# ——— Board
# 1:N 관계 작성(JOIN)

## 1. 연관관계와 관계형 데이터 베이스
- 관계형 데이터 베이스에서
Table 과 Table 사이의 관계를 맺을 수 있음

: 관계의 종류
- 대응수(Cardinality)

## 1:1
- 어느 한쪽의 기본키를 가지고 다른
테이블의 데이터를 조회하면
양쪽 모두 0 개 또는 1 개만 조회되는 경우
(일반적으로 잘 존재하지 않으며, 데부분 테이블을 수직적으로 분할 한 경우 발생)

— 수직적
— 수평적
*빅데이터 처리 : spark
*빅데이터 분석 : domain , Visualization

## 1:N
- 어느 한족의 기본키를 가지고 조회시
한쪽은 여러개가 조회 될 수 있고
다른 한쪽은 1개 또는 0개 조회
(가장 많은경우)

ex)
회원과 게시글
게시글과 댓글

## M:N
- 양쪽의 기본키를 가지고 다른쪽을 조회하면
양쪽 모두 여러개의 데이터가 조회

Ex)
상품 과 고객
회원과 여러개의 영화 리뷰

# 회원과 게시글의 관계

## ( 1:N 관계 )

- 한 명의 회원은 여러개의 게시글을 작성할 수 있다
- 하나의 게시글은 한명의 회원이 작성해야 한다.

### ——— 관계형 데이터 베이스의 1:N 관계의 처리

**1 쪽의 기본키를 N쪽의 외래키로 추가해 주어야 한다.**

### JPA 에서의 1:N 관계처리는 다를 수 있다.

단방향으로도 양방향으로도 설정할 수 있다 .

——〉 JPA 에서는 바라보는 방향에 따라
manyToOne 또는 OneToMany 라고 하기 때문

## ———————— Project 생성

### 1. option
- name : board
- type : gradle
- packaging : war

### 2. dependency
- Spring Boot DevTools
- Lombok
- Spring Web
- Thymeleaf
- Spring Data Jpa
- MySQL

### 3. 프로젝트 생성 후 build. gradle 파일에
- Thymeleaf 시간 처리 관련 라이브러리 의존성 설정

### 4. application. properties 파일에 JPA 와 Thymeleaf 관련 설정을 추가

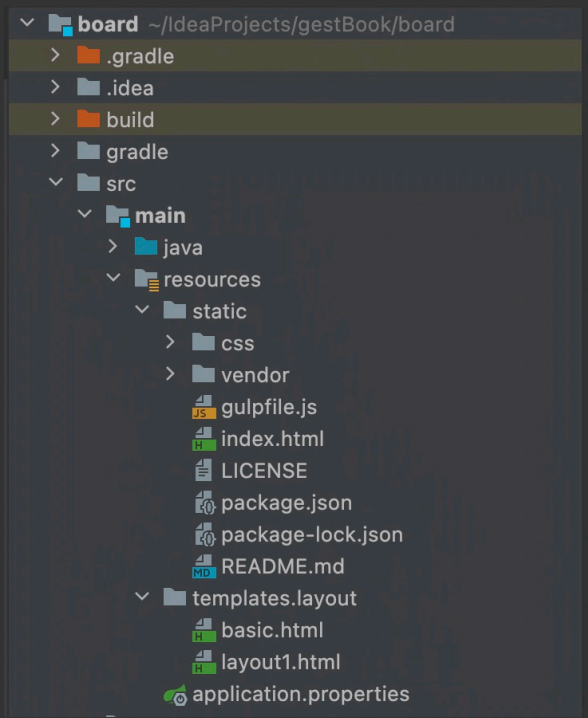
## ———— build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.6.2'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
    id 'war'
}

group = 'com.singsiuk'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

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 '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'
}

test {
    useJUnitPlatform()
}
```

## ———— application.properties

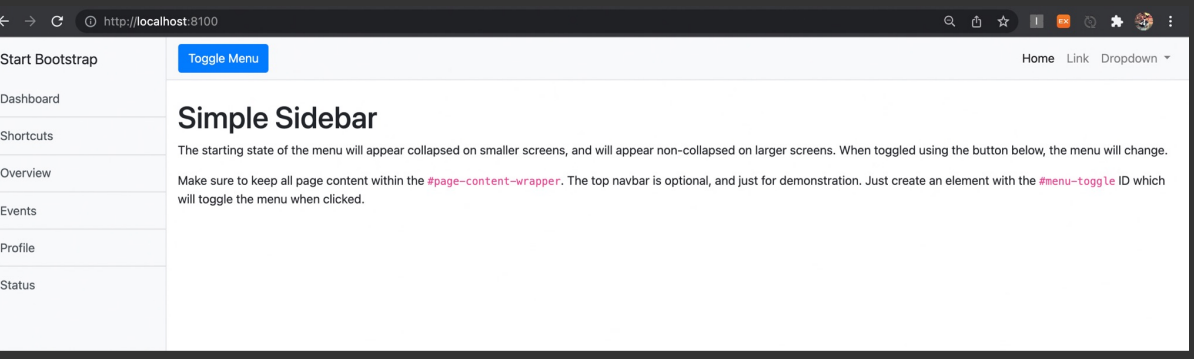
```
server.port=8100
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/SpringTest
spring.datasource.username=singsiuk
spring.datasource.password=ssw0304

