

Association

정규화

❖ 정규화

- ✓ 데이터 모델링에서는 데이터 중복으로 인한 메모리 낭비를 피하고 효율적인 데이터 관리를 위해 일반적으로 정규화(Normalization)를 수행하는데 정규화가 잘 된 데이터베이스에는 데이터 중복이 없고 규모가 비슷한 여러 개의 테이블이 생성됨
- ✓ 정규화된 테이블은 이후에 외래 키(Foreign Key)를 기반으로 다른 테이블의 데이터를 조회
- ✓ 객체지향에서도 시스템에서 사용하는 기능을 하나의 객체, 하나의 클래스에서 모두 처리하지 않고 여러 클래스에 나누어 설계하는데 여러 클래스에 흩어져 있는 기능들을 이용하여 사용자가 원하는 비즈니스 로직을 동작시키려면 객체들을 서로 연결할 수 있어야 함
- ✓ 클래스에는 데이터베이스의 외래 키 같은 개념이 없으므로 다른 클래스의 변수나 메소드를 사용하기 위해서는 객체 참조 변수를 사용 해야함
- ✓ 객체 지향과 관계형 데이터베이스는 비슷하면서도 다른 점들이 있지만 JPA는 객체와 테이블을 매핑하여 객체를 중심으로 데이터를 관리하는 기술이기 때문에 객체의 관계와 테이블의 관계를 매핑할 수 있어야 하는데 이를 연관 관계 매핑이라고 함

연관 관계 와 관계형 데이터베이스

❖ 관계

- ✓ 관계형 데이터베이스에서는 일대일(1:1), 일대다(1:N), 다대일(N:1), 다대다 (M:N)의 관계를 이용해서 데이터가 서로 간에 어떻게 구성 되었는지를 표현하는데 이 표현에서 가장 중요한 것이 PK(주키, Primary Key)와 FK(Foreign Key, Foreign Key)를 어떻게 설정할 것인가 하는 것
- ✓ 관계의 종류
 - 일대일(1:1): @OneToOne
 - 일대다(1:N): @OneToMany
 - 다대일(N:1): @ManyToOne
 - 다대다(N:M): @ManyToMany
- ✓ 관계의 방향성 – 관계형 데이터베이스 에서는 없음
 - 단방향
 - 양방향

연관 관계 와 관계형 데이터베이스

❖ 1: N 관계

✓ 회원 과 게시글 의 관계

회원 데이터

아이디	이름	패스워드
user1	사용자1	111
user2	사용자2	111
user3	사용자3	111
user4	사용자4	111
user5	사용자5	111

게시글 데이터

번호	제목	작성자	내용
1	오늘 날씨가..	user1	
2	집에 오늘 길은..	user2	
3	차가 막혀서..	user3	
4	어이 없는 일이..	user4	
5	감기 조심하세요..	user1	
6	이번 개봉 영화..	user2	
7	오늘 점심에..	user1	

연관 관계 와 관계형 데이터베이스

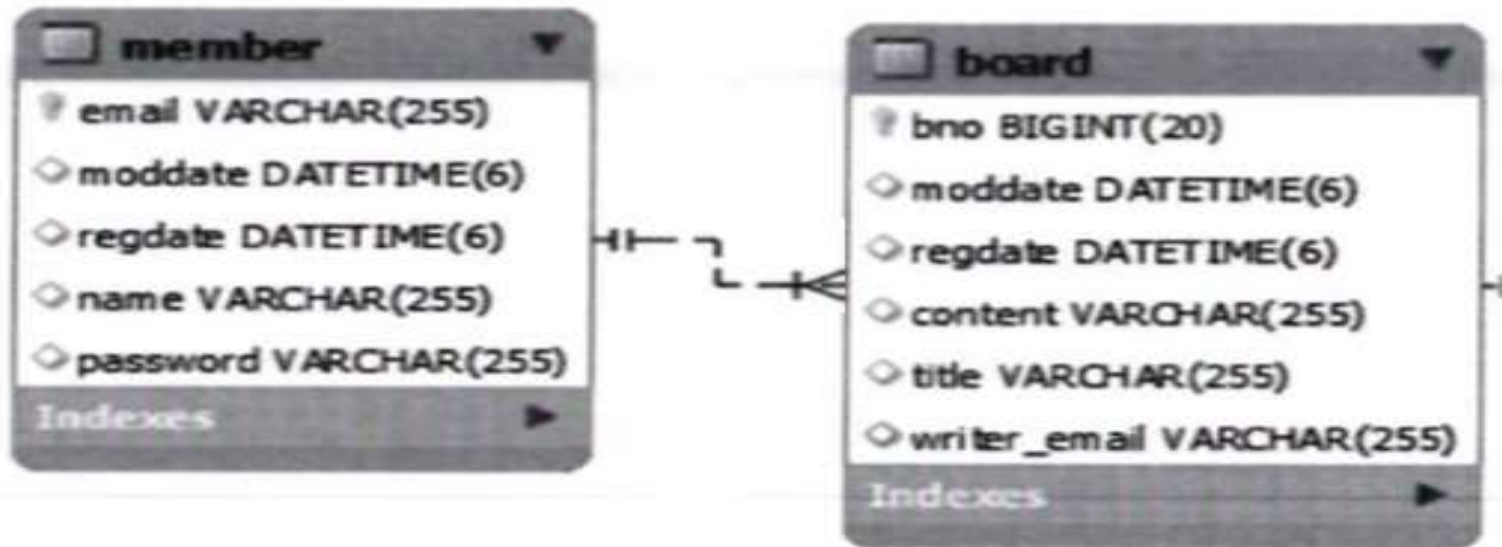
❖ 1: N 관계

✓ 회원 과 게시글 의 관계

- 회원 데이터의 아이디는 PK에 해당하는데 아이디는 회원을 구분할 수 있는 고유한 값을 가지게 되는데 게시글 데이터를 보면 작성자 칼럼 값으로 동일한 회원 아이디가 여러 번 나오는 것을 알 수 있음
- 회원 데이터의 입장에서는 하나(One)의 PK(아이디)가 여러 (Many) 게시글에서 참조(FK)되고 있는 관계
- 하나의 게시글은 한 명의 회원에 의해서 작성된다 라는 것은 게시글 데이터에 작성자 칼럼이 하나만 필요하다는 뜻일 뿐이지 위의 구성과 같이 작성자가 여러 개라는 사실을 찾아낼 수는 없음
- 테이블 간의 관계는 특정한 PK가 다른 곳에서 몇 번 FK로 사용되었는지가 중요한데 우선 어떠한 PK를 기준으로 할 것인지를 고민하고 해당 PK가 다른 곳에서 몇 번 사용되었는지를 세어보는 방식으로 찾아냄
- 위의 구조에서는 회원 데이터 쪽이 일(one) 이고 게시글 데이터는 동일한 회원 아이디가 여러 번 나오고 있으므로 다(many)로 판단하고 ERD를 작성

연관 관계 와 관계형 데이터베이스

- ❖ 1: N 관계
 - ✓ 회원 과 게시글의 관계



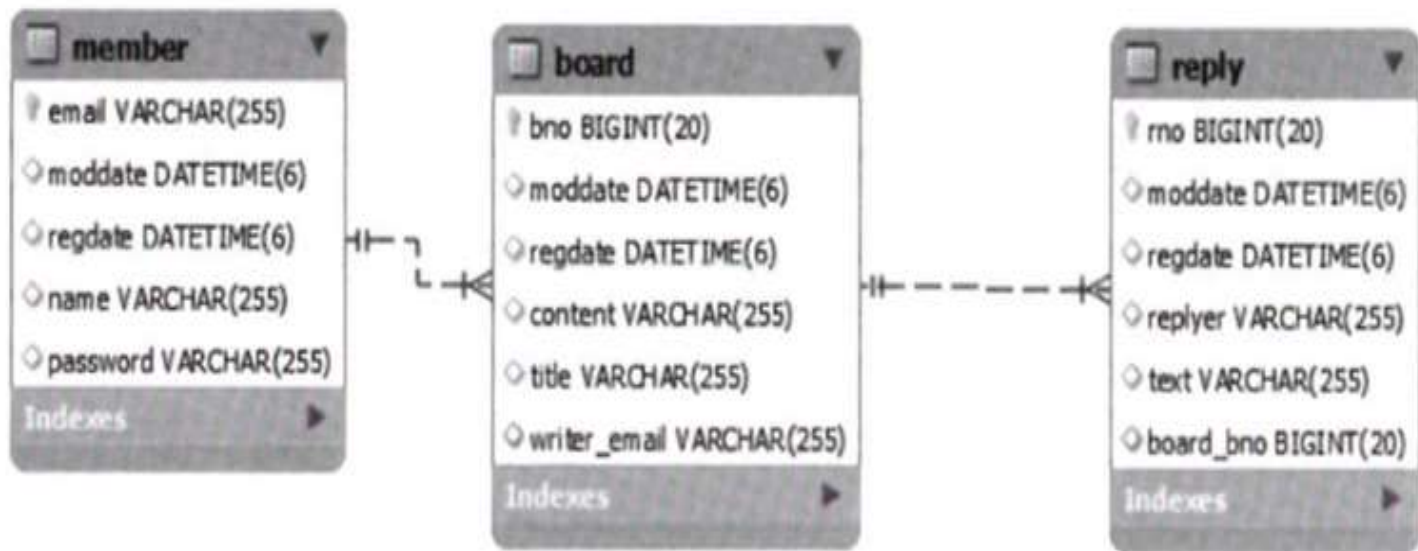
연관 관계 와 관계형 데이터베이스

❖ 1: N 관계

- ✓ 회원 과 게시글 의 관계
 - 한 명의 회원은 여러 개의 게시글을 작성 가능
 - 하나의 게시글은 한 명의 작성자 만을 표시
- ✓ 게시글 과 댓글 의 관계
 - 하나의 게시글은 여러 개의 댓글을 가질 수 있음
 - 하나의 댓글은 하나의 게시글에 속함
- ✓ 게시물과 댓글의 관계는 1:N(일대다)이고 댓글과 게시글의 입장에서는 N:1(다대일)의 관계

연관 관계 와 관계형 데이터베이스

- ❖ 1: N 관계
 - ✓ 회원/게시글/댓글



연관 관계 와 관계형 데이터베이스

❖ 1: N 관계

✓ 회원/게시글/댓글

- 회원이 있어야만 게시글을 작성할 수 있으므로 회원 테이블을 먼저 설계하고 게시글을 작성할 때는 특정 회원과의 관계를 설정해 주어야 하고 댓글은 게시글이 있어야만 작성할 수 있으므로 게시글을 우선 설계하고 댓글 테이블이 게시글을 FK로 작성
- FK를 기준으로 위의 관계를 해석하면 다음과 같음
 - ◆ 게시물(board)은 회원(member)과 다대일(N:1)의 관계
 - ◆ 댓글(reply)은 게시물(board)과 다대일(N:1)의 관계
- JPA는 객체지향의 입장에서 관계를 보기 때문에 데이터베이스와 달리 다음과 같은 선택이 가능
 - ◆ 회원 Entity가 게시물 Entity들을 참조하게 설정해야 하는가?
 - ◆ 게시물 Entity에서 회원 Entity를 참조하게 설정해야 하는가?
 - ◆ 회원 게시물 Entity 객체 양쪽에서 서로를 참조하게 설정해야 하는가?
- 관계형 데이터베이스에서는 PK와 FK만으로 표현되었던 관계가 객체 지향에서는 위와 같이 다양한 선택지가 존재하게 되고 단방향(unidirectional) 참조, 양방향(bidirectional) 참조 라고 표현하기도 하는데 실제 데이터베이스에서는 양방향이라는 말은 존재하지 않기 때문에 객체지향에서만 겪는 문제

연관 관계 와 관계형 데이터베이스

❖ 1: N 관계

✓ 회원/게시글/댓글

- 가장 간단한 시작은 객체지향보다는 관계형 데이터베이스 모델링을 위주로 해서 구성하는 것이 편리
- FK(Foreign Key)를 사용하는 Entity가 PK(주키)를 가진 Entity를 참조하는 구조로 설계하면 데이터베이스와 동일한 구조가 되기 때문에 관계를 이해하는 것도 편하고 자동으로 테이블이 생성될 때도 유용

연관 관계 와 관계형 데이터베이스

❖ 프로젝트 생성 - board

✓ 생성 옵션

- Type: Gradle
- Packaging: Jar
- 의존성: Spring Boot DevTools, Lombok, Spring Web, Thymeleaf, Spring Data JPA, MariaDB(MySQL, Oracle)

연관 관계 와 관계형 데이터베이스

❖ 기본 설정

- ✓ application.properties 파일을 application.yml 파일로 변경하고 설정을 추가

```
server:  
  port: 80
```

```
spring:  
  datasource:  
    url: jdbc:mariadb://localhost:3306/adam  
    driver-class-name: org.mariadb.jdbc.Driver  
    username: root  
    password: wnddkd
```

```
jpa:  
  hibernate:  
    ddl-auto: update  
  properties:  
    hibernate:  
      format_sql: true  
      show_sql: true  
      use_sql_comments: true
```

연관 관계 와 관계형 데이터베이스

❖ 기본 설정

- ✓ application.properties 파일을 application.yml 파일로 변경하고 설정을 추가

```
#Live Reload
devtools:
  livereload:
    enabled: true
```

```
thymeleaf:
  cache: false
```

```
logging:
  level:
    org.hibernate.type.descriptor.sql: trace
```

연관 관계 와 관계형 데이터베이스

❖ 기본 설정

- ✓ 기본적으로 실행되는 클래스에 자동 시간 처리를 위한 Annotation 추가

```
@SpringBootApplication
```

```
@EnableJpaAuditing
```

```
public class BoardApplication {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(BoardApplication.class, args);
```

```
    }
```

```
}
```

