대한 상세 조회는 findById 메소드로 처리함

```
@SpringBootTest
public class RelationMappingTest {
    @Autowired
    private BoardRepository boardRepo;

    @Test
    public void manyToOneSelect() {
        Board board = boardRepo.findById(3).get();

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

3 다대일 연관 매핑 설정

⑤ 연관관계 SELECT

> • @ManyToOne 어노테이션의 페치(fetch) 속성의 기본은 EAGER이므로 SELECT 결과에 외부 조인이 처리되는 것을 확인할 수 있음

```
Hibernate:
    select
        board0_.seq as seq1_0_0_,
        board0_.cnt as cnt2_0_0_,
        board0_.content as content3_0_0_,
        board0_.create_date as create_d4_0_0_,
        board0_.member_id as member_i6_0_0_,
        board0_.title as title5_0_0_,
        member1_.id as id1_1_1_,
        member1_.name as name2_1_1_,
        member1_.password as password3_1_1_,
        member1_.role as role4_1_1_
    from
        board board0_
    left outer join
        member member1_
            on board0_.member_id=member1_.id
    where
        board0_.seq=?
[ 3번 게시글 정보 ]
제목 : 둘리가 등록한 게시글 3
내용 : 둘리가 등록한 게시글 내용 3
작성자 : 둘리
```

단방향 연관관계 매핑

3 다대일 연관 매핑 설정

```
Hibernate:
    select
        board0_.seq as seq1_0_0_,
        board0_.cnt as cnt2_0_0_,
        board0_.content as content3_0_0_,
        board0_.create_date as create_d4_0_0_,
        board0_.member_id as member_i6_0_0_,
        board0_.title as title5_0_0_,
        member1_.id as id1_1_1_,
        member1_.name as name2_1_1_,
        member1_.password as password3_1_1_,
        member1_.role as role4_1_1_
    from
        board board0_
    left outer join
        member member1_
            on board0_.member_id=member1_.id
    where
        board0_.seq=?
[ 3번 게시글 정보 ]
제목 : 둘리가 등록한 게시글 3
내용 : 둘리가 등록한 게시글 내용 3
작성자 : 둘리
```

etch) 속성의
에 외부 조인이

외부 조인(OUTER JOIN)으로 처리

3 다대일 연관 매핑 설정

⑥ 외부 조인을 내부 조인(INNER JOIN)으로 변경

- 외부 조인보다는 내부 조인이 성능이 좋음
- 따라서 @JoinColumn의 nullable 속성을 이용하면 외부 조인으로 처리되는 쿼리를 내부 조인으로 변경할 수 있음

```
@ManyToOne
@JoinColumn(name="MEMBER_ID", nullable = false)
private Member member;
```

양방향 연관관계 매핑

1 양방향 연관관계 매핑

〈1〉 양방향 매핑

- 연관관계에 참여하는 엔티티가 서로에 대한 참조를 양쪽에서 가지는 것을 양방향이라고 함
- 테이블은 처음부터 양방향이었으므로 엔티티에 대한 관계만 양방향으로 설정하면 됨

1 양방향 연관관계 매핑

〈1〉 양방향 매핑