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.thymeleaf.cache=false
```

## ———— layout, static 파일 붙여 넣기



## ———————— Application 클래스에 Jpa 감사를 위한 Annotation 추가

```java
package com.singsiuk.board;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication
// 데이터 생성날짜
// 변경된 날짜를 자동적으로 확인할 수 있다
@EnableJpaAuditing
public class BoardApplication {

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




## ——— Date 생성 날짜와 Update 날짜를 생성하는 클래스 생성

```java
package com.singsiuk.board.entity;

import lombok.Getter;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.Column;
import javax.persistence.EntityListeners;
import javax.persistence.MappedSuperclass;
import java.time.LocalDateTime;
@MappedSuperclass
@EntityListeners(value={AuditingEntityListener.class})
@Getter
public class BaseEntity {
    @CreatedDate
    @Column(name="regdate", updatable = false)
    private LocalDateTime regDate;

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



}
```

## ——— Entity 생성

```java
package com.singsiuk.board.entity;

import lombok.*;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

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

## ———————— 게시물 정보를 위한 Entity 클래스를 생성
## :entity / board

```
package com.singsiuk.board.entity;

import lombok.*;
import net.bytebuddy.dynamic.loading.InjectionClassLoader;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

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

## ———————— 댓글 정보를 위한 Entity 클래스를 생성
## :entity / Reply

```
package com.singsiuk.board.entity;

import lombok.*;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

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

# ---- 연관관계 Annotation

## --1 종류

@OneToMany
@ManyToOne

@OneToOne
@ManyToMany

## -- 2 @ManyToOne

: 관계형 데이터베이스 개념으로 보면
외래키를 소유해야 하는 Entity에 Annotation 과 함께 속성을 설정

## -- 3 @OneToMany

: 외래키를 제공해야하는 Entity에 만드는 것으로
이 경우는 속성이 List 가 되어야 한다.

# ----- 기존 Entity 에 관계 설정

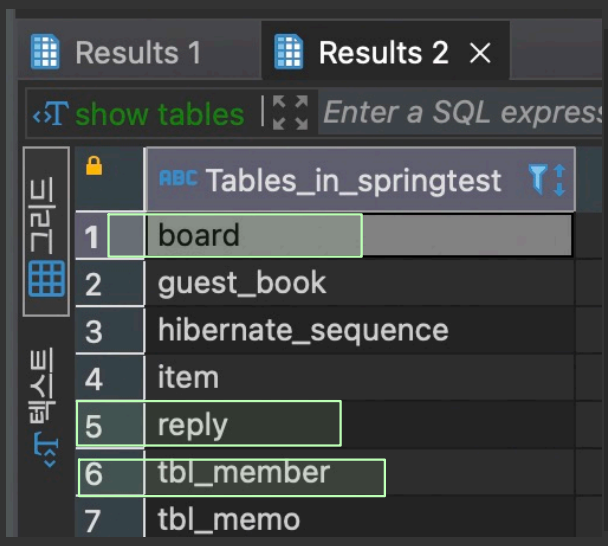
### ---------- Board Entity 에 추가

```
package com.singsiuk.board.entity;

import lombok.*;
import net.bytebuddy.dynamic.loading.InjectionClassLoader;

import javax.persistence.*;

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

     @ManyToOne
     private Member writer;
}
```

### ---------- Reply/Entity에 추가

```
package com.singsiuk.board.entity;

import lombok.*;
import javax.persistence.*;

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

   @ManyToOne
      private Board board;
}
```

# ——————————실행 한 후 수행되는 SQL 과 Table 확인




```
        email varchar(255) not null,
        moddate datetime(6),
        regdate datetime(6),
        name varchar(255),
        password varchar(255),
        primary key (email)
    ) engine=InnoDB
Hibernate:

    alter table board
       add constraint FKsv8b40a4mo0vwhuhr8ryedhoa
       foreign key (writer_email)
       references tbl_member (email)
Hibernate:

    alter table reply
       add constraint FKjdta3s6n8un0bv3ntufqfs6qa
       foreign key (board_bon)
       references board (bon)
```

## ———————————— 3개의 Entity 클래스에 대한 Repository 생성

### 1 ———— BoardRepository

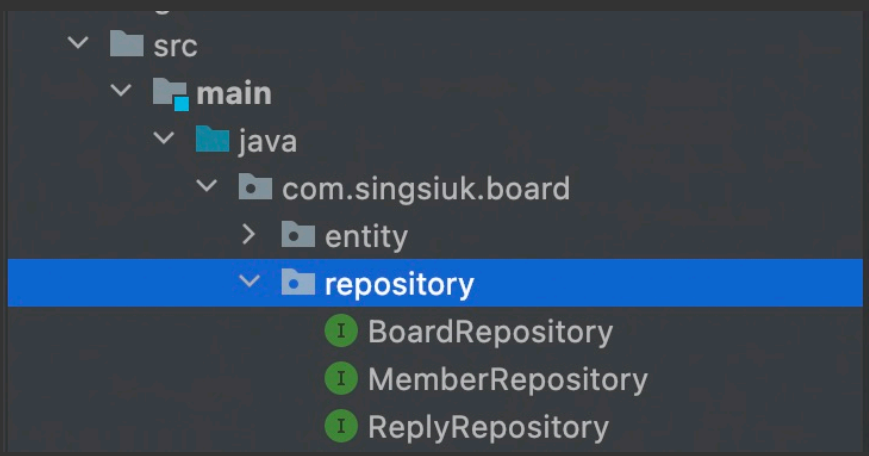
```java
package com.singsiuk.board.repository;

import com.singsiuk.board.entity.Board;
import org.springframework.data.jpa.repository.JpaRepository;

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

### 2 ———— MemberRepository

```java
package com.singsiuk.board.repository;

import com.singsiuk.board.entity.Member;
import org.springframework.data.jpa.repository.JpaRepository;

public interface MemberRepository extends JpaRepository<Member,String> {
}
```

### 3 ———— ReplyRepository

```java
package com.singsiuk.board.repository;

import com.singsiuk.board.entity.Reply;
import org.springframework.data.jpa.repository.JpaRepository;

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



## ———————— Test Class 에서 Data 100개 넣기

```java
package com.singsiuk.board;
import com.singsiuk.board.entity.Member;
import com.singsiuk.board.repository.BoardRepository;
import com.singsiuk.board.repository.MemberRepository;
import com.singsiuk.board.repository.ReplyRepository;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class RepositoryTest {
    @Autowired
    private MemberRepository memberRepository;

    @Autowired
    private BoardRepository boardRepository;

    @Autowired
    private ReplyRepository replyRepository;

    @Test
    public void insertMembers(){
        for(int i =1 ; i<=100;i++){
            Member member = Member.builder()
                    .email("user"+i+"@gmail.com")
                    .password("123"+i+"98@")
                    .name("오소리"+i)
                    .build();
            memberRepository.save(member);
        }
    }
}
```

```
select * from tbl_member tm ;
```

tbl_member 1 ×

select * from tbl_member tm | Enter a SQL expression to filter results (use Ctrl+Space)

| | email | moddate | regdate | name | password |
|---|---|---|---|---|---|
| 1 | user1@gmail.com | 2022-01-18 15:41:44.767903000 | 2022-01-18 15:41:44.767903000 | 오소리1 | 123198@ |
| 2 | user10@gmail.com | 2022-01-18 15:41:45.041341000 | 2022-01-18 15:41:45.041341000 | 오소리10 | 1231098@ |
| 3 | user100@gmail.com | 2022-01-18 15:41:46.357755000 | 2022-01-18 15:41:46.357755000 | 오소리100 | 12310098@ |
| 4 | user11@gmail.com | 2022-01-18 15:41:45.068627000 | 2022-01-18 15:41:45.068627000 | 오소리11 | 1231198@ |
| 5 | user12@gmail.com | 2022-01-18 15:41:45.098894000 | 2022-01-18 15:41:45.098894000 | 오소리12 | 1231298@ |
| 6 | user13@gmail.com | 2022-01-18 15:41:45.126823000 | 2022-01-18 15:41:45.126823000 | 오소리13 | 1231398@ |
| 7 | user14@gmail.com | 2022-01-18 15:41:45.147287000 | 2022-01-18 15:41:45.147287000 | 오소리14 | 1231498@ |
| 8 | user15@gmail.com | 2022-01-18 15:41:45.160989000 | 2022-01-18 15:41:45.160989000 | 오소리15 | 1231598@ |
| 9 | user16@gmail.com | 2022-01-18 15:41:45.188906000 | 2022-01-18 15:41:45.188906000 | 오소리16 | 1231698@ |
| 10 | user17@gmail.com | 2022-01-18 15:41:45.204521000 | 2022-01-18 15:41:45.204521000 | 오소리17 | 1231798@ |
| 11 | user18@gmail.com | 2022-01-18 15:41:45.232202000 | 2022-01-18 15:41:45.232202000 | 오소리18 | 1231898@ |

## ———————— 게시글 데이터 100개를 추가할 테스트 메서드로 작성하고 수행

— 게시글의 writer는 반드시 Member 테이블에 존재하는 데이터여야한다
( ManyToOne 으로 설정했기 때문에)

```
@Test
public void insertBoards(){
    for(int i =1 ; i<=100;i++){
        Member member = Member.builder()
                .email("user"+i+"@gmail.com")
                .password("123"+i+"98@")
                .name("오소리"+i)
                .build();
        Board board = Board.builder()
                .title("제목 : "+"인생"+i)
                .content("내용 : "+"날개다람쥐"+i)
                .writer(member)
                .build();
        boardRepository.save(board);