연관 관계 와 관계형 데이터베이스

❖ 모델 생성

- ✓ 기본 패키지에 domain 패키지를 생성하고 모든 Entity의 기본이 되는 BaseEntity 클래스를 생성하고 작성

```
@MappedSuperclass
@EntityListeners(value = { AuditingEntityListener.class })
@Getter
abstract class BaseEntity {
    @CreatedDate
    @Column(name = "regdate", updatable = false)
    private LocalDateTime regDate;

    @LastModifiedDate
    @Column(name = "moddate")
    private LocalDateTime modDate;
}
```

연관 관계 와 관계형 데이터베이스

❖ 모델 생성

- ✓ domain 패키지에 회원 정보 관련 Member Entity 클래스를 생성하고 작성

```
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString
@Table(name = "tbl_member")
public class Member extends BaseEntity {
    @Id
    private String email;
    private String password;
    private String name;
}
```


연관 관계 와 관계형 데이터베이스

❖ 모델 생성

- ✓ domain 패키지에 게시물 정보 관련 Board Entity 클래스를 생성하고 작성

```
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString
public class Board extends BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long bno;
    private String title;
    private String content;
}
```

연관 관계 와 관계형 데이터베이스

❖ 모델 생성

- ✓ entity 패키지에 댓글 정보 관련 Reply Entity 클래스를 생성하고 작성

```
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString
public class Reply extends BaseEntity{
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long rno;
    private String text;
    private String replyer;
}
```

연관 관계 와 관계형 데이터베이스

❖ 관계 Annotation의 세부 속성

- ✓ optional: 연관된 Entity의 필수 존재 여부로 반드시 있어야 하는 경우에는 false로 설정하는데 기본값은 true인데 이를 false로 설정하면 inner join 이 되고 true로 설정되면 outer join
- ✓ mappedBy: 양방향 연관 관계 일 때 반대쪽에 매핑되는 Entity의 필드를 연결하는 것으로 연관 관계에서 매핑이 되는 쪽에는 설정하지 않음
- ✓ cascade
 - Entity Cascade는 Entity의 상태 변화를 전파시키는 옵션
 - 단방향 혹은 양방향으로 매핑되어 있는 Entity에 대해 어느 한쪽 Entity의 상태(생성 혹은 삭제)가 변경되었을 시 그에 따른 변화를 바인딩 된 Entity 들에게 전파하는 것
 - 종류
 - ◆ PERSIST - 부모 Entity가 영속화 될 때 자식 Entity도 영속화
 - ◆ MERGE - 부모 Entity가 병합될 때 자식 Entity도 병합
 - ◆ REMOVE - 부모 Entity가 삭제될 때 연관된 자식 Entity도 삭제
 - ◆ REFRESH - 부모 Entity가 refresh되면 연관된 자식 Entity도 refresh
 - ◆ DETACH - 부모 Entity가 detach 되면 연관된 자식 Entity도 detach 상태로 변경, 트랜잭션의 영향을 받지 않음
 - ◆ ALL. - 부모 Entity의 영속성 상태 변화를 자식 Entity에 모두 전이

연관 관계 와 관계형 데이터베이스

❖ 관계 Annotation의 세부 속성

- ✓ orphanRemoval
 - PK(JoinColumn)값이 NULL로 변한 자식은 고아 객체라고 하여 연결된 점이 없는 객체
 - orphanRemoval 옵션을 true로 설정하면 NULL 인 데이터 삭제
- ✓ fetch: 관계의 Entity를 어떻게 가져올 것인지를 정하는 정책으로 Eager(초기) 혹은 Lazy(나중) 옵션을 지정

연관 관계 와 관계형 데이터베이스

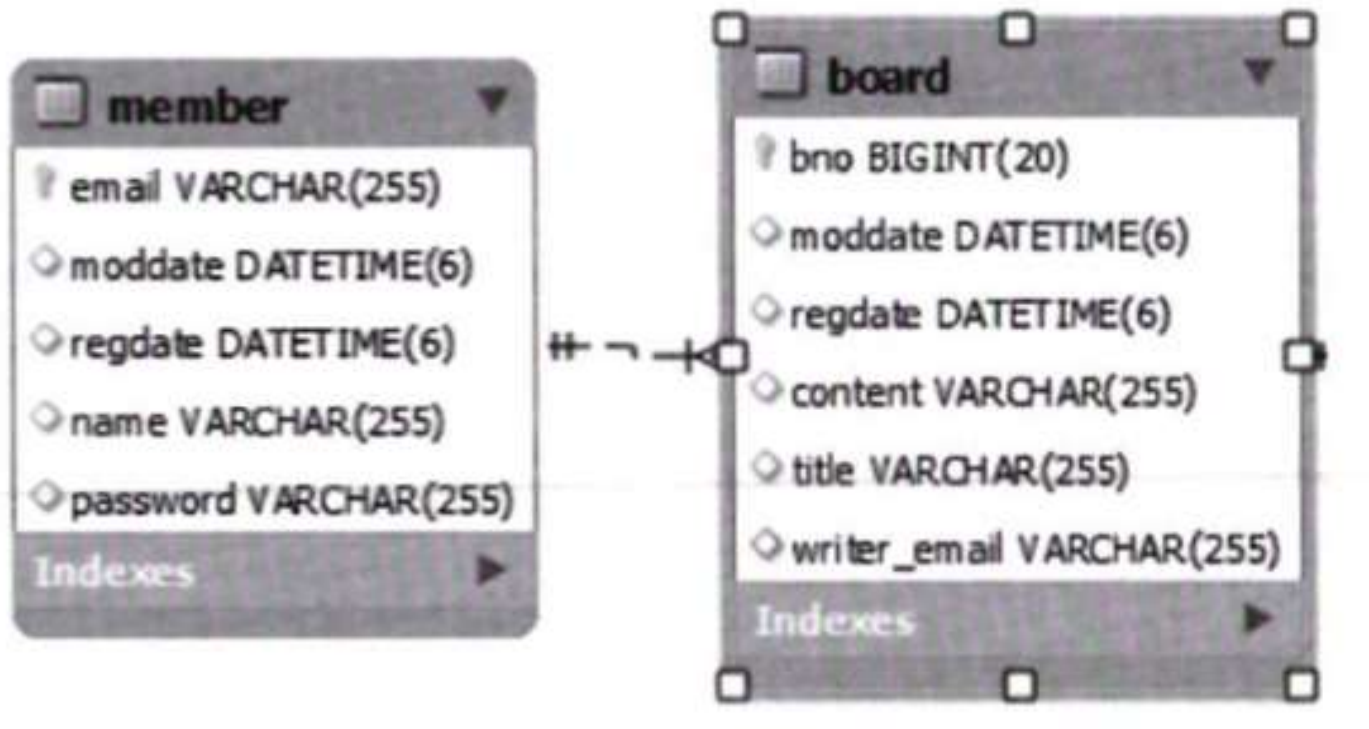
❖ @ManyToOne Annotation

- ✓ Foreign Key를 소유해야 하는 테이블을 위한 Entity에 설정하는데 @ManyToOne 이라는 Annotation과 함께 속성을 설정하면 됨
- ✓ 반대 방향으로 설정하고자 하는 경우는 @OneToMany
- ✓ Entity 설정 시 @JoinColumn 을 이용해서 join 하는 컬럼 이름을 명시하는 것도 가능한데 설정하지 않으면 참조변수이름 + '_' + 참조하는 테이블의 Primary Key 컬럼 이름 으로 생성
- ✓ 양방향 관계를 만들 때 Foreign Key 관리
 - 연관 관계의 주가 되는 곳은 Foreign Key가 있는 곳으로 설정
 - 연관 관계의 주가 되는 곳에서 Foreign Key 를 관리(등록, 수정, 삭제)
 - 주가 아닌 쪽은 연관 관계 매핑 시 mappedBy 속성의 값으로 연관 관계의 주가 되는 곳을 설정
 - 주가 아닌 쪽은 읽기만 가능

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne Annotation

- ✓ Board 테이블 과 Member 테이블의 관계



연관 관계 와 관계형 데이터베이스

❖ @ManyToOne Annotation

- ✓ Board 테이블 과 Member 테이블의 관계 – Board Entity 수정

```
import lombok.*;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Builder
```

```
@AllArgsConstructor
```

```
@NoArgsConstructor
```

```
@Getter
```

```
@ToString(exclude = "writer")
```

```
public class Board extends BaseEntity {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long bno;
```

```
    private String title;
```

```
    private String content;
```

```
    @ManyToOne
```

```
    private Member writer;
```

```
}
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne Annotation

- ✓ Reply 테이블 과 Board 테이블의 관계 – Reply 테이블 수정

```
import lombok.*;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Builder
```

```
@AllArgsConstructor
```

```
@NoArgsConstructor
```

```
@Getter
```

```
@ToString(exclude = "board")
```

```
public class Reply extends BaseEntity{
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long rno;
```

```
    private String text;
```

```
    private String replyer;
```

```
    @ManyToOne
```

```
    private Board board;
```

```
}
```


연관 관계 와 관계형 데이터베이스

❖ @ManyToOne Annotation

- ✓ 프로젝트를 실행하고 테이블 확인



연관 관계 와 관계형 데이터베이스

❖ Repository 인터페이스 생성 – persistence 패키지 생성

- ✓ persistence 패키지에 Member Entity를 위한 MemberRepository 인터페이스 생성

```
public interface MemberRepository extends JpaRepository<Member, String> {  
    }
```
- ✓ persistence 패키지에 Board Entity를 위한 BoardRepository 인터페이스 생성

```
public interface BoardRepository extends JpaRepository<Board, Long> {  
    }
```
- ✓ persistence 패키지에 Reply Entity를 위한 ReplyRepository 인터페이스 생성

```
public interface ReplyRepository extends JpaRepository<Reply, Long> {  
    }
```

연관 관계 와 관계형 데이터베이스

❖ 테스트

- ✓ Member Entity에 데이터 100개 추가하기

```
@SpringBootTest
public class RepositoryTest {

    @Autowired
    private MemberRepository memberRepository;

    @Test
    public void insertMembers() {

        IntStream.rangeClosed(1, 100).forEach(i -> {

            Member member = Member.builder()
                .email("user" + i + "@aaa.com")
                .password("1111")
                .name("USER" + i)
                .build();

            memberRepository.save(member);
        });
    }
}
```

연관 관계 와 관계형 데이터베이스

❖ 테스트

- ✓ Board Entity에 데이터 100개 추가하기

@SpringBootTest

public class RepositoryTest {

@Autowired

private BoardRepository boardRepository;

@Test

public void insertBoard() {

 IntStream.rangeClosed(1,100).forEach(i -> {

 Member member = Member.builder().email("user"+i + "@aaa.com").build();

 Board board = Board.builder()

 .title("Title..." + i)

 .content("Content...." + i)

 .writer(member)

 .build();

 boardRepository.save(board);

 });

}

}

연관 관계 와 관계형 데이터베이스

❖ 테스트

- ✓ Reply Entity에 데이터 300개 추가하기

@SpringBootTest

public class RepositoryTest {

@Autowired

private ReplyRepository replyRepository;

@Test

public void insertReply() {

 IntStream.rangeClosed(1, 300).forEach(i -> {

 //1부터 100까지의 임의의 번호

 long bno = (long)(Math.random() * 100) + 1;

 Board board = Board.builder().bno(bno).build();

 Reply reply = Reply.builder()

 .text("Reply....." +i)

 .board(board)

 .replier("guest")

 .build();

 replyRepository.save(reply);

 });

}

}

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

- ✓ 두 개 이상의 Entity 간의 연관 관계를 맺고 나면 Entity 클래스들이 실제 데이터베이스 상에서는 두 개 이상의 테이블로 생성되기 되기 때문에 연관 관계를 맺고 있다는 것은 조회를 할 때 데이터베이스의 입장으로 보면 조인이 필요하게 되는데 @ManyToOne의 경우에는 FK 쪽의 Entity를 가져올 때 PK 쪽의 Entity도 같이 가져옴
- ✓ 하나의 Board를 가져올 때 수행되는 쿼리

@Test

```
public void readBoard() {  
    Optional<Board> result = boardRepository.findById(100L); //데이터베이스에  
    존재하는 번호  
    Board board = result.get();  
    System.out.println(board);  
    System.out.println(board.getWriter());  
}
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

- ✓ 하나의 Board를 가져올 때 수행되는 쿼리 – Outer Join

Hibernate:

```
select
    board0_.bno as bno1_0_0_,
    board0_.moddate as moddate2_0_0_,
    board0_.regdate as regdate3_0_0_,
    board0_.content as content4_0_0_,
    board0_.title as title5_0_0_,
    board0_.writer_email as writer_e6_0_0_,
    member1_.email as email1_2_1_,
    member1_.moddate as moddate2_2_1_,
    member1_.regdate as regdate3_2_1_,
    member1_.name as name4_2_1_,
    member1_.password as password5_2_1_
from
    board board0_
left join
    tbl_member member1_
        on board0_.writer_email=member1_.email
where
    board0_.bno=?
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

✓ Board Entity 수정

```
public class Board extends BaseEntity {  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long bno;  
    private String title;  
    private String content;  
  
    @ManyToOne(optional = false)  
    private Member writer;  
}
```


연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

- ✓ 하나의 Board를 가져올 때 수행되는 쿼리 – Inner Join

```
select
    b1_0.bno,
    b1_0.content,
    b1_0.moddate,
    b1_0.regdate,
    b1_0.title,
    w1_0.email,
    w1_0.moddate,
    w1_0.name,
    w1_0.password,
    w1_0.regdate
from
    board b1_0
join
    tbl_member w1_0
    on w1_0.email=b1_0.writer_email
where
    b1_0.bno=?
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

- ✓ 하나의 Reply를 가져올 때 수행되는 쿼리

@Test

```
public void readReply() {  
    Optional<Reply> result = replyRepository.findById(1L);  
    Reply reply = result.get();  
  
    System.out.println(reply);  
    System.out.println(reply.getBoard());  
}
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

- ✓ 하나의 Reply를 가져올 때 수행되는 쿼리

Hibernate:

select