- 양방향 관계에서 매핑에 참여하는 참조 변수는 두 개인데 외래 키는 하나이기 때문에 둘 사이에 차이가 발생함
- 두 참조 변수 중 하나를 연관관계의 주인(Owner)으로 지정하여 외래 키를 소유하고 관리함

양방향 연관관계 매핑

1 양방향 연관관계 매핑

2 양방향 매핑 설정

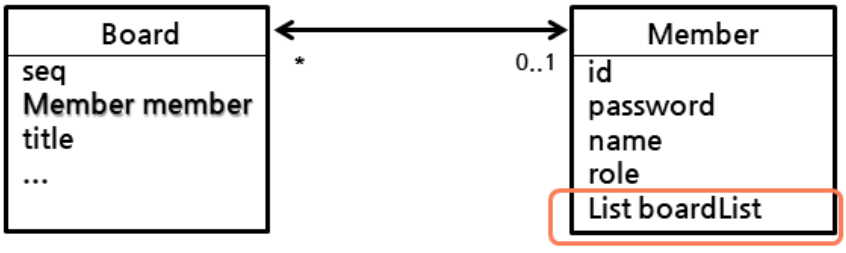
- 게시판 관점에서 회원은 다대일 관계지만 회원 입장에서 게시판은 일대다 관계가 됨
- 일대다 관계는 객체 하나가 여러 객체와 연관관계를 맺을 수 있으므로 List 같은 컬렉션을 사용해야 함

1 양방향 연관관계 매핑

2 양방향 매핑 설정

- 일대다 관계를 매핑할 때는 @OneToMany를 사용함

[양방향 객체 연관관계]



양방향 연관관계 매핑

1 양방향 연관관계 매핑

〈3〉 양방향 매핑 적용

- @OneToMany는 mappedBy 속성과 fetch 속성을 사용할 수 있음
- fetch 속성은 연관된 엔티티의 조회 시점을 결정함
- @OneToMany의 기본 설정은 LAZY

1 양방향 연관관계 매핑

〈3〉 양방향 매핑 적용

- mappedBy 속성은 연관관계의 주인이 아닌 변수에 설정하여 반대 쪽이 연관관계의 주인임을 알려줌

양방향 연관관계 매핑

1 양방향 연관관계 매핑

〈3〉 양방향 매핑 적용

- 일반적으로 연관관계의 주인은 테이블에 외래 키가 있는 엔티티로 설정함

```
@Data
@Entity
public class 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<Board>();
}
```

Member는 주인이 아니며 Board.member 변수로 매핑되어 있음

회원 엔티티를 가져올 때 연관된 게시글 엔티티 목록도 같이 가져옴

2 순환 참조와 영속성 전이

〈1〉 순환 참조

- 양방향 매핑에서는 연관관계에 있는 엔티티들이 서로를 참조하는 순환 참조 문제가 발생할 수 있음

양방향 연관관계 매핑

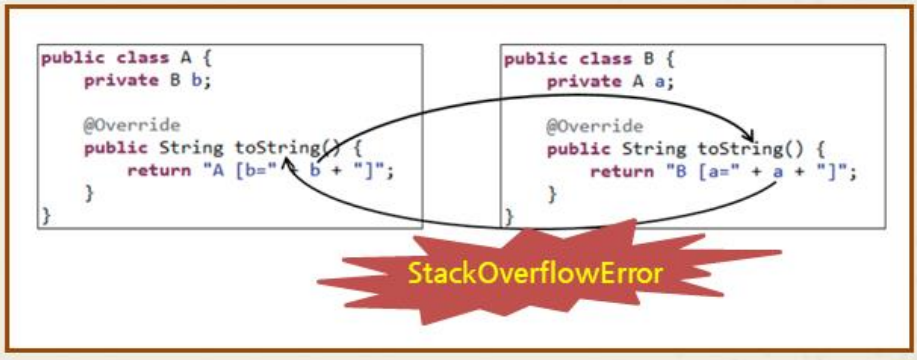
2 순환 참조와 영속성 전이

⟨1⟩ 순환 참조

1. 롬복(Lombok)이 제공하는 toString 메소드는 양방향 관계에서 서로의 toString 메소드를 참조