    }
```

```
SELECT * from board b ;
```

ard 1 ×

LECT * from board b  Enter a SQL expression to filter results (use Ctrl+Space)

| bon | moddate | regdate | content | title | writer_email |
|---|---|---|---|---|---|
| 1 | 2022-01-18 15:49:04.700393000 | 2022-01-18 15:49:04.700393000 | 내용 : 날개다람쥐1 | 제목 : 인생1 | user1@gmail.com |
| 2 | 2022-01-18 15:49:04.799747000 | 2022-01-18 15:49:04.799747000 | 내용 : 날개다람쥐2 | 제목 : 인생2 | user2@gmail.com |
| 3 | 2022-01-18 15:49:04.808824000 | 2022-01-18 15:49:04.808824000 | 내용 : 날개다람쥐3 | 제목 : 인생3 | user3@gmail.com |
| 4 | 2022-01-18 15:49:04.819978000 | 2022-01-18 15:49:04.819978000 | 내용 : 날개다람쥐4 | 제목 : 인생4 | user4@gmail.com |
| 5 | 2022-01-18 15:49:04.826174000 | 2022-01-18 15:49:04.826174000 | 내용 : 날개다람쥐5 | 제목 : 인생5 | user5@gmail.com |
| 6 | 2022-01-18 15:49:04.837214000 | 2022-01-18 15:49:04.837214000 | 내용 : 날개다람쥐6 | 제목 : 인생6 | user6@gmail.com |
| 7 | 2022-01-18 15:49:04.843549000 | 2022-01-18 15:49:04.843549000 | 내용 : 날개다람쥐7 | 제목 : 인생7 | user7@gmail.com |
| 8 | 2022-01-18 15:49:04.855487000 | 2022-01-18 15:49:04.855487000 | 내용 : 날개다람쥐8 | 제목 : 인생8 | user8@gmail.com |
| 9 | 2022-01-18 15:49:04.869748000 | 2022-01-18 15:49:04.869748000 | 내용 : 날개다람쥐9 | 제목 : 인생9 | user9@gmail.com |
| 10 | 2022-01-18 15:49:04.880303000 | 2022-01-18 15:49:04.880303000 | 내용 : 날개다람쥐10 | 제목 : 인생10 | user10@gmail.com |

---------- 댓글 데이터를 300개를 삽입하는 TEST Method 를 작성
존재하는 게시글을 기반으로 생성되어야 한다.

```
@Test
    public void insertReplies(){
        Random r = new Random();
        for(long i =1 ; i<=300;i++){
            Board board = Board.builder()
                    .bon((long)(r.nextInt(100)+1))
                    .build();
            Reply reply = Reply.builder()
                    .rno(i)
                    .text("ㅎ"+i)
                    .board(board)
                    .replyer("Customer")
                    .build();
            replyRepository.save(reply);
        }
    }
}
```

```
SELECT * from reply r ;
```

reply 1 ×

SELECT * from reply r | Enter a SQL expression to filter results (use Ctrl+Space)

| | rno | moddate | regdate | replyer | text | board_bon |
|---|---|---|---|---|---|---|
| 291 | 291 | 2022-01-18 16:00:04.727435000 | 2022-01-18 16:00:04.727435000 | Customer | ㅎ291 | 30 |
| 292 | 292 | 2022-01-18 16:00:04.732910000 | 2022-01-18 16:00:04.732910000 | Customer | ㅎ292 | 62 |
| 293 | 293 | 2022-01-18 16:00:04.736169000 | 2022-01-18 16:00:04.736169000 | Customer | ㅎ293 | 19 |
| 294 | 294 | 2022-01-18 16:00:04.740811000 | 2022-01-18 16:00:04.740811000 | Customer | ㅎ294 | 69 |
| 295 | 295 | 2022-01-18 16:00:04.746387000 | 2022-01-18 16:00:04.746387000 | Customer | ㅎ295 | 3 |
| 296 | 296 | 2022-01-18 16:00:04.751480000 | 2022-01-18 16:00:04.751480000 | Customer | ㅎ296 | 94 |
| 297 | 297 | 2022-01-18 16:00:04.755663000 | 2022-01-18 16:00:04.755663000 | Customer | ㅎ297 | 52 |
| 298 | 298 | 2022-01-18 16:00:04.759009000 | 2022-01-18 16:00:04.759009000 | Customer | ㅎ298 | 43 |
| 299 | 299 | 2022-01-18 16:00:04.762849000 | 2022-01-18 16:00:04.762849000 | Customer | ㅎ299 | 53 |
| 300 | 300 | 2022-01-18 16:00:04.769167000 | 2022-01-18 16:00:04.769167000 | Customer | ㅎ300 | 43 |

## ———————————— Eager & Lazy Loading

**@OneToMany 나 @ManyToOne 으로 연관관계가 설정된 상태에서**
**데이터를 검색하면 자동으로 join을 수행해서 데이터를 가져온다**

**Board 테이블의 데이터를 조회할 때**
**Member 와 Join을 해서 Member 의 내용을 가져온다**
**——› 이 방식이 Eager Loading 이다**

**———————————— TEST method 를 작성하고 실행한 후 수행된 SQL을 확인**