```
reply0_.rno as rno1_1_0_,  
reply0_.moddate as moddate2_1_0_,  
reply0_.regdate as regdate3_1_0_,  
reply0_.board_bno as board_bn6_1_0_,  
reply0_.replyer as replyer4_1_0_,  
reply0_.text as text5_1_0_,  
board1_.bno as bno1_0_1_,  
board1_.moddate as moddate2_0_1_,  
board1_.regdate as regdate3_0_1_,  
board1_.content as content4_0_1_,  
board1_.title as title5_0_1_,  
board1_.writer_email as writer_e6_0_1_,  
member2_.email as email1_2_2_,  
member2_.moddate as moddate2_2_2_,  
member2_.regdate as regdate3_2_2_,  
member2_.name as name4_2_2_,  
member2_.password as password5_2_2_
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

- ✓ 하나의 Reply를 가져올 때 수행되는 쿼리

```
from
    reply reply0_
left join
    board board1_
        on reply0_.board_bno=board1_.bno
left join
    tbl_member member2_
        on board1_.writer_email=member2_.email
where
    reply0_.rno=?
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

✓ Eager Loading

- 쿼리 실행 결과와 같이 특정한 Entity를 조회할 때 연관 관계를 가진 모든 Entity를 같이 로딩하는 것을 Eager loading 이라고 하는데 즉시 로딩 이라고도 함
- 즉시 로딩은 한번에 연관 관계가 있는 모든 Entity를 가져온다는 장점이 있지만 여러 연관 관계를 맺고 있거나 연관 관계가 복잡할수록 조인으로 인한 성능 저하를 피할 수 없음
- JPA에서는 연관 관계의 데이터를 어떻게 가져올 것인가를 fetch(패치)라고 하는데 연관 관계 Annotation의 fetch 속성으로 모드를 지정
- 즉시 로딩은 불필요한 조인까지 처리해야 하는 경우가 많기 때문에 가능하면 사용하지 않고 그와 반대되는 개념으로 Lazy loading 으로 처리

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

✓ Lazy Loading

- Lazy loading은 지연 로딩 혹은 게으른 로딩 이라고 표현하는데 Foreign Key 부분에 fetch 모드를 설정하면 됨
- Board 클래스 수정

```
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString(exclude = "writer")
public class Board extends BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long bno;
    private String title;
    private String content;

    @ManyToOne(fetch = FetchType.LAZY)
    private Member writer;
}
```

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

✓ Lazy Loading

- 이전에 만든 Board 데이터를 읽어오는 메서드를 테스트하면 에러

@Test

```
public void readBoard() {
```

```
    Optional<Board> result = boardRepository.findById(100L); //데이터베이스에  
존재하는 번호
```

```
    Board board = result.get();
```

```
    System.out.println(board);
```

```
    System.out.println(board.getWriter());
```

```
}
```

- ◆ could not initialize proxy

[kr.co.adamsoft.board.entity.Member#user100@aaa.com] - no Session 에러
발생

- ◆ board.getWriter()를 호출하는 부분에서 에러가 발생하는데 getWriter()를
호출하기 전에 데이터베이스 연결이 해제되었기 때문인데 이 경우에는
@Transactional을 추가해서 해결

- ◆ @Transactional 은 하나의 트랜잭션으로 처리하는데 필요하다면 데이터베이스를
다시 연결해서 작업을 수행

연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

✓ Lazy Loading

- Board 데이터를 읽어오는 메서드를 수정

@Transactional

@Test

```
public void readBoard() {
```

```
    Optional<Board> result = boardRepository.findById(100L); //데이터베이스에  
    존재하는 번호
```

```
    Board board = result.get();
```

```
    System.out.println(board);
```

```
    System.out.println(board.getWriter());
```

```
}
```


연관 관계 와 관계형 데이터베이스

❖ @ManyToOne 과 Eager/Lazy Loading

✓ 연관 관계에서 @ToString()

- Entity 간에 연관 관계를 지정하는 경우에는 항상 @ToString()을 주의해야 함
- @ToString()은 해당 클래스의 모든 멤버 변수를 출력하게 되는데 Board 객체의 @ToString()을 하면 writer 변수로 선언된 Member 객체 역시 출력
- Member를 출력하기 위해서는 Member 객체의 toString()이 호출되어야 하고 이때 데이터베이스 연결이 필요하게 됨
- exclude는 해당 속성 값으로 지정된 변수는 toString()에서 제외하도록 해주는 속성
- 연관 관계가 있는 Entity 클래스의 경우 @ToString()을 할 때는 exclude 속성을 사용하는 것을 권장

✓ Lazy Loading 의 장단점

- 지연 로딩은 조인을 하지 않기 때문에 단순히 하나의 테이블을 이용하는 경우에는 빠른 속도의 처리가 가능하다는 장점이 있음
- 필요한 순간에 쿼리를 실행해야 하기 때문에 연관 관계가 복잡한 경우에는 여러 번의 쿼리가 실행된다는 단점이 있음
- 보편적인 코딩 가이드는 지연 로딩을 기본으로 사용하고 상황에 맞게 필요한 방법을 설정

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left join

- ✓ 목록 화면 게시글의 정보와 함께 댓글의 수를 같이 조회하고자 하는 경우에는 단순히 하나의 Entity Type 만으로는 해결할 수 없는데 이러한 경우에는 JPQL의 Join을 이용해서 해결
- ✓ Spring Boot 2 버전 이후에 포함되는 JPA 버전은 Entity Class 내에 연관 관계가 없더라도 join을 이용할 수 있는데 Inner JOIN 이나 JOIN 또는 Left Outer Join 이나 Left Join을 이용할 수 있음

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ Entity Class 내부에 연관 관계가 있는 경우에는 그 내부의 속성을 이용해서 조인을 처리

- BoardRepository 인터페이스에 Board 데이터를 가져올 때 Writer 데이터도 같이 가져오도록 하는 메서드 선언

```
public interface BoardRepository extends JpaRepository<Board, Long> {  
    @Query("select b, w from Board b left join b.writer w where b.bno =:bno")  
    Object getBoardWithWriter(@Param("bno") Long bno);  
}
```

- 생성된 메서드를 테스트

```
@Test  
public void testReadWithWriter() {  
    Object result = boardRepository.getBoardWithWriter(100L);  
    Object[] arr = (Object[])result;  
    System.out.println("-----");  
    System.out.println(Arrays.toString(arr));  
}
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

- ✓ Entity Class 내부에 연관 관계가 없는 경우에는 on을 이용해서 조인 조건을 작성
 - 게시물 과 해당 게시물에 속한 댓글을 조회하는 경우

◆ SQL

```
select
    board.bno, board.title, board.writer_email,
    rno, text
from board left outer join reply
    on reply.board_bno = board.bno
where board.bno = 100;
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ Entity Class 내부에 연관 관계가 없는 경우에는 on을 이용해서 조인 조건을 작성

● 게시물 과 해당 게시물에 속한 댓글을 조회하는 경우

◆ BoardRepository 인터페이스에서 메서드 생성

```
@Query("SELECT b, r FROM Board b LEFT JOIN Reply r ON r.board = b  
WHERE b.bno = :bno")
```

```
List<Object[]> getBoardWithReply(@Param("bno") Long bno);
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ Entity Class 내부에 연관 관계가 없는 경우에는 on을 이용해서 조인 조건을 작성

- 게시물 과 해당 게시물에 속한 댓글을 조회하는 경우

◆ BoardRepository 인터페이스에서 메서드 테스트

```
@Test
public void testGetBoardWithReply() {
    List<Object[]> result = boardRepository.getBoardWithReply(100L);
    for (Object[] arr : result) {
        System.out.println(Arrays.toString(arr));
    }
}
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ Entity Class 내부에 연관 관계가 없는 경우에는 on을 이용해서 조인 조건을 작성

- 게시물 과 해당 게시물에 속한 댓글을 조회하는 경우

◆ 결과

```
[Board(bno=100, title=Title...100, content=Content...100), Reply(rno=53,
text=Reply.....53, replyer=guest)]
[Board(bno=100, title=Title...100, content=Content...100), Reply(rno=146,
text=Reply.....146, replyer=guest)]
[Board(bno=100, title=Title...100, content=Content...100), Reply(rno=201,
text=Reply.....201, replyer=guest)]
[Board(bno=100, title=Title...100, content=Content...100), Reply(rno=297,
text=Reply.....297, replyer=guest)]
```

연관 관계 와 관계형 데이터베이스

- ❖ JPQL 과 left outer join
 - ✓ 목록 화면

Board List Page [REGISTER](#)

----- ▾

Search Clear

#	Title	Writer	Regdate
102	첫번째 글 ----- [0]	USER30 user30@aaa.com	2022/01/15
101	Test. ----- [0]	USER55 user55@aaa.com	2022/01/15
100	Title...100 ----- [4]	USER100 user100@aaa.com	2022/01/14
99	Title...99 ----- [2]	USER99 user99@aaa.com	2022/01/14
98	Title...98 ----- [2]	USER98 user98@aaa.com	2022/01/14
97	Title...97 ----- [0]	USER97 user97@aaa.com	2022/01/14
96	Title...96 ----- [5]	USER96 user96@aaa.com	2022/01/14
95	Title...95 ----- [6]	USER95 user95@aaa.com	2022/01/14
94	Title...94 ----- [0]	USER94 user94@aaa.com	2022/01/14
93	Title...93 ----- [4]	USER93 user93@aaa.com	2022/01/14

1 2 3 4 5 6 7 8 9 10 Next

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ 목록 화면에 필요한 JPQL 생성

● 필요한 데이터

- ◆ Board: 게시물 번호, 제목, 작성 시간
- ◆ Member: 회원의 이름 / 이메일
- ◆ Reply: 해당 게시물의 댓글 수

● BoardRepository 인터페이스에 메서드 생성

```
@Query(value ="SELECT b, w, count(r) " +  
        " FROM Board b " +  
        " LEFT JOIN b.writer w " +  
        " LEFT JOIN Reply r ON r.board = b " +  
        " GROUP BY b",  
        countQuery ="SELECT count(b) FROM Board b")  
Page<Object[]> getBoardWithReplyCount(Pageable pageable);
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ 목록 화면에 필요한 JPQL 생성

● 테스트 메서드를 만들어서 확인

```
@Test
public void testWithReplyCount(){
    Pageable pageable = PageRequest.of(0,10, Sort.by("bno").descending());

    Page<Object[]> result =
boardRepository.getBoardWithReplyCount(pageable);
    result.get().forEach(row -> {
        Object[] arr = (Object[])row;
        System.out.println(Arrays.toString(arr));
    });
}
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ 목록 화면에 필요한 JPQL 생성

- 오라클은 GROUP BY를 사용할 때 GROUP BY에 사용한 절 과 집계 함수로 집계를 하는 항목만을 SELECT 절에 사용할 수 있으므로 메서드를 수정해야 함 - 이후 모든 곳에서 동일한 작업을 수행해야 함
- BoardRepository 인터페이스에 메서드 생성

```
@Query(value ="SELECT b, min(w.name), min(w.email), count(r) " +  
        " FROM Board b " +  
        " LEFT JOIN b.writer w " +  
        " LEFT JOIN Reply r ON r.board = b " +  
        " GROUP BY b",  
        countQuery ="SELECT count(b) FROM Board b")  
Page<Object[]> getBoardWithReplyCount(Pageable pageable);
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ 목록 화면에 필요한 JPQL 생성

● 결과

```
[Board(bno=100, title=Title...100, content=Content....100),  
Member(email=user100@aaa.com, password=1111, name=USER100), 4]  
[Board(bno=99, title=Title...99, content=Content....99),  
Member(email=user99@aaa.com, password=1111, name=USER99), 4]  
[Board(bno=98, title=Title...98, content=Content....98),  
Member(email=user98@aaa.com, password=1111, name=USER98), 5]  
[Board(bno=97, title=Title...97, content=Content....97),  
Member(email=user97@aaa.com, password=1111, name=USER97), 1]  
[Board(bno=96, title=Title...96, content=Content....96),  
Member(email=user96@aaa.com, password=1111, name=USER96), 2]  
[Board(bno=95, title=Title...95, content=Content....95),  
Member(email=user95@aaa.com, password=1111, name=USER95), 5]  
[Board(bno=94, title=Title...94, content=Content....94),  
Member(email=user94@aaa.com, password=1111, name=USER94), 4]  
[Board(bno=93, title=Title...93, content=Content....93),  
Member(email=user93@aaa.com, password=1111, name=USER93), 0]  
[Board(bno=92, title=Title...92, content=Content....92),  
Member(email=user92@aaa.com, password=1111, name=USER92), 3]  
[Board(bno=91, title=Title...91, content=Content....91),  
Member(email=user91@aaa.com, password=1111, name=USER91), 3]
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ 목록 화면에 필요한 JPQL 생성

● BoardRepository 인터페이스에 메서드 선언

```
@Query("SELECT b, w, count(r) " +  
        " FROM Board b LEFT JOIN b.writer w " +  
        " LEFT OUTER JOIN Reply r ON r.board = b" +  
        " WHERE b.bno = :bno")  
Object getBoardByBno(@Param("bno") Long bno);
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ 목록 화면에 필요한 JPQL 생성

● 테스트 메서드를 만들어서 확인

```
@Test
public void testWithDetail(){
    Object result = boardRepository.getBoardByBno(100L);
    Object[] arr = (Object[])result;
    System.out.println(Arrays.toString(arr));
}
```

연관 관계 와 관계형 데이터베이스

❖ JPQL 과 left outer join

✓ 목록 화면에 필요한 JPQL 생성

● 결과

```
[Board(bno=100, title=Title...100, content=Content....100),  
Member(email=user100@aaa.com, password=1111, name=USER100), 4]
```