2. StackOverflowError가 발생함

2 순환 참조와 영속성 전이

⟨1⟩ 순환 참조



양방향 연관관계 매핑

2 순환 참조와 영속성 전이

② 순환 참조 처리

- 순환 참조 문제를 해결하기 위해서 @ToString의 exclude 속성을 사용함

2 순환 참조와 영속성 전이

② 순환 참조 처리

- exclude 속성은 연관관계에 참여하는 두 엔티티 중 한 곳에만 설정해도 됨

```
@Data
@ToString(exclude = "boardList")
@Entity
public class 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<Board>();
}
```

양방향 연관관계 매핑

2 순환 참조와 영속성 전이

〈3〉 영속성 전이

- 특정 엔티티를 상태를 변경할 때 연관된 엔티티의 상태도 같이 변경하는 것

2 순환 참조와 영속성 전이

〈3〉 영속성 전이

- 특정 엔티티를 상태를 변경할 때 연관된 엔티티의 상태도 같이 변경하는 것

영속성 전이

예 회원 정보를 삭제할 때 회원이 등록한 게시글 목록도 같이 삭제하는 것

양방향 연관관계 매핑

2 순환 참조와 영속성 전이

〈3〉 영속성 전이

- cascade 속성을 이용하면 부모 엔티티의 상태 변화가 자식 엔티티 쪽으로 전이되도록 할 수 있음

2 순환 참조와 영속성 전이

〈3〉 영속성 전이

CascadeType의 종류

- ALL : CascadeType - CascadeType
- DETACH : CascadeType - CascadeType
- MERGE : CascadeType - CascadeType
- PERSIST : CascadeType - CascadeType
- REFRESH : CascadeType - CascadeType
- REMOVE : CascadeType - CascadeType

- 모든 전이를 적용하려면 ALL을 설정함

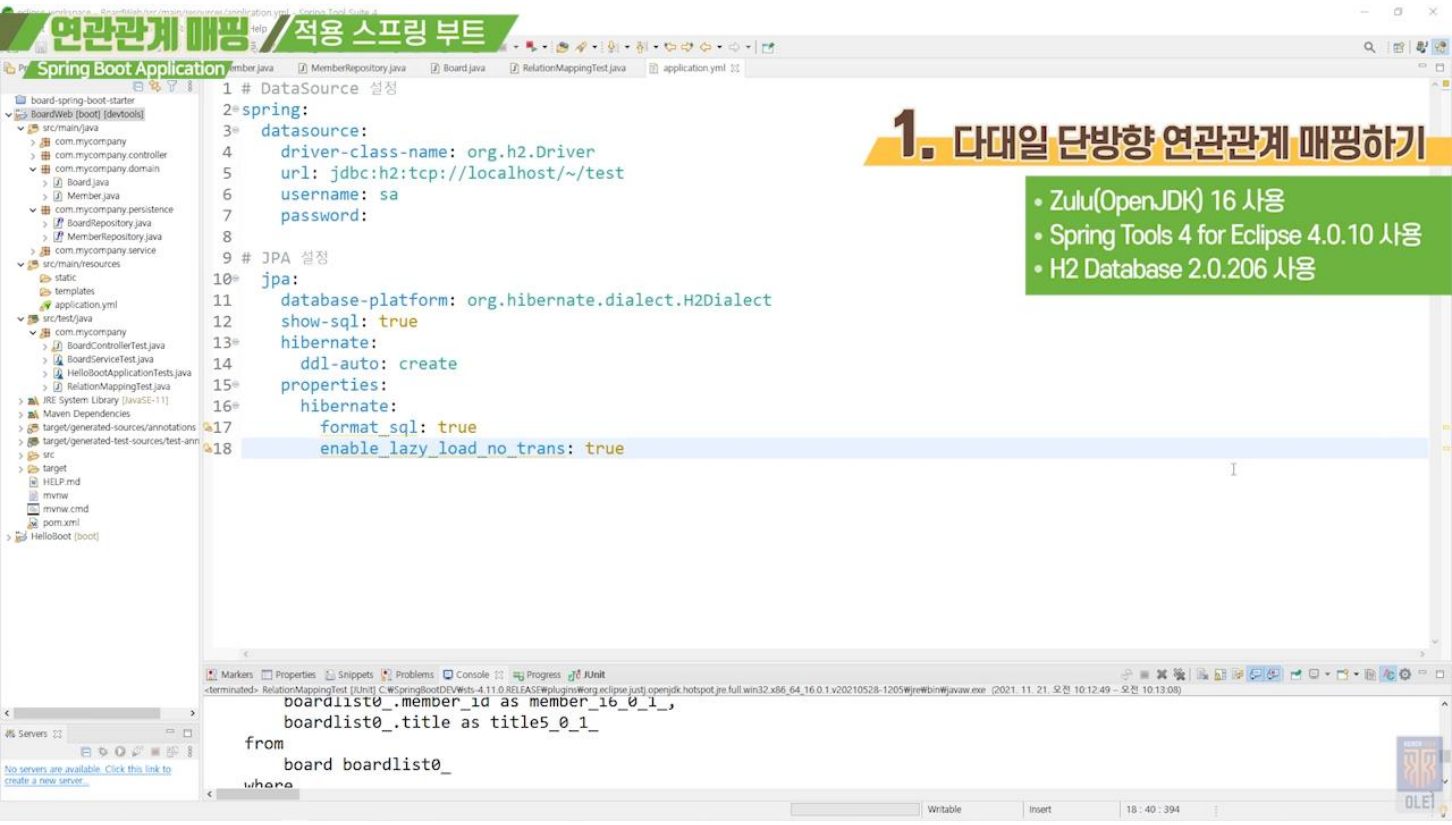
양방향 연관관계 매핑

2 순환 참조와 영속성 전이

〈3〉 영속성 전이