```
// 게시물 1 개를 가져오는 method 테스트
    @Test
    public void eagerLoading(){
        Optional<Board> board = boardRepository.findById(100L);
        if(board.isPresent()){
            System.out.println(board.get());
            System.out.println(board.get().getWriter());
        }
    }
}
```

—> 데이터 베이스에서 가장 부담이 되는 SQL = JOIN

```
    member1_.password as password5_2_1_
from
    board board0_
left outer join
    tbl_member member1_
        on board0_.writer_email=member1_.email
where
    board0_.bon=?
ard(bon=100, title=제목 : 인생100, content=내용 : 날개다람쥐100, writer=Member(email=user100@gmail.com, password=12310098@, name=오소리100))
mber(email=user100@gmail.com, password=12310098@, name=오소리100)
22-01-18 16:14:05.467  INFO 37632 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory fo
22-01-18 16:14:05.477  INFO 37632 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated..
22-01-18 16:14:05.506  INFO 37632 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
ILD SUCCESSFUL in 14s
actionable tasks: 2 executed, 2 up-to-date
14:05 PM: Execution finished ':test --tests "com.singsiuk.board.RepositoryTest.eagerLoading"'.
```

## ———————————— Eager Loading(즉시 로딩)

특정 Entity를 조회 시 연관관계를 갖는

모든 Entity를 같이 Loading 하는 것

그래서 한번만 가져오면 연관된 모든 데이터를 가져오기 때문에

데이터를 조회하는 method 의 호출 횟수가 적다

〈——〉

연관관계가 복잡해지면 Join으로 인한 성능 저하 발생

## ———————————— lazy Loading(지연 로딩)

필요할 때 데이터를 직접 가져오는 방법

— 단순하게 하나의 Table 의 데이터만 이용하는 경우 조회 속도 빠름

— 필요할때 마다 쿼리를 실행해야 하기 때문에 여러번의 쿼리가 실행 될 수도 있음

## ———————————— JPA 에서의 설정

관계 Annotation(fetch=FetchType.모드)

* 특별한 경우가 아니면 Lazy로 설정

—〉 왜냐하면 Eager로 설정시 memory를 과하게 많이 사용하기 때문에

## ———————————— 연관관계를 전부 Lazy Loading 으로 변경

### 1 . ———— BoardEntity 관계를 수정

```
package com.singsiuk.board.entity;

import lombok.*;
import net.bytebuddy.dynamic.loading.InjectionClassLoader;

import javax.persistence.*;

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

    @ManyToOne
    private Member writer;
}
```

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

## 2. ———————— replyEntity 수정

```java
package com.singsiuk.board.entity;

import lombok.*;

import javax.persistence.*;

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

    @ManyToOne
    private Board board;
}
```

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

## ———————————————— 하나의 Board를 가져와서 출력하는 Code 를 작성

```java
@Test
public void lazyLoading(){
    Optional<Board> board = boardRepository.findById(100L);
    if(board.isPresent()){
        System.out.println(board.get());
        System.out.println(board.get().getWriter());
    }
}
```

## ————〉 Error 발생(NoSession Error)

```
    where
        board0_.bon=?

could not initialize proxy [com.singsiuk.board.entity.Member#user100@gmail.com] - no Session
org.hibernate.LazyInitializationException: could not initialize proxy [com.singsiuk.board.entity.Member#user100@gmail.com] - no Session
    at app//org.hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyInitializer.java:176)
    at app//org.hibernate.proxy.AbstractLazyInitializer.getImplementation(AbstractLazyInitializer.java:322)
    at app//org.hibernate.proxy.pojo.bytebuddy.ByteBuddyInterceptor.intercept(ByteBuddyInterceptor.java:45)
    at app//org.hibernate.proxy.ProxyConfiguration$InterceptorDispatcher.intercept(ProxyConfiguration.java:95)
    at app//com.singsiuk.board.entity.Member$HibernateProxy$0qJ3wGFM.toString(Unknown Source)
    at java.base@11.0.12/java.lang.String.valueOf(String.java:2951)
```

ToString 은 기본적으로 자신이 참조하고 있는 DATA 에 ToString()을 호출하는데
————〉 지연 로딩을 사용하도록 설정했기 때문에 Writer 속성의 값이
아직 호출되지 않은 상태에서 toString 을 호출해서 Error 발생

## ———— Session Error

: Hibernate 가 Board의 데이터를 가져오고 Session 을 닫아서
Board가 참조하고 있는 Member의 Data를 가져오지 못했기 때문

## ————————— ›문제 해결 방법

Transaction을 사용해서 Board 데이터를 가져오고
Session 을 닫지 말고
Member data를 가져오도록 해주면 된다.

## board 와 reply Entity 에서 ToString() 속성을 변경 ————————————————

### 1. Board