DTO 계층 과 Service 계층

❖ DTO 계층 과 Service 계층 작성

- ✓ 프로젝트의 기본 패키지 안에 dto 와 service 패키지를 작성
- ✓ dto 패키지에 BoardDTO 클래스를 생성하고 작성

```
import lombok.*;
import java.time.LocalDateTime;

@Data
@ToString
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class BoardDTO {
    private Long bno;
    private String title;
    private String content;
    private String writerEmail; //작성자의 이메일(id)
    private String writerName; //작성자의 이름
    private LocalDateTime regDate;
    private LocalDateTime modDate;
    private int replyCount; //해당 게시글의 댓글 수
}
```


DTO 계층 과 Service 계층

❖ DTO 계층 과 Service 계층 작성

- ✓ service 패키지에 BoardService 인터페이스를 생성하고 DTO 와 Entity 사이의 변환을 위한 메서드를 작성

```
public interface BoardService {  
    default Board dtoToEntity(BoardDTO dto){  
  
        Member member = Member.builder().email(dto.getWriterEmail()).build();  
  
        Board board = Board.builder()  
            .bno(dto.getBno())  
            .title(dto.getTitle())  
            .content(dto.getContent())  
            .writer(member)  
            .build();  
        return board;  
    }  
}
```

DTO 계층 과 Service 계층

❖ DTO 계층 과 Service 계층 작성

- ✓ service 패키지에 BoardService 인터페이스를 생성하고 DTO 와 Entity 사이의 변환을 위한 메서드를 작성

```
default BoardDTO entityToDTO(Board board, Member member, Long replyCount) {
```

```
    BoardDTO boardDTO = BoardDTO.builder()
        .bno(board.getBno())
        .title(board.getTitle())
        .content(board.getContent())
        .regDate(board.getRegDate())
        .modDate(board.getModDate())
        .writerEmail(member.getEmail())
        .writerName(member.getName())
        .replyCount(replyCount.intValue()) //int로 처리하도록
        .build();
```

```
    return boardDTO;
```

```
    }
}
```

DTO 계층 과 Service 계층

❖ 게시물 등록

- ✓ 게시물 등록은 BoardDTO 타입을 파라미터로 전달받고 생성된 게시물의 번호를 반환하도록 작성
- ✓ BoardService 인터페이스에 게시물 등록을 위한 메서드를 선언
`Long register(BoardDTO dto);`

DTO 계층 과 Service 계층

❖ 게시물 등록

- ✓ service 패키지에 BoardServiceImpl 클래스를 생성하고 게시물 등록 메서드를 구현

```
@Service
@RequiredArgsConstructor
@Log4j2
public class BoardServiceImpl implements BoardService{
    private final BoardRepository repository;

    @Override
    public Long register(BoardDTO dto) {
        log.info(dto);
        Board board = dtoToEntity(dto);
        repository.save(board);
        return board.getBno();
    }
}
```

DTO 계층 과 Service 계층

❖ 게시물 등록

- ✓ BoardService를 위한 Test 클래스를 생성하고 게시물 등록 메서드를 테스트

```
@SpringBootTest  
public class ServiceTest {
```

```
    @Autowired  
    private BoardService boardService;
```

```
    @Test  
    public void testRegister() {
```

```
        BoardDTO dto = BoardDTO.builder()  
            .title("Test.")  
            .content("Test...")  
            .writerEmail("user55@aaa.com") //현재 데이터베이스에 존재하는 회원  
            .build();  
  
        Long bno = boardService.register(dto);  
        System.out.println(bno);  
    }  
}
```

이메일

DTO 계층 과 Service 계층

❖ 게시물 목록 보기

- ✓ 게시물 목록을 가져오는데 사용할 요청 클래스 생성 – dto.PageRequestDTO

```
@Builder
@AllArgsConstructor
@Data
public class PageRequestDTO {

    private int page;
    private int size;
    private String type;
    private String keyword;

    public PageRequestDTO(){
        this.page = 1;
        this.size = 10;
    }

    public Pageable getPageable(Sort sort){
        return PageRequest.of(page -1, size, sort);
    }
}
```

DTO 계층 과 Service 계층

❖ 게시물 목록 보기

- ✓ 게시물 목록의 결과로 사용할 응답 클래스 생성 – dto.PageResultDTO

@Data

```
public class PageResultDTO<DTO, EN> {
```

```
    //DTO리스트
```

```
    private List<DTO> dtoList;
```

```
    //총 페이지 번호
```

```
    private int totalPage;
```

```
    //현재 페이지 번호
```

```
    private int page;
```

```
    //목록 사이즈
```

```
    private int size;
```

```
    //시작 페이지 번호, 끝 페이지 번호
```

```
    private int start, end;
```

```
    //이전, 다음
```

```
    private boolean prev, next;
```

```
    //페이지 번호 목록
```

```
    private List<Integer> pageList;
```

DTO 계층 과 Service 계층

❖ 게시물 목록 보기

- ✓ 게시물 목록의 결과로 사용할 응답 클래스 생성 – dto.PageResultDTO

```
public PageResultDTO(Page<EN> result, Function<EN,DTO> fn ){  
    dtoList = result.stream().map(fn).collect(Collectors.toList());  
    totalPage = result.getTotalPages();  
    makePageList(result.getPageable());  
}
```

```
private void makePageList(Pageable pageable){  
    this.page = pageable.getPageNumber() + 1; // 0부터 시작하므로 1을 추가  
    this.size = pageable.getPageSize();
```

```
    //temp end page  
    int tempEnd = (int)(Math.ceil(page/10.0)) * 10;  
    start = tempEnd - 9;  
    prev = start > 1;
```

```
    end = totalPage > tempEnd ? tempEnd: totalPage;  
    next = totalPage > tempEnd;  
    pageList = IntStream.rangeClosed(start, end).boxed().collect(Collectors.toList());
```

```
}
```

```
}
```


DTO 계층 과 Service 계층

❖ 게시물 목록 보기

- ✓ BoardService 인터페이스에 게시물 목록 요청을 처리하기 위한 메서드 선언

```
PageResultDTO<BoardDTO, Object[]> getList(PageRequestDTO pageRequestDTO);
```

DTO 계층 과 Service 계층

❖ 게시물 목록 보기

- ✓ BoardServiceImpl 클래스에 게시물 목록 요청을 처리하기 위한 메서드 구현

```
@Override
public PageResultDTO<BoardDTO, Object[]> getList(PageRequestDTO pageRequestDTO)
{

    log.info(pageRequestDTO);

    Function<Object[], BoardDTO> fn = (en ->
    entityToDTO((Board)en[0],(Member)en[1],(Long)en[2]));

    Page<Object[]> result = repository.getBoardWithReplyCount(
        pageRequestDTO.getPageable(Sort.by("bno").descending()) );

    return new PageResultDTO<>(result, fn);
}
```

DTO 계층 과 Service 계층

❖ 게시물 목록 보기

- ✓ ServiceTest 클래스에 게시물 목록 요청을 처리하기 위한 메서드 테스트

```
@Test
public void testList() {

    //1페이지 10개
    PageRequestDTO pageRequestDTO = new PageRequestDTO();

    PageResultDTO<BoardDTO, Object[]> result =
    boardService.getList(pageRequestDTO);

    for (BoardDTO boardDTO : result.getDtoList()) {
        System.out.println(boardDTO);
    }

}
```

DTO 계층 과 Service 계층

❖ 게시물 상세 보기

✓ BoardService 인터페이스에 게시물 상세 보기를 처리하기 위한 메서드 선언
BoardDTO get(Long bno);

✓ BoardServiceImpl 클래스에 게시물 상세 보기를 처리하기 위한 메서드 구현
@Override
public BoardDTO get(Long bno) {
 Object result = repository.getBoardByBno(bno);
 Object[] arr = (Object[])result;
 return entityToDTO((Board)arr[0], (Member)arr[1], (Long)arr[2]);
}

DTO 계층 과 Service 계층

❖ 게시물 상세 보기

- ✓ ServiceTest 클래스에 게시물 상세 보기를 처리하기 위한 메서드 테스트

```
@Test
public void testGet() {
    Long bno = 100L;
    BoardDTO boardDTO = boardService.get(bno);
    System.out.println(boardDTO);
}
```

DTO 계층 과 Service 계층

❖ 게시물 삭제

- ✓ 삭제를 만들 때는 연관된 데이터를 같이 삭제할 것인지에 대해서 고민을 해야 하는데 실제 Service에서는 대부분 state 컬럼을 추가해서 삭제를 요청하면 삭제를 하지 않고 상태를 이용해서 삭제된 것 과 동일한 효과를 만드는 경우가 많음
- ✓ ReplyRespository 인터페이스에 글 번호를 이용해서 댓글을 삭제하는 메서드를 선언

@Modifying

@Query("delete from Reply r where r.board.bno =:bno ")

void deleteByBno(@Param("bno") Long bno);

DTO 계층 과 Service 계층

❖ 게시물 삭제

- ✓ BoardService 인터페이스에 글 번호를 이용해서 게시글을 삭제하는 메서드를 선언
void removeWithReplies(Long bno);
- ✓ BoardServiceImpl 클래스에 글 번호를 이용해서 게시글을 삭제하는 메서드를 구현
private final ReplyRepository replyRepository;

```
@Transactional
@Override
public void removeWithReplies(Long bno) {
    //댓글 부터 삭제
    replyRepository.deleteByBno(bno);
    repository.deleteById(bno);
}
```

DTO 계층 과 Service 계층

❖ 게시물 삭제

- ✓ ServiceTest 클래스에 삭제를 위한 테스트를 만들어서 테스트

```
@Test  
public void testRemove() {  
    boardService.removeWithReplies(1L);  
}
```


DTO 계층 과 Service 계층

❖ 게시물 수정

- ✓ Board를 수정할 수 있도록 Board 클래스에 제목 과 내용을 수정할 수 있도록 메서드 추가

```
public void changeTitle(String title){  
    this.title = title;  
}
```

```
public void changeContent(String content){  
    this.content = content;  
}
```

DTO 계층 과 Service 계층

❖ 게시물 수정

- ✓ BoardService 인터페이스에 수정을 위한 메서드를 선언

```
void modify(BoardDTO boardDTO);
```

- ✓ BoardServiceImpl 클래스에 수정을 위한 메서드를 구현

```
@Transactional
```

```
@Override
```

```
public void modify(BoardDTO boardDTO) {  
    Optional<Board> board = repository.findById(boardDTO.getBno());  
  
    if(board.isPresent()) {  
        board.get().changeTitle(boardDTO.getTitle());  
        board.get().changeContent(boardDTO.getContent());  
  
        repository.save(board.get());  
    }  
}
```

DTO 계층 과 Service 계층

❖ 게시물 수정

- ✓ ServiceTest 클래스에 수정을 위한 테스트를 만들어서 테스트

@Test

public void testModify() {

```
BoardDTO boardDTO = BoardDTO.builder()
    .bno(2L)
    .title("제목 변경합니다.2")
    .content("내용 변경합니다.2")
    .build();
```

```
boardService.modify(boardDTO);
```

```
}
```

Controller 계층 과 View 계층

❖ View 계층을 위한 기본 설정

- ✓ View 계층에 공통된 디자인을 적용하기 위해서 이전 프로젝트에서 static 디렉토리의 내용을 복사
- ✓ templates 디렉토리에 layout 디렉토리에 basic.html 파일을 복사



Controller 계층 과 View 계층

- ❖ controller 패키지에 BoardController를 생성하고 기본 코드 작성

```
@Controller
@Log4j2
@RequiredArgsConstructor
public class BoardController {
    private final BoardService boardService;

}
```

Controller 계층 과 View 계층

❖ 목록 보기

Board List Page [REGISTER](#)

..... ▾

Search Clear

#	Title	Writer	Regdate
102	첫번째 글 [0]	USER30 user30@aaa.com	2022/01/15
101	Test. [0]	USER55 user55@aaa.com	2022/01/15
100	Title...100 [4]	USER100 user100@aaa.com	2022/01/14
99	Title...99 [2]	USER99 user99@aaa.com	2022/01/14
98	Title...98 [2]	USER98 user98@aaa.com	2022/01/14
97	Title...97 [0]	USER97 user97@aaa.com	2022/01/14
96	Title...96 [5]	USER96 user96@aaa.com	2022/01/14
95	Title...95 [6]	USER95 user95@aaa.com	2022/01/14
94	Title...94 [0]	USER94 user94@aaa.com	2022/01/14
93	Title...93 [4]	USER93 user93@aaa.com	2022/01/14

1 2 3 4 5 6 7 8 9 10 Next

Controller 계층 과 View 계층

❖ 목록 보기

- ✓ BoardController에 목록 보기를 위한 요청 처리 메서드를 작성

```
@GetMapping("/{", "/board/list"})
public String list(PageRequestDTO pageRequestDTO, Model domain){
    log.info("list....." + pageRequestDTO);
    domain.addAttribute("result", boardService.getList(pageRequestDTO));
    return "board/list";
}
```

Controller 계층 과 View 계층

❖ 목록 보기

- ✓ templates 디렉토리에 board 디렉토리를 생성하고 list.html 파일을 생성하고 작성

```
<!DOCTYPE html>
```

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">
```

```
<th:block th:fragment="content">
```

```
<h1 class="mt-4">Board List Page
```

```
<span>
```

```
<a th:href="@{/board/register}">
```

```
<button type="button" class="btn btn-outline-primary">REGISTER
```

```
</button>
```

```
</a>
```

```
</span>
```

```
</h1>
```


Controller 계층 과 View 계층

❖ 목록 보기

- ✓ templates 디렉토리에 board 디렉토리를 생성하고 list.html 파일을 생성하고 작성

```
<form action="/board/list" method="get" id="searchForm">
  <div class="input-group">
    <input type="hidden" name="page" value = "1">
    <div class="input-group-prepend">
      <select class="custom-select" id="type" name="type">
        <option th:selected="${pageRequestDTO.type == null}">-----</option>
        <option value="t" th:selected="${pageRequestDTO.type
== 't'}" >제목</option>
        <option value="c" th:selected="${pageRequestDTO.type
== 'c'}" >내용</option>
        <option value="w" th:selected="${pageRequestDTO.type
== 'w'}" >작성자</option>
        <option value="tc" th:selected="${pageRequestDTO.type == 'tc'}" >제목 +
내용</option>
        <option value="tcw" th:selected="${pageRequestDTO.type == 'tcw'}" >제목 +
내용 + 작성자</option>
      </select>
    </div>
  </div>
```

Controller 계층 과 View 계층

❖ 목록 보기

- ✓ templates 디렉토리에 board 디렉토리를 생성하고 list.html 파일을 생성하고 작성