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

적용 스프링 부트

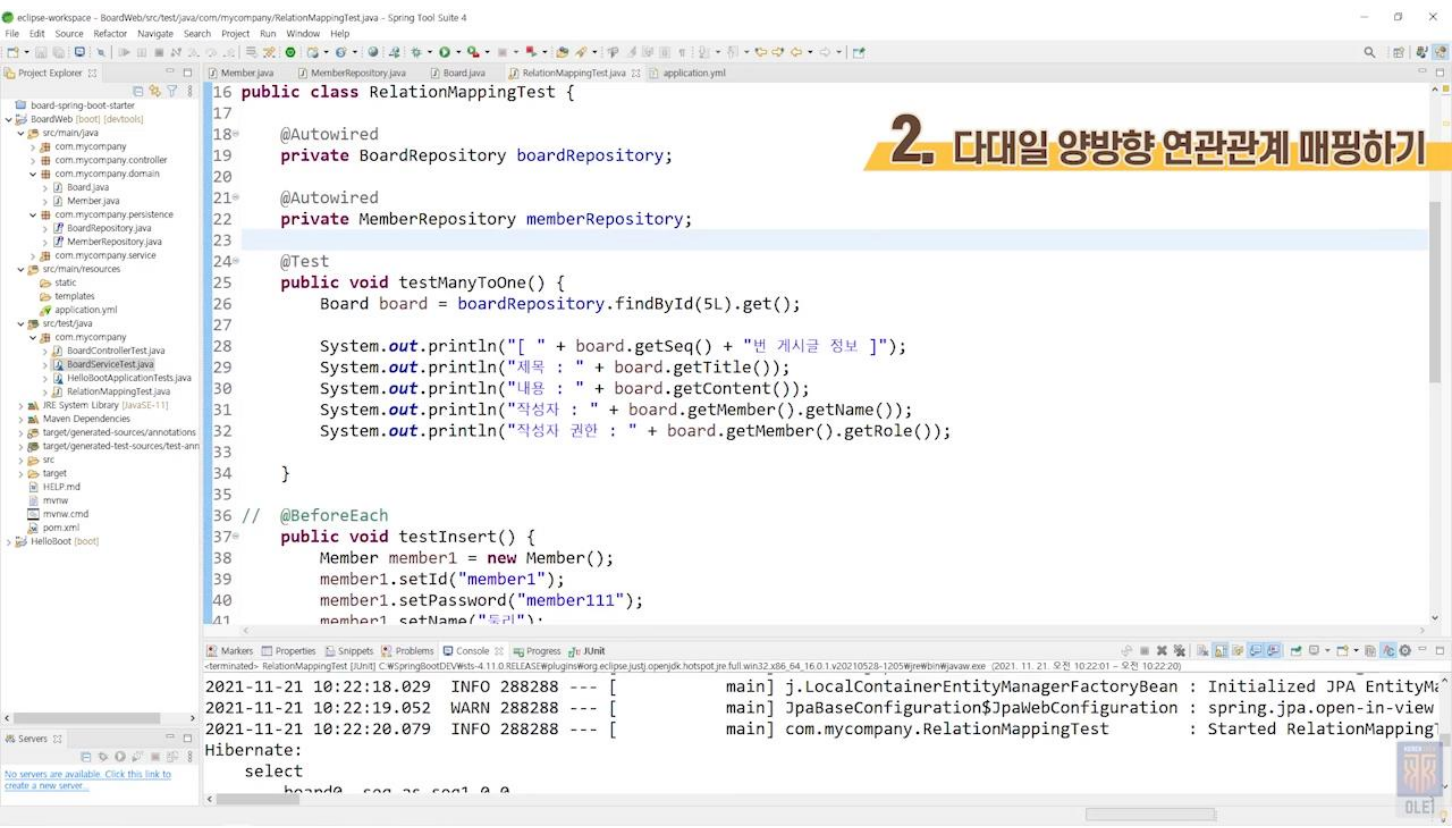


실습단계
다대일 단방향 매핑과 다대일, 일대다 양방향 매핑
트랜잭션이 없는 환경에서도 lazy loading을 사용할 수 있도록 설정
네 개의 컬럼으로 구성
id가 식별자 필드
CRUD 기능을 처리
연관 매핑 추가
게시판 입장에서 게시판과 회원 관계는 다대일
member 참조변수 추가
기존 writer 변수는 주석으로 막음
주석으로 막음으로 인해 test 케이스 중 하나가 에러날 것
에러나는 것이 싫을 경우 TestCase를 열어 writer 변수 값 세팅을 주석으로 막을 것
다대일 관계
중요! 보드 테이블에 MEMBER_ID 컬럼을 추가하고, 외래 키로 지정

적용 스프링 부트

실습단계
다대일 관계 테스트 케이스 작성
테스트 데이터를 등록하는 메소드로 구현
Member 엔티티 두 개 저장
Board 엔티티를 각각 생성해서 저장
Board 엔티티와 연관관계에 있는 Member 엔티티를 세팅해서 save( ) 한 것
MEMBER 테이블, BOARD 테이블이 새로 생성됨
중요! 테이블이 만들어졌을 때 H2 데이터베이스를 통해 확인
MEMBER_ID : 외래키가 됨
주석으로 막음
있는 테이블 유지 → update로 변경
중요! 이 게시글을 등록한 회원을 board.getMember 메소드를 통해 멤버 엔티티에 접근할 수 있음
Member 엔티티의 name과 role 값도 사용할 수 있음
두 가지 확인, 1. 정보 출력
2. 내부적으로 join 쿼리가 동작하는 것

적용 스프링 부트



실습단계
양방향 매핑
List Collection 타입의 boardList 변수 추가
연관관계에서 주인이 아님을 알려주기 위한 mappedby라는 속성에 member 변수 지정
중요! toString( ) 메소드가 순환 참조하지 않도록 boardList 변수는 제외
Member 정보 1개 SELECT하여 회원 이름 출력
글목록도 출력
디폴트가 lazy이기 때문에 Member 정보만 사용한다면 BOARD 테이블을 SELECT 하지 않음
회원 이름 출력
회원이 등록한 게시글 목록 열람 가능