```
package com.singsiuk.board.entity;

import lombok.*;
import net.bytebuddy.dynamic.loading.InjectionClassLoader;

import javax.persistence.*;

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

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

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

```
package com.singsiuk.board.entity;

import lombok.*;
import net.bytebuddy.dynamic.loading.InjectionClassLoader;

import javax.persistence.*;

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

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

## 2. Reply

```
package com.singsiuk.board.entity;

import lombok.*;

import javax.persistence.*;

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

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

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

```
package com.singsiuk.board.entity;

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

## 3. RepositoryTest 수정

```
@Test
    public void lazyLoading(){
        Optional<Board> board = boardRepository.findById(100L);
        if(board.isPresent()){
            System.out.println(board.get());
            System.out.println(board.get().getWriter());
        }
    }
}
```

```
@Transactional
```

```
@Test
    @Transactional
    public void lazyLoading(){
        Optional<Board> board = boardRepository.findById(100L);
        if(board.isPresent()){
            System.out.println(board.get());
            System.out.println(board.get().getWriter());
        }
    }
}
```

————————————————————————〉 수행되는 SQL을 확인해서 Join 하지 않고
2개의 SQL로 나누어서 처리하는지 확인

```
Hibernate:
    select
        board0_.bon as bon1_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_
    from
        board board0_
    where
        board0_.bon=?
Board(bon=100, title=제목 : 인생100, content=내용 : 날개다람쥐100)
Hibernate:
    select
        member0_.email as email1_2_0_,
        member0_.moddate as moddate2_2_0_,
        member0_.regdate as regdate3_2_0_,
        member0_.name as name4_2_0_,
        member0_.password as password5_2_0_
    from
        tbl_member member0_
    where
        member0_.email=?
Member(email=user100@gmail.com, password=12310098@, name=오소리100)
```

———————————————————————— JPQL 을 이용한 Join

: 여러개의 Entity 조합으로 데이터를 가져오는 경우
기존 Entity 타입만으로 해결할 수 없는 경우가 있다
게시물  정보와 댓글의 개수를 가져오는 경우에는 하나의 Entity 만으로 불가하다.

——〉 이 경우 JOIN 을 이용

JPQL 을 이용할 때는 연관관계가 있는지
여부에 따라 On을 사용해서
Join 조건을 설정 해주어야 한다.

연관 관계가 있는 경우 On을 생략하지만
연관관계가 없는 경우 On 을 설정 해야한다.

# 1. 연관관계가 있는 경우의 Join ———
# BoardRepository Interface에 Member 와 Board Entity의 Join 을 수행하는 method 생성

## ——— BoardRepository에 생성

```
package com.singsiuk.board.repository;

import com.singsiuk.board.entity.Board;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface BoardRepository extends JpaRepository<Board, Long> {
    // Member 와 Board Entity 의 Join을 수행하는 method를 생성
    // Board Entity 에는 Member  Entit와 연관관계를 갖는 writer 가 존재

    // bon 에 해당하는 Board 를 가져올 때 Member에 대한 정보도 가져오기
    // w 가 member table
    // on 이 들어가지 않음
    @Query("select b,w from Board b  left join b.writer w where b.bon=:bon")
    Object getBoardWithWriter(@Param("bon”)Long bon);


}
```

## ——— Test Method 를 만들어서 확인

```
@Test
    public void testJoin(){
        Object result = boardRepository.getBoardWithWriter(100L);
        // 배열로 Return 되기 때문
        Object [] arr = (Object [])result;
        System.out.println(Arrays.toString(arr));
    }
}
```

```
Hibernate:
    select
        board0_.bon as bon1_0_0_,
        member1_.email as email1_2_1_,
        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_.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 outer join
        tbl_member member1_
            on board0_.writer_email=member1_.email
    where
        board0_.bon=?
[Board(bon=100, title=제목 : 인생100, content=내용 : 날개다람쥐100), Member(email=user100@gmail.com, password=12310098@, name=오소리100)]
```

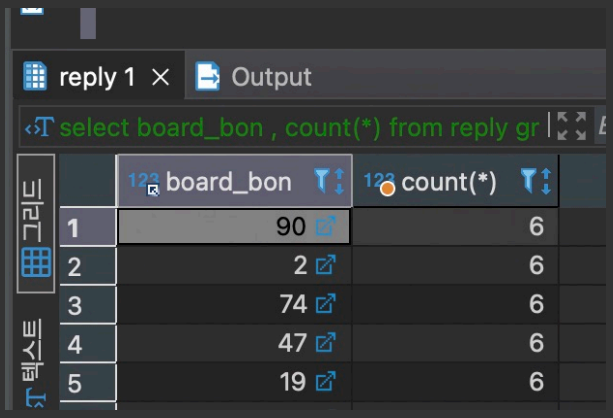
# 연관관계가 없는 경우 JOIN

## BoardRespository
## Interface에 Board 와 Reply Entity를
## Join해서
## // 일반적으로 하나의 정보를 가져올 때 댓글의 정보를 가져옴