```
<input class="form-control" id="keyword" name="keyword"
th:value="${pageRequestDTO.keyword}">
<div class="input-group-append" id="button-addon4">
  <button class="btn btn-outline-secondary btn-search" id='btn-search'
type="button">Search</button>
  <button class="btn btn-outline-secondary btn-clear" id='btn-clear'
type="button">Clear</button>
</div>
</div>
</form>
```

Controller 계층 과 View 계층

❖ 목록 보기

- ✓ templates 디렉토리에 board 디렉토리를 생성하고 list.html 파일을 생성하고 작성

```
<table class="table table-striped">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">Title</th>
      <th scope="col">Writer</th>
      <th scope="col">Regdate</th>
    </tr>
  </thead>
```

Controller 계층 과 View 계층

❖ 목록 보기

- ✓ templates 디렉토리에 board 디렉토리를 생성하고 list.html 파일을 생성하고 작성

```
<tbody>
<tr th:each="dto : ${result.dtoList}" >
  <th scope="row">
    <a th:href="@{/board/read(bno = ${dto.bno},
      page= ${result.page},
      type=${pageRequestDTO.type} ,
      keyword = ${pageRequestDTO.keyword})}">
      [[${dto.bno}]]
    </a>
  </th>
  <td>[[${dto.title}]] ----- [ <b th:text="${dto.replyCount}"> </b>] </td>
  <td>[[${dto.writerName}]] <small>[[${dto.writerEmail}]] </small> </td>
  <td>[[${#temporals.format(dto.regDate, 'yyyy/MM/dd')}]<td>]] </td>
</tr>
</tbody>
</table>
```

Controller 계층 과 View 계층

❖ 목록 보기

- ✓ templates 디렉토리에 board 디렉토리를 생성하고 list.html 파일을 생성하고 작성

```
<ul class="pagination h-100 justify-content-center align-items-center">
  <li class="page-item " th:if="${result.prev}">
    <a class="page-link" th:href="@{/board/list(page= ${result.start -1},
      type=${pageRequestDTO.type} ,
      keyword = ${pageRequestDTO.keyword} ) }" tabindex="-
1">Previous</a>
  </li>
  <li th:class=" 'page-item ' + ${result.page == page?'active':''}" th:each="page:
${result.pageList}">
    <a class="page-link" th:href="@{/board/list(page = ${page} ,
      type=${pageRequestDTO.type} ,
      keyword = ${pageRequestDTO.keyword} )}">
      [[${page}]]
    </a>
  </li>
  <li class="page-item" th:if="${result.next}">
    <a class="page-link" th:href="@{/board/list(page= ${result.end + 1} ,
      type=${pageRequestDTO.type} ,
      keyword = ${pageRequestDTO.keyword} )}">Next</a>
  </li>
</ul>
```

Controller 계층 과 View 계층

❖ 목록 보기

- ✓ templates 디렉토리에 board 디렉토리를 생성하고 list.html 파일을 생성하고 작성

```
<script th:inline="javascript">  
  let msg = [{${msg}}];  
  console.log(msg);
```

```
  if(msg){  
    alert(msg);  
  }
```

```
  let searchForm = document.getElementById("searchForm");
```

```
  document.getElementById('btn-search').addEventListener('click', (e) => {  
    searchForm.submit();  
  });
```

```
  document.getElementById('btn-clear').addEventListener('click', (e) => {  
    document.getElementById("keyword").value = "";  
    document.getElementById("type").value = "";  
    searchForm.submit();  
  });
```

```
</script>
```

Controller 계층 과 View 계층

❖ 게시물 등록

Board Register Page

Title

Content

Writer Email

Controller 계층 과 View 계층

❖ 게시물 등록

- ✓ BoardController 클래스에 게시물 등록을 위한 메서드를 생성

```
@GetMapping("/board/register")
public void register(){
    log.info("register get...");
}
```

```
@PostMapping("/board/register")
public String registerPost(BoardDTO dto, RedirectAttributes redirectAttributes){
    log.info("dto..." + dto);
    //새로 추가된 Entity의 번호
    Long bno = boardService.register(dto);
    log.info("BNO: " + bno);
    redirectAttributes.addFlashAttribute("msg", bno + " 삽입 성공");
    return "redirect:/board/list";
}
```


Controller 계층 과 View 계층

❖ 게시물 등록

- ✓ templates/board 디렉토리에 register.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">

<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">
  <th:block th:fragment="content">
    <h1 class="mt-4">Board Register Page</h1>

    <form th:action="@{/board/register}" th:method="post">
      <div class="form-group">
        <label >Title</label>
        <input type="text" class="form-control" name="title" placeholder="Enter Title">
      </div>
      <div class="form-group">
        <label >Content</label>
        <textarea class="form-control" rows="5" name="content"> </textarea>
      </div>
      <div class="form-group">
        <label >Writer Email</label>
        <input type="email" class="form-control" name="writerEmail"
placeholder="Writer Email ">
      </div>
```

Controller 계층 과 View 계층

❖ 게시물 등록

- ✓ templates/board 디렉토리에 register.html 파일을 생성하고 작성

```
<button type="submit" class="btn btn-primary">Submit</button>  
</form>
```

```
</th:block>
```

```
</th:block>
```



Controller 계층 과 View 계층

❖ 게시물 상세 보기

Board Read Page

Bno

102

Title

첫번째 글

Content

안녕하세요

Writer

USER30

RegDate

2022/01/15 11:53:34

ModDate

2022/01/15 11:53:34

Modify

List

Controller 계층 과 View 계층

❖ 게시물 상세 보기

- ✓ BoardController 클래스에서 상세보기 요청을 처리하기 위한 메서드 생성

```
@GetMapping("/board/read")  
public void read(@ModelAttribute("requestDTO") PageRequestDTO pageRequestDTO,  
Long bno, Model domain){  
    log.info("bno: " + bno);  
    BoardDTO boardDTO = boardService.get(bno);  
    log.info(boardDTO);  
    domain.addAttribute("dto", boardDTO);  
}
```

Controller 계층 과 View 계층

❖ 게시물 상세 보기

- ✓ board 디렉토리에 read.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">
  <th:block th:fragment="content">
    <h1 class="mt-4">Board Read Page</h1>

    <div class="form-group">
      <label >Bno</label>
      <input type="text" class="form-control" name="gno" th:value="${dto.bno}"
readonly >
    </div>
    <div class="form-group">
      <label >Title</label>
      <input type="text" class="form-control" name="title" th:value="${dto.title}"
readonly >
    </div>
    <div class="form-group">
      <label >Content</label>
      <textarea class="form-control" rows="5" name="content"
readonly>[[${dto.content}]]</textarea>
    </div>
```

Controller 계층 과 View 계층

❖ 게시물 상세 보기

- ✓ board 디렉토리에 read.html 파일을 생성하고 작성

```
<div class="form-group">
  <label >Writer</label>
  <input type="text" class="form-control" name="writer"
th:value="${dto.writerName}" readonly>
</div>
<div class="form-group">
  <label >RegDate</label>
  <input type="text" class="form-control" name="regDate"
th:value="${#temporals.format(dto.regDate, 'yyyy/MM/dd HH:mm:ss')}" readonly>
</div>
<div class="form-group">
  <label >ModDate</label>
  <input type="text" class="form-control" name="modDate"
th:value="${#temporals.format(dto.modDate, 'yyyy/MM/dd HH:mm:ss')}" readonly>
</div>
```

Controller 계층 과 View 계층

❖ 게시물 상세 보기

- ✓ board 디렉토리에 read.html 파일을 생성하고 작성

```
<a th:href="@{/board/modify(bno = ${dto.bno}, page=${requestDTO.page},  
type=${requestDTO.type}, keyword = ${requestDTO.keyword})}">  
    <button type="button" class="btn btn-primary">수정</button>  
</a>
```

```
<a th:href="@{/board/list(page=${requestDTO.page} , type=${requestDTO.type},  
keyword = ${requestDTO.keyword})}">  
    <button type="button" class="btn btn-info">목록</button>  
</a>
```

```
</th:block>
```

```
</th:block>
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

Board Modify Page

Bno

102

Title

첫번째 글

Content

안녕하세요

Writer

user30@aaa.com

RegDate

2022/01/15 11:53:34

ModDate

2022/01/15 11:53:34

수정

목록

삭제

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ BaordController의 상세보기 요청 처리 메서드를 수정

@GetMapping({"/board/read", "/board/modify"})

```
public void read(@ModelAttribute("requestDTO") PageRequestDTO pageRequestDTO,  
Long bno, Model domain){  
    log.info("bno: " + bno);  
    BoardDTO boardDTO = boardService.get(bno);  
    log.info(boardDTO);  
    domain.addAttribute("dto", boardDTO);  
}
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ BaordController에 수정 과 삭제 처리 메서드를 추가

```
@PostMapping("/board/modify")
public String modify(BoardDTO dto,
                    @ModelAttribute("requestDTO") PageRequestDTO requestDTO,
                    RedirectAttributes redirectAttributes){
    log.info("post modify.....");
    log.info("dto: " + dto);

    boardService.modify(dto);

    redirectAttributes.addAttribute("page",requestDTO.getPage());
    redirectAttributes.addAttribute("type",requestDTO.getType());
    redirectAttributes.addAttribute("keyword",requestDTO.getKeyword());

    redirectAttributes.addAttribute("bno",dto.getBno());

    return "redirect:/board/read";
}
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ BaordController에 수정 과 삭제 처리 메서드를 추가

```
@PostMapping("/board/remove")
public String remove(long bno, RedirectAttributes redirectAttributes){
    log.info("bno: " + bno);
    boardService.removeWithReplies(bno);
    redirectAttributes.addFlashAttribute("msg", bno + " 번 삭제");
    return "redirect:/board/list";
}
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ borad 디렉토리에 modify.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">
  <th:block th:fragment="content">
    <h1 class="mt-4">Board Modify Page</h1>
    <form action="/board/modify" method="post" id="form">

      <!--페이지 번호 -->
      <input type="hidden" name="page" id="page" th:value="${requestDTO.page}">
      <input type="hidden" name="type" id="type" th:value="${requestDTO.type}" >
      <input type="hidden" name="keyword" id="keyword"
th:value="${requestDTO.keyword}" >
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ borad 디렉토리에 modify.html 파일을 생성하고 작성

```
<div class="form-group">
  <label >Bno</label>
  <input type="text" class="form-control" name="bno" th:value="${dto.bno}"
readonly >
</div>

<div class="form-group">
  <label>Title</label>
  <input type="text" class="form-control" name="title" th:value="${dto.title}" >
</div>
<div class="form-group">
  <label >Content</label>
  <textarea class="form-control" rows="5"
name="content">[[${dto.content}]]</textarea>
</div>
<div class="form-group">
  <label >Writer</label>
  <input type="text" class="form-control" name="writer"
th:value="${dto.writerEmail}" readonly>
</div>
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ borad 디렉토리에 modify.html 파일을 생성하고 작성

```
<div class="form-group">
  <label >RegDate</label>
  <input type="text" class="form-control"
th:value="${#temporals.format(dto.regDate, 'yyyy/MM/dd HH:mm:ss')}" readonly>
</div>
<div class="form-group">
  <label >ModDate</label>
  <input type="text" class="form-control"
th:value="${#temporals.format(dto.modDate, 'yyyy/MM/dd HH:mm:ss')}" readonly>
</div>
</form>

<button type="button" class="btn btn-primary modifyBtn"
id="modifyBtn">수정</button>
<button type="button" class="btn btn-info listBtn" id="listBtn">목록</button>
<button type="button" class="btn btn-danger removeBtn"
id="removeBtn">삭제</button>
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ borad 디렉토리에 modify.html 파일을 생성하고 작성

```
<script th:inline="javascript">
  let actionForm = document.getElementById("form"); //form 태그 객체

  document.getElementById("removeBtn").addEventListener('click', (e) => {
    if(!confirm("삭제하시겠습니까?")){
      return ;
    }
    actionForm.action = "/board/remove";
    actionForm.method = "post";
    actionForm.submit();
  });

  document.getElementById("modifyBtn").addEventListener('click', (e) => {
    if(!confirm("수정하시겠습니까?")){
      return ;
    }
    actionForm.action = "/board/modify";
    actionForm.method = "post";
    actionForm.submit();
  });
```

Controller 계층 과 View 계층

❖ 게시물 수정/삭제

- ✓ borad 디렉토리에 modify.html 파일을 생성하고 작성

```
document.getElementById("listBtn").addEventListener('click', (e) => {  
  let page = document.getElementById('page').value;  
  let type = document.getElementById('type').value;  
  let keyword = document.getElementById('keyword').value;  
  
  actionForm.reset(); //form 태그의 모든 내용을 지우고  
  
  actionForm.action = '/board/list?page=' + page + '&type=' + type +  
'&keyword=' + keyword;  
  actionForm.method = 'get';  
  
  actionForm.submit();  
})  
</script>  
  
</th:block>  
</th:block>
```


Querydsl

❖ Querydsl

- ✓ Sprint Boot 2.x 프로젝트의 경우 build.gradle 수정

```
buildscript {  
    ext {  
        queryDslVersion = "5.0.0"  
    }  
}
```

```
plugins {  
    id 'org.springframework.boot' version '2.6.3'  
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'  
    id 'java'  
    id "com.ewerk.gradle.plugins.querydsl" version "1.0.10"  
}
```

```
group = 'com.adamsoft'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = '11'
```

Querydsl

❖ Querydsl

- ✓ Sprint Boot 2.x 프로젝트의 경우 build.gradle 수정