```
package com.singsiuk.board.repository;

import com.singsiuk.board.entity.Board;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface BoardRepository extends JpaRepository<Board, Long> {
    // Member 와 Board Entity 의 Join을 수행하는 method를 생성
    // Board Entity 에는 Member  Entit와 연관관계를 갖는 writer 가 존재

    // bon 에 해당하는 Board 를 가져올 때 Member에 대한 정보도 가져오기
    // w 가 member table
    // on 이 들어가지 않음
    @Query("select b,w from Board b  left join b.writer w where b.bon=:bon")
    Object getBoardWithWriter(@Param("bon")Long bon);

     // Reply 와 Board Entity 의 Join을 수행하는 method를 생성
     // Board Entity 에는 Reply  Entity와 연관관계를 가지고 있지 않음
     // 양쪽에 공통된 속성을 찾아야한다
     // reply 가 Board 의 정보를 Board 라는 속성으로 가지고 있음

     // bon 에 해당하는 Board 를 가져올 때 Member에 대한 정보도 가져오기
     @Query("select b,r from Board b  left join Reply r on r.board = b  where b.bon=:bon")
     // 하나의 게시글에 댓글이 여러개 있을 수 있어서 Return Type 은 List
     List<Object[]> getBoardWithReply(@Param("bon")Long bon);
}
```

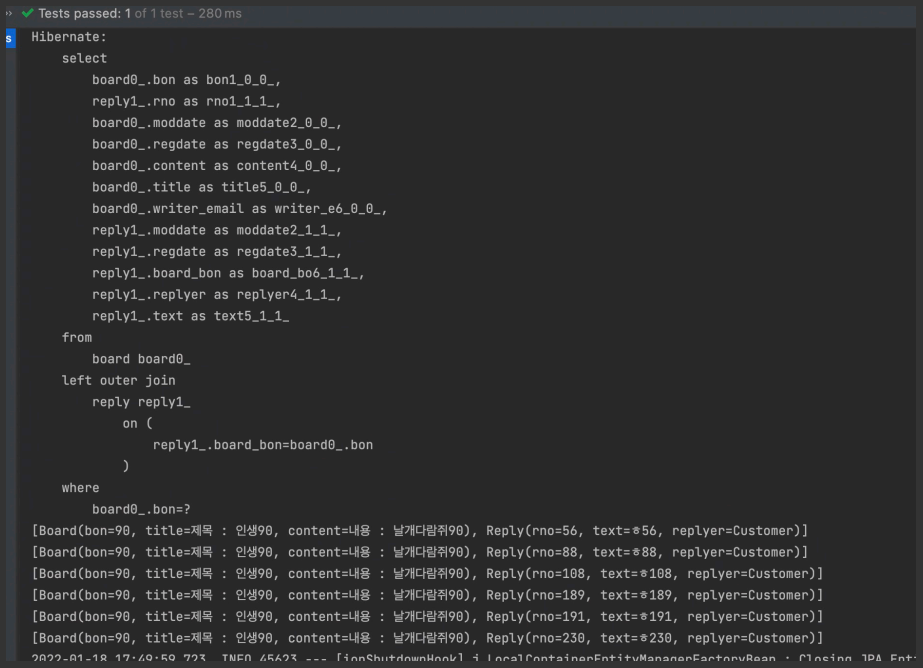
## 데이터 베이스에서 댓글이 많은 순으로 글번호 조회

```
select  board_bon , count(*)
from reply
group by board_bon
order by count(*) desc;
```



## Test Class 에서 Method 생성

```
@Test
    public void testJoin2(){
        List<Object[]> result =boardRepository.getBoardWithReply(90L);
            for(Object [] ar:result){
                System.out.println(Arrays.toString(ar));
            }
    }
}
```




## 목록 보기 화면에 필요한 method

필요한 데이터

- Board: 게시물 번호 ,게시물 제목 ,작성 시간
- Member : 이름과 Email
- Reply : 댓글 수를 가져오기

* 집계를 할때는 Group By (Query) 가 들어감

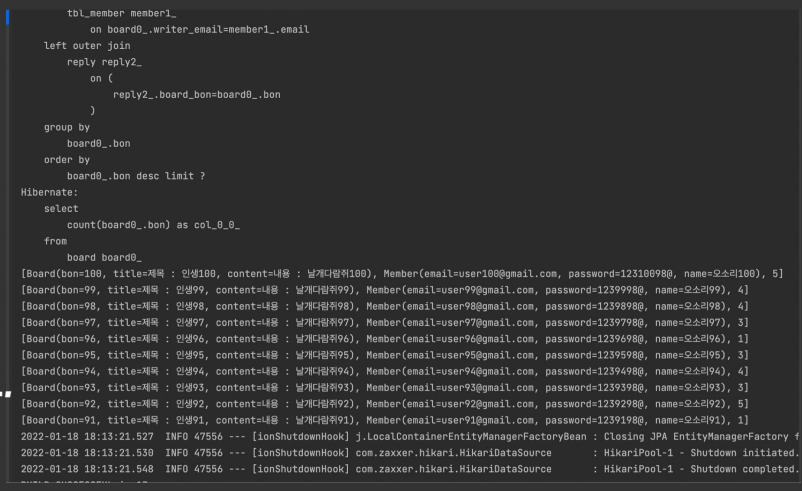
## BoardRepository에 Interface 목록 보기를 위한 Method 선언