```
configurations {  
    compileOnly {  
        extendsFrom annotationProcessor  
    }  
}  
  
repositories {  
    mavenCentral()  
}
```

Querydsl

❖ Querydsl

- ✓ Sprint Boot 2.x 프로젝트의 경우 build.gradle 수정

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    compileOnly 'org.projectlombok:lombok'  
    developmentOnly 'org.springframework.boot:spring-boot-devtools'  
    runtimeOnly 'com.oracle.database.jdbc:ojdbc8'  
    runtimeOnly 'mysql:mysql-connector-java'  
    annotationProcessor 'org.projectlombok:lombok'  
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
  
    implementation group: 'org.thymeleaf.extras', name: 'thymeleaf-extras-java8time '  
  
    // QueryDSL  
    implementation "com.querydsl:querydsl-jpa:${queryDslVersion}"  
    implementation "com.querydsl:querydsl-apt:${queryDslVersion}"  
}
```

Querydsl

❖ Querydsl

- ✓ Sprint Boot 2.x 프로젝트의 경우 build.gradle 수정

```
def querydslDir = "$buildDir/generated/querydsl"

querydsl {
    jpa = true
    querydslSourcesDir = querydslDir
}
sourceSets {
    main.java.srcDir querydslDir
}
configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
    querydsl.extendsFrom compileClasspath
}
compileQuerydsl {
    options.annotationProcessorPath = configurations.querydsl
}

tasks.named('test') {
    useJUnitPlatform()
}
```

Querydsl

❖ Querydsl

✓ Sprint Boot 2.x 프로젝트의 경우

- 프로젝트를 선택하고 마우스 오른쪽을 누르고 [Gradle] – [Refresh Gradle Project]
- Gradle Task 에서 build 와 jar를 실행



Querydsl

❖ Querydsl

✓ Sprint Boot 3.x 프로젝트의 경우 build.gradle 수정

- dependencies에 추가

```
// == 스프링 부트 3.0 이상 ==  
implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'  
annotationProcessor "com.querydsl:querydsl-  
apt:${dependencyManagement.importedProperties['querydsl.version']}:jakarta"  
annotationProcessor "jakarta.annotation:jakarta.annotation-api"  
annotationProcessor "jakarta.persistence:jakarta.persistence-api"
```

JPQL를 이용한 검색

❖ Repository 확장

✓ 단계

- 쿼리 메서드나 @Query 등으로 처리할 수 없는 기능은 별도의 인터페이스로 설계
- 별도의 인터페이스에 대한 구현 클래스를 QuerydslRepositorySupport라는 클래스로부터 상속을 받아서 작성
- 구현 클래스에 인터페이스의 기능을 Q도메인 클래스와 JPQLQuery를 이용해서 구현

JPQL를 이용한 검색

❖ Repository 확장

✓ QuerydslRepositorysupport 동작 확인

- persistence 패키지에 SearchBoardRepository 인터페이스를 생성

```
public interface SearchBoardRepository {  
}
```

- persistence 패키지에 SearchBoardRepository 인터페이스를 구현한 SearchBoardRepositoryImpl 클래스를 생성

```
public class SearchBoardRepositoryImpl extends QuerydslRepositorySupport  
implements SearchBoardRepository {  
    public SearchBoardRepositoryImpl() {  
        super(Board.class);  
    }  
}
```


JPQL를 이용한 검색

❖ Repository 확장

✓ JPQLQuery 객체

- JPQL을 작성하고 실행하기 위해서는 Querydsl 라이브러리 내에는 JPQLQuery라는 인터페이스를 활용

◆ SearchBoardRepository 인터페이스에 메서드 선언

```
Board search1();
```

JPQL를 이용한 검색

❖ Repository 확장

✓ JPQLQuery 객체

- JPQL을 작성하고 실행하기 위해서는 Querydsl 라이브러리 내에는 JPQLQuery라는 인터페이스를 활용

◆ SearchBoardRepositoryImpl 클래스에 메서드 구현

@Log4j2

```
public class SearchBoardRepositoryImpl extends QuerydslRepositorySupport
implements SearchBoardRepository {
    public SearchBoardRepositoryImpl() {
        super(Board.class);
    }
```

@Override

```
public Board search1() {
    log.info("search1.....");
    QBoard board = QBoard.board;
    JPQLQuery<Board> jpqlQuery = from(board);
    jpqlQuery.select(board).where(board.bno.eq(1L));
    log.info("-----");
    log.info(jpqlQuery);
    log.info("-----");
    List<Board> result = jpqlQuery.fetch();
    return null;
}
```

```
}
```

JPQL를 이용한 검색

❖ Repository 확장

✓ JPQLQuery 객체

- JPQL을 작성하고 실행하기 위해서는 Querydsl 라이브러리 내에는 JPQLQuery라는 인터페이스를 활용

◆ BoardRepository 인터페이스의 선언 부분 수정

```
public interface BoardRepository extends JpaRepository<Board, Long>,  
SearchBoardRepository {
```

◆ 테스트 메서드를 생성해서 테스트 수행

```
@Test  
public void testSearch1() {  
    boardRepository.search1();  
}
```

JPQL를 이용한 검색

❖ Repository 확장

✓ JPQLQuery 객체

- JPQL을 작성하고 실행하기 위해서는 Querydsl 라이브러리 내에는 JPQLQuery라는 인터페이스를 활용

◆ 실행 결과

```
2022-01-16 10:55:31.361 INFO 83313 --- [ Test worker]
k.c.a.b.r.s.SearchBoardRepositoryImpl : select board
from Board board
where board.bno = ?1
2022-01-16 10:55:31.361 INFO 83313 --- [ Test worker]
k.c.a.b.r.s.SearchBoardRepositoryImpl : -----
Hibernate:
select
  board0_.bno as bno1_0_,
  board0_.moddate as moddate2_0_,
  board0_.regdate as regdate3_0_,
  board0_.content as content4_0_,
  board0_.title as title5_0_,
  board0_.writer_email as writer_e6_0_
from
  board board0_
where
  board0_.bno=?
```

JPQL를 이용한 검색

❖ Repository 확장

✓ JPQLQuery 의 left Join/on()

- JPQLQuery로 다른 Entity와 조인을 처리하기 위해서는 join() 혹은 leftjoin(), rightjoin() 등을 이용하고 필요한 경우 on()을 이용해서 조인에 필요한 부분을 완성할 수 있음
- Board는 Reply와 left (outer) join을 이용해야 하는 상황이면 다음과 같이 코드를 작성할 수 있음

```
QBoard board = QBoard.board;
```

```
QReply reply = QReply.reply;
```

```
JPQLQuery<Board> jpqlQuery = from(board);  
jpqlQuery.leftJoin(reply).on(reply.board.eq(board));
```

◆ 실행 시 만들어지는 JPQL

```
select board from Board board
```

```
left join Reply reply with reply.board = board where board.bno = ?1
```

JPQL를 이용한 검색

❖ Repository 확장

✓ JPQLQuery 의 left Join/on()

- Board는 Reply와 left (outer) join을 이용해야 하는 상황이면 다음과 같이 코드를 작성할 수 있음

◆ 실행 시 만들어지는 SQL

Hibernate: select

```
board0_.bno as bno1_0_, board0_.moddate as moddate2_0_,  
board0_.regdate as regdate3_0_, board0_.content as content4_0_,  
board0_.title as title5_0_, board0_.writer_email as writer_e6_0_
```

from

board board0_

left outer join reply replyl_

on (

replyl_.board_bno=board0_. bno

) where

board0_.bno=?

JPQL를 이용한 검색

❖ Repository 확장

✓ Board 객체

- SearchBoardRepositoryImpl 클래스의 메서드 수정

@Override

public Board search1() {

log.info("search1.....");

QBoard board = QBoard.board;

QReply reply = QReply.reply;

QMember member = QMember.member;

JPQLQuery<Board> jpqlQuery = from(board);

jpqlQuery.leftJoin(member).on(board.writer.eq(member));

jpqlQuery.leftJoin(reply).on(reply.board.eq(board));

jpqlQuery.select(board, member.email, reply.count()).groupBy(board);

log.info("-----");

log.info(jpqlQuery);

log.info("----- ");

List<Board> result = jpqlQuery.fetch();

log.info(result);

return null;

}

JPQL를 이용한 검색

- ❖ Repository 확장
 - ✓ Board 객체
 - 테스트 메서드 재실행



JPQL를 이용한 검색

❖ Repository 확장

✓ Tuple 객체

- JPQLQuery의 leftjoin(), join()을 이용해서 Board, Member, Reply를 처리할 수 있고 groupBy() 등을 이용해서 집합 함수를 처리하는 것도 가능



JPQL를 이용한 검색

❖ Repository 확장

✓ Tuple 객체

- SearchBoardRepositoryImpl 클래스의 메서드 수정

@Override

```
public Board search1() {  
    log.info("search1.....");
```

```
    QBoard board = QBoard.board;  
    QReply reply = QReply.reply;  
    QMember member = QMember.member;
```

```
    JPQLQuery<Board> jpqlQuery = from(board);  
    jpqlQuery.leftJoin(member).on(board.writer.eq(member));  
    jpqlQuery.leftJoin(reply).on(reply.board.eq(board));
```

JPQL를 이용한 검색

❖ Repository 확장

✓ Tuple 객체

● SearchBoardRepositoryImpl 클래스의 메서드 수정

```
/*
    jpqlQuery.select(board, member.email, reply.count()).groupBy(board);
    log.info("-----");
    log.info(jpqlQuery);
    log.info("----- ");
    List<Board> result = jpqlQuery.fetch();
*/

//위의 코드를 Tuple로 변경
JPQLQuery<Tuple> tuple = jpqlQuery.select(board, member.email,
reply.count());
tuple.groupBy(board);

log.info("-----");
log.info(tuple);
log.info("-----");

List<Tuple> result = tuple.fetch();
log.info(result);
return null;
}
```

JPQL를 이용한 검색

- ❖ Repository 확장
 - ✓ Tuple 객체
 - 테스트 메서드 재실행



JPQL를 이용한 검색

- ❖ Repository 확장
 - ✓ Tuple 객체
 - 결과

JPQL

```
select board, member1.email, count(reply)
from Board board
     left join Member member1 with board.writer = member1
     left join Reply reply with reply.board = board
group by board
```

JPQL를 이용한 검색

- ❖ Repository 확장
 - ✓ Tuple 객체
 - 결과

실행되는 SQL

```
select
    board0_.bno as col_0_0_,
    member1_.email as col_1_0_,
    count(reply2_.rno) as col_2_0_,
    board0_.bno as bno1_0_,
    board0_.moddate as moddate2_0_,
    board0_.regdate as regdate3_0_,
    board0_.content as content4_0_,
    board0_.title as title5_0_,
    board0_.writer_email as writer_e6_0_
from
    board board0_
left outer join
    tbl_member member1_
        on (
            board0_.writer_email=member1_.email
        )
left outer join
    reply reply2_
        on (
            reply2_.board_bno=board0_.bno
        )
group by
    board0_.bno
```

JPQL를 이용한 검색

- ❖ Repository 확장
 - ✓ Tuple 객체
 - 결과

`tuple.fetch()`
결과

```
[[Board(bno=2, title=Title...3 test, content=Content....3),  
 user2@aaa.com, 6], [Board(bno=4, title=Title...4,  
 content=Content....4), user4@aaa.com, 1], [Board(bno=5,  
 title=Title...5, content=Content....5), user5@aaa.com,  
 4], [Board(bno=6, title=Title...6, content=Content....6),  
 user6@aaa.com, 1], [Board(bno=7, title=Title...7,  
 content=Content....7), user7@aaa.com, 5], [Board(bno=8,  
 title=Title...8, content=Content....8), user8@aaa.com, 4],  
 ...  
 [Board(bno=102, title=한글테스트, content=한글테스트), user30@  
 aaa.com, 0]]
```

JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

- ✓ SearchBoardRepository 인터페이스에 Pageable 을 전송하고 Page<Object[]> 로 리턴받는 메서드를 선언

```
Page<Object[]> searchPage(String type, String keyword, Pageable pageable);
```



JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

- ✓ SearchBoardRepositoryImpl 클래스에 Pageable 을 전송하고 Page<Object[]> 로 리턴받는 메서드를 구현

@Override

```
public Page<Object[]> searchPage(String type, String keyword, Pageable pageable) {  
    log.info("searchPage.....");
```

```
    QBoard board = QBoard.board;
```

```
    QReply reply = QReply.reply;
```

```
    QMember member = QMember.member;
```

```
    JPQLQuery<Board> jpqlQuery = from(board);
```

```
    jpqlQuery.leftJoin(member).on(board.writer.eq(member));
```

```
    jpqlQuery.leftJoin(reply).on(reply.board.eq(board));
```

```
    //SELECT b, w, count(r) FROM Board b
```

```
    //LEFT JOIN b.writer w LEFT JOIN Reply r ON r.board = b
```

```
    JPQLQuery<Tuple> tuple = jpqlQuery.select(board, member, reply.count());
```

JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

- ✓ SearchBoardRepositoryImpl 클래스에 Pageable 을 전송하고 Page<Object[]> 로 리턴받는 메서드를 구현

```
BooleanBuilder booleanBuilder = new BooleanBuilder();  
BooleanExpression expression = board.bno.gt(0L);  
booleanBuilder.and(expression);
```

JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

- ✓ SearchBoardRepositoryImpl 클래스에 Pageable 을 전송하고 Page<Object[]> 로 리턴받는 메서드를 구현

```
if(type != null){
    String[] typeArr = type.split("");
    //검색 조건을 작성하기
    BooleanBuilder conditionBuilder = new BooleanBuilder();
    for (String t:typeArr) {
        switch (t){
            case "t":
                conditionBuilder.or(board.title.contains(keyword));
                break;
            case "w":
                conditionBuilder.or(member.email.contains(keyword));
                break;
            case "c":
                conditionBuilder.or(board.content.contains(keyword));
                break;
        }
    }
    booleanBuilder.and(conditionBuilder);
}

tuple.where(booleanBuilder);
```

JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

- ✓ SearchBoardRepositoryImpl 클래스에 Pageable 을 전송하고 Page<Object[]> 로 리턴받는 메서드를 구현

```
//order by
Sort sort = pageable.getSort();

//tuple.orderBy(board.bno.desc());

sort.stream().forEach(order -> {
    Order direction = order.isAscending()? Order.ASC: Order.DESC;
    String prop = order.getProperty();

    PathBuilder orderByExpression = new PathBuilder(Board.class, "board");
    tuple.orderBy(new OrderSpecifier(direction, orderByExpression.get(prop)));
});

tuple.groupBy(board);
```

JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

- ✓ SearchBoardRepositoryImpl 클래스에 Pageable 을 전송하고 Page<Object[]> 로 리턴받는 메서드를 구현

```
//page 처리  
tuple.offset(pageable.getOffset());  
tuple.limit(pageable.getPageSize());
```

```
List<Tuple> result = tuple.fetch();
```

```
log.info(result);
```

```
//튜플의 개수 구하기  
long count = tuple.fetchCount();
```

```
log.info("COUNT: " + count);
```

```
return new PageImpl<Object[]>(  
    result.stream().map(t -> t.toArray()).collect(Collectors.toList()),  
    pageable,  
    count);
```

```
}
```

JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

✓ 테스트 코드 작성 후 테스트

```
@Test
public void testSearchPage() {
    Pageable pageable = PageRequest.of(0,10,
        Sort.by("bno").descending()
            .and(Sort.by("title").ascending()));
    Page<Object[]> result = boardRepository.searchPage("t", "1", pageable);
}
```

JPQL를 이용한 검색

❖ JPQLQuery 로 Page<Object []> 처리

- ✓ BoardServiceImpl 클래스의 getList 메서드 수정

```
@Override
public PageResultDTO<BoardDTO, Object[]> getList(PageRequestDTO
pageRequestDTO) {

    log.info(pageRequestDTO);

    Function<Object[], BoardDTO> fn = (en ->
entityToDTO((Board)en[0],(Member)en[1],(Long)en[2]));

    //      Page<Object[]> result = repository.getBoardWithReplyCount(
    //          pageRequestDTO.getPageable(Sort.by("bno").descending()) );

    Page<Object[]> result = repository.searchPage(
        pageRequestDTO.getType(),
        pageRequestDTO.getKeyword(),
        pageRequestDTO.getPageable(Sort.by("bno").descending()) );

    return new PageResultDTO<>(result, fn);
}
```

@RestController & JSON

❖ 댓글 처리

- ✓ REST 방식이라고 불리는 다양한 방식(method)의 호출을 이용해서 댓글을 처리
- ✓ 모든 댓글은 게시물의 조회 화면에서 처리되도록 하고 Ajax를 이용해서 컨트롤러와 JSON 포맷으로 데이터를 교환하는 방식으로 작성
- ✓ 브라우저에서는 '/replies/'라는 경로를 이용해서 원하는 작업을 다음과 같이 호출

방식	호출 대상	파라미터	작업	반환되는 데이터
GET	/replies/board/ {bno} (게시물 번호)	게시물 번호	해당 게시물의 댓글 조회	JSON 배열
POST	/replies/	JSON으로 구성된 댓글 데이터	댓글 추가	추가된 댓글 번호
DELETE	/replies/{no}	댓글 번호	댓글 삭제	삭제 결과 문자열
PUT	/replies/{no}	댓글 번호 + 수정할 내용	댓글 수정	수정 결과 문자열

@RestController & JSON

❖ 댓글 작업

✓ Reply Entity 수정

```
@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString(exclude = "board")
public class Reply extends BaseEntity{
    @Id
    @GeneratedValue(strategy = GenerationType. AUTO)
    private Long rno;
    private String text;
    private String replyer;

    @ManyToOne(fetch = FetchType.LAZY)
    private Board board;
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ ReplyRepository 인터페이스에 메서드 추가

//게시물로 댓글 목록 가져오기

```
List<Reply> getRepliesByBoardOrderByRno(Board board);
```

- ✓ 테스트 메서드에서 테스트 수행

```
@Test
```

```
public void testListByBoard() {
```

```
    List<Reply> replyList =
```

```
    replyRepository.getRepliesByBoardOrderByRno( Board.builder().bno(95L).build());
```

```
    replyList.forEach(reply -> System.out.println(reply));
```

```
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ 화면에서 댓글을 처리하기 위한 ReplyDTO 클래스를 dto 패키지에 생성하고 작성

```
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Builder
@AllArgsConstructor
@NoArgsConstructor
@Data
public class ReplyDTO {
    private Long rno;
    private String text;
    private String replyer;
    private Long bno; //게시글 번호
    private LocalDateTime regDate, modDate;
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ 댓글을 처리하기 위한 메서드를 소유한 ReplyService 인터페이스
 - Reply를 ReplyDTO를 변환하는 entityToDTO()
 - ReplyDTO를 Reply로 변환하는 dtoToEntity()
 - 댓글을 등록하는 기능(register)
 - 특정 게시물의 댓글 리스트를 가져오는 기능(getList)
 - 댓글을 수정(modify)하고 삭제(remove)하는 기능

@RestController & JSON

❖ 댓글 작업

- ✓ 댓글을 처리하기 위한 메서드를 소유한 ReplyService 인터페이스를 service 패키지에 생성하고 메서드 선언

```
public interface ReplyService {  
    Long register(ReplyDTO replyDTO); //댓글의 등록  
    List<ReplyDTO> getList(Long bno); //특정 게시물의 댓글 목록  
    void modify(ReplyDTO replyDTO); //댓글 수정  
    void remove(Long rno); //댓글 삭제
```

@RestController & JSON

❖ 댓글 작업

- ✓ 댓글을 처리하기 위한 메서드를 소유한 ReplyService 인터페이스를 service 패키지에 생성하고 메서드 선언

```
//ReplyDTO를 Reply객체로 변환 Board객체의 처리가 수반됨
default Reply dtoToEntity(ReplyDTO replyDTO){
    Board board = Board.builder().bno(replyDTO.getBno()).build();
    Reply reply = Reply.builder().rno(replyDTO.getRno())
        .text(replyDTO.getText())
        .replyer(replyDTO.getReplyer())
        .board(board)
        .build();
    return reply;
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ 댓글을 처리하기 위한 메서드를 소유한 ReplyService 인터페이스를 service 패키지에 생성하고 메서드 선언

```
//Reply객체를 ReplyDTO로 변환 Board 객체가 필요하지 않으므로 게시물 번호만
default ReplyDTO entityToDTO(Reply reply) {
    ReplyDTO dto = ReplyDTO.builder()
        .rno(reply.getRno())
        .text(reply.getText())
        .replyer(reply.getReplyer())
        .regDate(reply.getRegDate())
        .modDate(reply.getModDate())
        .build();
    return dto;
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ 댓글을 처리하기 위한 메서드를 구현한 ReplyServiceImpl 클래스를 service 패키지에 생성하고 메서드 구현

```
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.stream.Collectors;
```

```
@Service
```

```
@RequiredArgsConstructor
```

```
public class ReplyServiceImpl implements ReplyService {
    private final ReplyRepository replyRepository;
```

```
@Override
```

```
public Long register(ReplyDTO replyDTO) {
    Reply reply = dtoToEntity(replyDTO);
    replyRepository.save(reply);
    return reply.getRno();
}
```


@RestController & JSON

❖ 댓글 작업

- ✓ 댓글을 처리하기 위한 메서드를 구현한 ReplyServiceImpl 클래스를 service 패키지에 생성하고 메서드 구현

```
@Override
public List<ReplyDTO> getList(Long bno) {
    List<Reply> result =
replyRepository.getRepliesByBoardOrderByRno(Board.builder().bno(bno).build());
    result.sort(new Comparator<Reply>() {
        @Override
        public int compare(Reply o1, Reply o2) {
            return o2.getModDate().compareTo(o1.getModDate());
        }
    });

    return result.stream().map(reply -> entityToDTO(reply)).collect(Collectors.toList());
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ 댓글을 처리하기 위한 메서드를 구현한 ReplyServiceImpl 클래스를 service 패키지에 생성하고 메서드 구현

```
@Override
public void modify(ReplyDTO replyDTO) {
    Reply reply = dtoToEntity(replyDTO);
    replyRepository.save(reply);
}

@Override
public void remove(Long rno) {
    replyRepository.deleteById(rno);
}
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ ReplyService 의 메서드를 테스트하기 위한 ServiceTest 클래스에 코드를 추가

```
@Autowired
```

```
private ReplyService replyService;
```

```
@Test
```

```
public void testGetList() {
```

```
    Long bno = 95L; //데이터베이스에 존재하는 번호
```

```
    List<ReplyDTO> replyDTOList = replyService.getList(bno);
```

```
    replyDTOList.forEach(replyDTO -> System.out.println(replyDTO));
```

```
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ 댓글 요청 처리를 수행할 controller.ReplyController를 생성하고 댓글을 가져오는 요청을 처리하는 메서드를 생성

```
@RestController
@RequestMapping("/replies/")
@Log4j2
@RequiredArgsConstructor
public class ReplyController {
    private final ReplyService replyService;//자동주입을 위해 final
    @GetMapping(value = "/board/{bno}", produces =
    MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<List<ReplyDTO>> getListByBoard(@PathVariable("bno") Long
    bno ){
        log.info("bno: " + bno);
        return new ResponseEntity<>(replyService.getList(bno), HttpStatus.OK);
    }
}
```

@RestController & JSON

❖ 댓글 작업

- ✓ read.html 파일에 댓글 출력을 위한 영역을 생성

```
<div >
  <div class="mt-4">
    <h5> <span class="badge badge-secondary replyCount" id="replyCount"> 댓글
[[${dto.replyCount}]]</span> </h5>
  </div>
  <div class="list-group replyList" id="replyList">
  </div>
</div>
```

@RestController & JSON

❖ 댓글 작업

- ✓ read.html 파일에 댓글 출력을 위한 스크립트 코드 추가

```
<script th:inline="javascript">
  window.addEventListener("load", (e) => {
    let bno = [${dto.bno}];

    document.getElementById("replyCount").addEventListener("click", (e) => {
      loadJSONData();
    });

    //댓글이 추가될 영역
    let listGroup = document.getElementById("replyList");
    //날짜 처리를 위한 함수
    let formatTime = (str) => {
      let date = new Date(str);
      return date.getFullYear() + '/' + (date.getMonth() + 1) + '/' + date.getDate() + '
' + date.getHours() + ':' + date.getMinutes();
    }
  });
</script>
```

@RestController & JSON

❖ 댓글 작업

- ✓ read.html 파일에 댓글 출력을 위한 스크립트 코드 추가

//특정한 게시글의 댓글을 처리하는 함수

```
let loadJSONData = () => {
```

```
  fetch('/replies/board/' + bno)
```

```
    .then((response) => response.json())
```

```
    .then((arr) => {
```

```
      //console.log(arr);
```

```
      let str = "";
```

```
      document.getElementById('replyCount').innerHTML = "댓글 수 " +
```

```
arr.length;
```

@RestController & JSON

❖ 댓글 작업

- ✓ read.html 파일에 댓글 출력을 위한 스크립트 코드 추가

```
for(const reply of arr){
    console.log(reply);
    str += '<div class="card-body" data-rno="' + reply.rno + '"><b>' +
reply.rno + '</b>';
    str += '<h5 class="card-title">' + reply.text + '</h5>';
    str += '<h6 class="card-subtitle mb-2 text-muted">' + reply.replier +
'</h6>';

    str += '<p class="card-text">' + formatTime(reply.regDate) + '</p>';
    str += '</div>';
}
listGroup.innerHTML = str;
});
</script>
```


@RestController & JSON

❖ 댓글 작업

✓ 댓글 추가

- read.html 파일에 댓글 삽입, 수정 및 삭제에 이용한 모달창 영역을 생성

```
<div class="modal" tabindex="-1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Modal title</h5>
        <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
          <span aria-hidden="true">&times;</span> </button>
      </div>
      <div class="modal-body">
        <div class="form-group">
          <input class="form-control" type="text" name="replyText"
placeholder="댓글 작성...">
        </div>
        <div class="form-group">
          <input class="form-control" type="text" name="replyer"
placeholder="작성자..." >
          <input type="hidden" name="rno" >
        </div>
      </div>
    </div>
  </div>
</div>
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 추가

- read.html 파일에 댓글 삽입, 수정 및 삭제에 이용한 모달창 영역을 생성

```
<div class="modal-footer">
  <button type="button" class="btn btn-danger
replyRemove">삭제</button>
  <button type="button" class="btn btn-warning
replyModify">수정</button>
  <button type="button" class="btn btn-primary
replySave">추가</button>
  <button type="button" class="btn btn-outline-secondary replyClose"
data-dismiss="modal">닫기</button>
</div>
</div>
</div>
</div>
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 추가

- read.html 파일에 댓글 삽입 요청을 생성

```
<div >
  <div class="mt-4">
    <h5> <span class="badge badge-info addReply">댓글 작성</span>
  </h5>
    <h5> <span class="badge badge-secondary replyCount"> 댓글
[[${dto.replyCount}]]</span> </h5>
  </div>
  <div class="list-group replyList">
  </div>
</div>
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 추가

- read.html 파일에 댓글 삽입 요청을 위한 스크립트를 추가

```
//모달창
let modal = $('#modal');
//닫기 버튼을 눌렀을 때 처리
$(".replyClose").on("click", function(){
    modal.modal('hide');
});

$(".addReply").click(function () {
    modal.modal('show');
    //댓글 입력하는 부분 초기화 시키기
    $('input[name="replyText"]').val("");
    $('input[name="replyer"]').val("");
    $(".modal-footer .btn").hide(); //모달 내의 모든 버튼을 안 보이도록
    $(".replySave, .replyClose").show(); //필요한 버튼들만 보이도록
});
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 추가