```
package com.singsiuk.board.repository;

import com.singsiuk.board.entity.Board;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface BoardRepository extends JpaRepository<Board, Long> {
    // Member 와 Board Entity 의 Join을 수행하는 method를 생성
    // Board Entity 에는 Member  Entity와 연관관계를 갖는 writer 가 존재

    // bon 에 해당하는 Board 를 가져올 때 Member에 대한 정보도 가져오기
    // w 가 member table
    // on 이 들어가지 않음
    @Query("select b,w from Board b  left join b.writer w where b.bon=:bon")
    Object getBoardWithWriter(@Param("bon")Long bon);

    // Reply 와 Board Entity 의 Join을 수행하는 method를 생성
    // Board Entity 에는 Reply  Entity와 연관관계를 가지고 있지 않음
    // 양쪽에 공통된 속성을 찾아야한다
    // reply 가 Board 의 정보를 Board 라는 속성으로 가지고 있음

    // bon 에 해당하는 Board 를 가져올 때 Member에 대한 정보도 가져오기
    @Query("select b,r from Board b  left join Reply r on r.board = b  where b.bon=:bon")
    // 게시판에 댓글이 여러개 있을 수 있어서 Return Type 은 List
    List<Object[]> getBoardWithReply(@Param("bon")Long bon);
```

```
    // 목록을 보기 위한 method
    // JPQL 에서는 Page 단위로 리턴시 CountQuery 가 필수
    @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);
}
```

TEST ————————————————————————————————————————

```java
@Test
    public void testBoardList(){
        // 페이징 조건 생성
        // 0 page 에서 10개의 데이터를
        // bon 의 내림차순으로 가져오기
        Pageable pageable = PageRequest.of(0,10,
                Sort.by("bon").descending());

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



```
        tbl_member member1_
            on board0_.writer_email=member1_.email
    left outer join
        reply reply2_
            on (
                reply2_.board_bon=board0_.bon
            )
    group by
        board0_.bon
    order by
        board0_.bon desc limit ?
Hibernate:
    select
        count(board0_.bon) as col_0_0_
    from
        board board0_
[Board(bon=100, title=제목 : 인생100, content=내용 : 날개다람쥐100), Member(email=user100@gmail.com, password=12310098@, name=오소리100), 5]
[Board(bon=99, title=제목 : 인생99, content=내용 : 날개다람쥐99), Member(email=user99@gmail.com, password=1239998@, name=오소리99), 4]
[Board(bon=98, title=제목 : 인생98, content=내용 : 날개다람쥐98), Member(email=user98@gmail.com, password=1239898@, name=오소리98), 4]
[Board(bon=97, title=제목 : 인생97, content=내용 : 날개다람쥐97), Member(email=user97@gmail.com, password=1239798@, name=오소리97), 3]
[Board(bon=96, title=제목 : 인생96, content=내용 : 날개다람쥐96), Member(email=user96@gmail.com, password=1239698@, name=오소리96), 1]
[Board(bon=95, title=제목 : 인생95, content=내용 : 날개다람쥐95), Member(email=user95@gmail.com, password=1239598@, name=오소리95), 3]
[Board(bon=94, title=제목 : 인생94, content=내용 : 날개다람쥐94), Member(email=user94@gmail.com, password=1239498@, name=오소리94), 4]
[Board(bon=93, title=제목 : 인생93, content=내용 : 날개다람쥐93), Member(email=user93@gmail.com, password=1239398@, name=오소리93), 3]
[Board(bon=92, title=제목 : 인생92, content=내용 : 날개다람쥐92), Member(email=user92@gmail.com, password=1239298@, name=오소리92), 5]
[Board(bon=91, title=제목 : 인생91, content=내용 : 날개다람쥐91), Member(email=user91@gmail.com, password=1239198@, name=오소리91), 1]
2022-01-18 18:13:21.527  INFO 47556 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory f
2022-01-18 18:13:21.530  INFO 47556 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated.
2022-01-18 18:13:21.548  INFO 47556 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

Hibernate:
    select
        board0_.bon as col_0_0_,
        member1_.email as col_1_0_,
        count(reply2_.rno) as col_2_0_,
        board0_.bon as bon1_0_0_,
        member1_.email as email1_2_1_,
        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_.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 outer join
        tbl_member member1_
            on board0_.writer_email=member1_.email
    left outer join
        reply reply2_
            on (
                reply2_.board_bon=board0_.bon
            )
    group by
        board0_.bon
    order by
        board0_.bon desc limit ?
Hibernate:
    select
        count(board0_.bon) as col_0_0_
    from
        board board0_