- read.html 파일에 댓글 삽입 요청을 위한 스크립트를 추가

```
$(".replySave").click(function() {  
    let reply = { bno: bno,  
        text: $('input[name="replyText"]').val(),  
        replyer: $('input[name="replyer"]').val()  
    }  
    console.log(reply);  
    $.ajax({  
        url: '/replies/',  
        method: 'post',  
        data: JSON.stringify(reply),  
        contentType: 'application/json; charset=utf-8', dataType: 'json',  
        success: function(data){  
            console.log(data);  
            let newRno = parseInt(data);  
            alert(newRno + "번 댓글이 등록되었습니다.");  
            modal.modal('hide');  
            loadJSONData();  
        }  
    });  
});
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 추가

- ReplyController 클래스에 댓글 추가 요청을 처리하기 위한 메서드를 추가

```
@PostMapping("")
public ResponseEntity<Long> register(@RequestBody ReplyDTO replyDTO){
    log.info(replyDTO);
    Long rno = replyService.register(replyDTO);
    return new ResponseEntity<>(rno, HttpStatus.OK);
}
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 클릭 – 대화상자로 댓글 출력

- read.html 파일에 댓글을 클릭했을 때 처리를 위한 스크립트 추가

//댓글을 클릭했을 때 처리

```
$('.replyList').on("click", ".card-body", function(){  
    let rno = $(this).data("rno");  
    $("input[name='replyText']").val( $(this).find('.card-title').html());  
    $("input[name='replyer']").val( $(this).find('.card-subtitle').html());  
    $("input[name='rno']").val(rno);  
  
    $(".modal-footer .btn").hide();  
    $(".replyRemove, .replyModify, .replyClose").show();  
  
    modal.modal('show');  
});
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 삭제

- read.html 파일에 댓글 삭제 요청을 위한 스크립트를 추가

```
//삭제 버튼을 눌렀을 때 처리
$(".replyRemove").on("click", function(){
    let rno = $("input[name='rno']").val(); //모달창에 보이는 댓글 번호로
    hidden처리되어 있음
    $.ajax({
        url: '/replies/' + rno,
        method: 'delete',
        success: function(result){
            console.log("result: " + result);
            if(result === 'success'){
                alert("댓글이 삭제되었습니다.");
                modal.modal('hide');
                loadJSONData();
            }
        }
    });
});
```


@RestController & JSON

❖ 댓글 작업

✓ 댓글 삭제

- ReplyController 클래스에 댓글 삭제 요청을 처리하기 위한 메서드를 추가

```
@DeleteMapping("/{rno}")  
public ResponseEntity<String> remove(@PathVariable("rno") Long rno) {  
    log.info("RNO:" + rno );  
    replyService.remove(rno);  
    return new ResponseEntity("success", HttpStatus.OK);  
}
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 수정

● 처리

- ◆ 댓글의 수정은 PUT 방식으로 하고 댓글의 번호와 게시물의 번호, 내용, 작성자를 같이 전달해야 함
- ◆ 수정 작업은 등록과 유사하게 모든 내용을 하나의 객체로 구성해서 이를 JSON 형태로 전달하고 서버에서는 JSON 데이터를 ReplyDTO로 변환해서 처리

@RestController & JSON

❖ 댓글 작업

✓ 댓글 수정

- read.html 파일에 댓글 수정 버튼을 클릭했을 때 수행될 스크립트 코드를 작성

```
$(".replyModify").click(function() {  
    let rno = $("input[name='rno']").val();  
    let reply = {  
        rno: rno,  
        bno: bno,  
        text: $('input[name="replyText"]').val(),  
        replyer: $('input[name="replyer"]').val()  
    }  
    console.log(reply);
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 수정

- read.html 파일에 댓글 수정 버튼을 클릭했을 때 수행될 스크립트 코드를 작성

```
$.ajax({  
  url: '/replies/' + rno,  
  method: 'put',  
  data: JSON.stringify(reply),  
  contentType: 'application/json; charset=utf-8',  
  success: function(result){  
    console.log("RESULT: " + result);  
    if(result === 'success'){  
      alert("댓글이 수정되었습니다");  
      modal.modal('hide');  
      loadJSONData();  
    }  
  }  
});
```

@RestController & JSON

❖ 댓글 작업

✓ 댓글 수정

- ReplyController 클래스에 댓글 수정 요청을 처리하기 위한 메서드를 추가

```
@PutMapping("/{rno}")  
public ResponseEntity<String> modify(@RequestBody ReplyDTO replyDTO) {  
    log.info(replyDTO);  
    replyService.modify(replyDTO);  
    return new ResponseEntity<>("success", HttpStatus.OK);  
}
```

다대일 양방향 매핑

❖ 양방향 매핑

- ✓ Entity의 관계를 양방향으로 매핑할 때 테이블의 양방향 과 객체의 양방향 메커니즘이 완전히 다름
- ✓ 관계형 데이터베이스는 처음부터 외래 키(Foreign Key) 하나를 이용하여 양방향 조회가 가능
- ✓ 객체는 테이블과 달리 객체 참조 변수를 통해 다른 객체를 참조하기 때문에 객체를 양방향으로 매핑하기 위해서는 매핑에 참여하는 두 객체 상호간에 참조 변수를 가지도록 해야 함

Employee.java	Department.java
<pre>@Entity public class Employee { @Id private Long id; private Department dept; }</pre>	<pre>@Entity public class Department { @Id private Long deptId; private Employee emp; }</pre>

다대일 양방향 매핑

❖ 양방향 매핑

- ✓ 데이터 모델링 관점에서 테이블을 양방향으로 연결하기 위한 연결점은 하나(외래키)인데 객체지향에서 객체를 양방향으로 연결하기 위한 연결점은 두 개(객체 참조 변수)
- ✓ Entity의 관계를 양방향으로 매핑하면 당연히 데이터를 사용할 때 편리한 점이 많지만 양방향 매핑 설정 과정은 생각보다는 복잡하고 성능 관련 이슈도 존재하기 때문에 반드시 정확한 이해를 바탕으로 사용해야 함



다대일 양방향 매핑

❖ 양방향 매핑

✓ 양방향 매핑을 위한 샘플 데이터 생성

-- 테이블 삭제

```
DROP TABLE s_dept;
```

```
DROP TABLE s_emp;
```

-- 부서 테이블

```
CREATE TABLE s_dept(  
    dept_id INT NOT NULL,  
    name VARCHAR(25) NOT NULL, -- 부서 이름  
    CONSTRAINT s_dept_id_pk PRIMARY KEY (dept_id)  
);
```

```
CREATE TABLE s_emp(  
    id      INT NOT NULL,  
    name VARCHAR(25) NOT NULL,  
    dept_id INT,  
    CONSTRAINT s_emp_id_pk PRIMARY KEY (id)  
);
```

```
ALTER TABLE s_emp ADD CONSTRAINT s_emp_dept_id_fk  
    FOREIGN KEY (dept_id) REFERENCES s_dept (dept_id);
```


다대일 양방향 매핑

❖ 양방향 매핑

✓ 양방향 매핑을 위한 샘플 데이터 생성

```
INSERT INTO s_dept VALUES(1, '개발부');  
INSERT INTO s_emp VALUES(1, '둘리', 1);  
INSERT INTO s_emp VALUES(2, '도우너', 1);
```

```
commit;
```



다대일 양방향 매핑

❖ 양방향 매핑

✓ Entity 생성 - Employee

```
@Data
@Entity
@Table(name = "s_emp")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 25, nullable = false)
    private String name;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "DEPT_ID")
    private Department dept;
```

다대일 양방향 매핑

❖ 양방향 매핑

✓ Entity 생성 - Employee

```
public void setDept(Department department) {  
    this.dept = department;  
  
    // Department 엔티티의 컬렉션에도 Employee 참조를 설정한다.  
    department.getEmployeeList().add(this);  
}  
  
// 부서 정보를 null로 설정하여 직원을 대기 상태로 전환시킨다.  
public void standby() {  
    this.dept = null;  
}  
}
```

다대일 양방향 매핑

❖ 양방향 매핑

✓ Entity 생성 - Department

```
@EqualsAndHashCode(exclude = "employeeList")
@Entity
@Table(name = "s_dept")
@Getter
@ToString(exclude = "employeeList")
@Setter
public class Department {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "DEPT_ID")
    private Long deptId;

    @Column(length = 25, nullable = false)
    private String name;

    @OneToMany(mappedBy = "dept", fetch = FetchType.LAZY)
    private List<Employee> employeeList = new ArrayList<Employee>();
}
```

다대일 양방향 매핑

❖ 양방향 매핑

- ✓ 다중성은 상대적인 개념이기 때문에 직원 관점에서 부서를 봤을 때 다대일 관계라면 반대로 부서에서 바라본 직원과의 관계는 일대다가 되는 것이므로 다대일 연관관계 매핑에서 @ManyToOne을 사용했다면 일대다 매핑에서는 컬렉션 변수 위에 @OneToMany 어노테이션을 사용
- ✓ @OneToMany도 여러 속성을 사용할 수 있으며 의미도 @ManyToOne과 동일한데 mappedBy라는 새로운 속성을 사용했는데 mappedBy는 객체를 양방향으로 매핑할 때 연관 관계의 소유자(Owner)와 관련된 속성
- ✓ 양방향 연관 관계에서 연관 관계의 소유자 개념을 이해하는 것이 가장 어렵고 중요한데 mappedBy는 연관 관계의 소유자가 아닌 객체에 설정하는 속성
- ✓ @OneToMany 설정에서 두 번째로 적용한 속성은 fetch인데 @ManyToOne에서는 fetch 속성의 기본 설정이 FetchType.EAGER로 되어 있는데 이는 다대일(ManyToOne) 관계에서 연관 관계에 있는 객체가 하나기 때문에 처음부터 조인을 통해 연관된 데이터를 가져오는 것이 성능상 유리하기 때문이지만 @OneToMany에서는 연관된 객체가 하나가 아닌 컬렉션 형태이기 때문에 fetch 속성의 기본 설정이 FetchType.LAZY로 되어있기 때문에 현재 상태에서는 fetch 속성을 제거해도 실행 결과는 동일

다대일 양방향 매핑

❖ 양방향 매핑

- ✓ Repository 생성 - DepartmentRepository

```
public interface DepartmentRepository extends JpaRepository<Department, Long> {  
}
```

다대일 양방향 매핑

❖ 양방향 매핑

- ✓ Repository 생성 - EmployeeRepository

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
}
```

다대일 양방향 매핑

❖ 양방향 매핑

- ✓ Test 코드 작성 후 실행

```
@Autowired  
EmployeeRepository employeeRepository;
```

```
@Autowired  
DepartmentRepository departmentRepository;
```

```
@Transactional  
@Test  
public void testSelect(){  
    Optional<Department> result = departmentRepository.findById(1L);  
    if(result.isPresent()){  
        Department department = result.get();  
        System.out.println(department);  
        for(Employee employee : department.getEmployeeList()){  
            System.out.println(employee);  
        }  
    }else{  
        System.out.println("데이터 없음");  
    }  
}
```


다대일 양방향 매핑

❖ 양방향 매핑

✓ 일대다 연관 관계 매칭 과 컬렉션

- 일대다 관계를 매핑할 때는 List나 Set 같은 컬렉션을 사용하는데 이 두 컬렉션은 동작 메커니즘이 조금 다름
- 일대다 연관관계 매핑에서 가장 많이 사용하는 List는 순서대로 객체를 저장하며 인덱스가 증가하고 동일한 객체에 대한 중복 저장을 허용하는 반면 Set 은 객체의 순서를 관리하지 않으며 객체의 중복 저장도 허용하지 않기 때문에 정확하게 동작하기 위해서는 반드시 Set에 저장되는 Entity가 equals와 hashCode 메서드를 재정의해야 함

다대일 양방향 매핑

❖ 연관 관계 소유자

- ✓ 연관 관계의 소유자(Owner)는 관계형 데이터베이스와 객체 지향의 구조적 불일치 문제를 해결하기 위해 만들어진 개념
- ✓ 관계형 데이터베이스는 원래부터 외래 키 하나로 양방향 연결이 가능했지만 객체를 양방향으로 연결하기 위해서는 서로에 대한 참조 변수가 양쪽 모두 필요
- ✓ 양방향 연관 관계에서 외래 키는 하나인데 참조 변수는 두 개이므로 두 개의 참조 변수 중 하나를 외래 키와 매핑시켜야 하는 문제가 생김
- ✓ 양방향 연관 관계에서 외래 키와 매핑된 참조 변수를 연관 관계 소유자(Owner)라고 하는데 양방향에서는 연관 관계 소유자만이 외래 키에 대해 관리(등록, 수정, 삭제)할 수 있고, 반대쪽은 오로지 읽기(조회)만 가능

다대일 양방향 매핑

❖ 연관 관계 소유자

✓ 소유자 지정

- 양방향 연관 관계에서는 일반적으로 외래 키를 가진 테이블 쪽 엔티티가 연관 관계의 소유자가 되기 때문에 S_EMP 테이블과 매핑되는 Employee 클래스를 소유자로 결정
- 소유자인 Employee 클래스에는 소유자와 관련된 어떤 설정도 주가지 않았고 오히려 반대쪽인 Department.employeeList 변수에 소유자 와 관련된 mappedBy 설정 추가
- 양방향 매핑에서 소유자를 설정하는 방법은 소유자에 해당하는 엔티티에는 아무런 설정도 하지 않는 대신 소유자가 아닌 반대쪽 엔티티에 mappedBy 속성을 사용하여 자신(Department)은 소유자가 아님을 영속 컨테이너에게 알려줌
- mappedBy 속성 값은 소유자 쪽에 있는 참조 변수를 지정
- 이론적으로는 Department 클래스를 소유자로 지정할 수도 있지만 외래 키를 가지지 않는 엔티티를 소유자로 지정하면 가독성도 떨어지고 관리도 어렵기 때문에 소유자는 외래 키를 소유한 테이블 쪽 엔티티로 선정하는 것이 바람직