

목차

1. 프로젝트 설명

- a. 프로젝트 폴더 계층 구조
- b. 사용 기술 및 언어 본인 역할

2. ERD

3. 페이징 분석

- a. 페이징 구조 분석
- b. 페이징 공식 도출

4. 답변형 게시판(고객센터 게시판) 분석

- a. 답변형 게시판 예시 그림
- b. 추상화 및 부분 구조 분석
- c. 구조 확대 및 전체 구조 분석

5. 게시판 및 댓글 관련 기능

- a. 게시판 종류 및 설명
- b. 게시판 **CRUD**
- c. 답변형 게시판 **CRUD**
- d. 게시글 검색
- e. 댓글 및 대댓글(중첩 페이징)

6. 유저 관련 기능

- a. 내가 쓴 글 확인

7. 관리자 관련 기능

- a. 글 및 회원관리

8. ERD 개선점

- a. 추천기능 문제점 및 개선점

- b. 게시판 관리 관점
- c. 선택한 관리 관점 **ERD** 구현
- d. 데이터(댓글)로딩 설명 및 추가 고려점

9. 프로젝트 느낀점

1-a. 프로젝트 폴더 계층구조(MVC 모델)

*HealthChain->src->main에 가보면 **java**폴더와 **webapp**폴더가 있는데 이 두 폴더 기준으로 설명드립니다. 아래 구조에서 설명안된 것들은 설정관련 파일들 이라 생각하시면 됩니다.

- **java**
 - **api** : api 관련 기능.
 - **controller** : MVC 중 Controller.
 - **model** : MVC 중 Model(DTO).
 - **dao** : DAO
 - **service** : MVC 중 Model의 service 레이어.
 - **xml files** : Mybatis SQL 코드들.
- **webapp**
 - **index.jsp** : 프로그램 진입점.
 - **view** : MVC 중 View.

1-b. 사용기술 및 언어 및 본인 역할

- 기술 : JSP, Servlet, ORACLE, Mybatis, AJAX
- 언어 : JS, JAVA, SQL
- 본인역할 : 게시판 CRUD, 마이페이지, 검색, 댓글 및 대댓글, 페이징, 관리자 페이지

2. ERD

ERD 주소 : <https://www.erdcloud.com/d/r6jwhxYsMndAXySjf>

설명 : 사용자 및 의사가 자유롭게 의견을 주고 받을 수 있는 사이트입니다.

3-a. 페이징 구조 분석

설명 : 주로 게시판을 만드는 프로젝트이므로 페이징이 필요하다고 생각했습니다. 페이징에 대해 본인이 어떻게 생각했는지 이러한 생각을 어떻게 구현했는지에 대해 설명을 하겠습니다.

예시 게시판 사진 :

뉴스

커뮤니티

하늘의 퀘지 왓맨
코드 이벤트 진행

리뷰

게이머존

라이브/영상

게이밍/IT

게임스토어

FM26
예약 판매 중

보더랜드4 인벤게임즈 입점

5685661	[수다] 저그가 이론상 제일 사기여야하는 종족 아닌가 [14]	감낭	13:55	95	0
5685660	[수다] 비틱) 플라즈마 하트 균형의 수호자들에게 [2]	피영	13:55	72	1
5685659	[수다] 요즘 메봉이들 답글도 안달아주고 말이야 [16]	아내	13:55	34	0
5685658	[수다] 스타포스 성공적	루이드엘	13:55	47	0
5685657	[수다] 숨숨이 아가방 어찌고 보고 충격 받음 [10]	말랭이	13:55	122	0
5685656	[수다] 츄츄지지 스타포스 안 보이는데 어떻게 보나요 [3]	Wound	13:55	37	0
5685655	[수다] 신규 에픽던전 보스 엄청세네	에스페라	13:54	45	0
5685654	[수다] 알색 드롭률보니까 마지막주품에 개비싸질거같음	도체리아	13:54	53	0
5685653	[수다] 귀칼콜라보 1년만 늦게했다면!!!! [4]	엔젤릭버스터	13:54	95	0
5685652	[수다] 그 웹폰3렙단들이 문제역자녀	문주윤	13:54	29	0
5685650	[수다] 언컨 가격 보면 풀하는 그냥 해자로보임 [1]	꾸여역	13:53	77	0
5685649	[수다] 부캐 풀드메렘 팔까요? ≡ 체력이 안됨 [2]	Wakeup	13:53	81	0

목록

글쓰기

< 이전

1

2

3

4

5

6

7

8

9

10

다음 >

제목

▼

Q

역할 분석 : 저는 우선 클라이언트의 역할이 무엇인지, 서버의 역할은 무엇인지, **DB**의 역할이 무엇인지를 가장 먼저 고려하였습니다.

클라이언트 분석: 클라이언트가 무엇을 요청하는지에 대해 아래와 같은 그림을 생각해 볼 수 있습니다. 브라우저 사용자의 행동(마우스 클릭)을 중점으로 생각하면 됩니다.

뉴스

커뮤니티

하늘의 귀적 갯면
코드 이벤트 진행

리뷰

게이머존

라이브/영상

게이밍/IT

게임스토어

FM26
예약 판매 중

보더랜드4 인벤게임즈 입점

요청1

요청2

5685661	[수다] 저그가 이론상 제일 사기여야하는 종족 아닌가 [14]	깜냥	13:55	95	0
5685660	[수다] 비텍) 플라즈마 하트 균형의 수호자들에게 [2]	피영	13:55	72	1
5685659	[수다] 요즘 메봉이들 답글도 안달아주고 말이야 [16]	아내	13:55	34	0
5685658	[수다] 스타포스 성공적	루이드엘	13:55	47	0
5685657	[수다] 솜솜이 아가방 어찌고 보고 충격 받은 [10]	말랭이	13:55	122	0
5685656	[수다] 추추지지 스타포스 안 보이는데 어떻게 보나요 [3]	Wound	13:55	37	0
5685655	[수다] 신규 에픽던전 보스 엄청세네	에스페라	13:54	45	0
5685654	[수다] 알색 드롭률보니까 마지막주쯤에 개비싸질거같음	도체리아	13:54	53	0
5685653	[수다] 귀칼콜라보 1년만 늦게했다면!!!! [4]	엔젤릭버스터	13:54	95	0
5685652	[수다] 그 웹폰3렙단들이 문제역자녀	문주윤	13:54	29	0
5685650	[수다] 언컨 가격 보면 풀하는 그냥 해자로보임 [1]	꾸여역	13:53	77	0
5685649	[수다] 부캐 폴드메템 팔까요? ㅇㅇ 체력이 안됨 [2]	Wakeup	13:53	81	0

목록

글쓰기

< 이전

1

2

3

4

5

6

7

8

9

10

다음 >

제목

▼

Q

크게 **2**가지 요청으로 나뉘게 됩니다. **요청1**은 페이징 이후의 특정 게시판에 대한 데이터를 요청하고, **요청2**는 페이징을 하기위해 해당 페이지 번호를 요청을 하게 됩니다. **요청1**은 페이징 이후의 처리이므로 우리는 요청2를 먼저 고려해야 합니다.

서버 분석 : 클라이언트의 역할이 정해 졌으니 서버의 역할을 찾아야 합니다. 그런데 클라이언트에서 요청한게 페이지번호 하나뿐입니다.

어떻게 해야 할지 막막합니다. 게다가 옆친데 덮친격으로 또 고려해야 할 사항이 있는데, 각각의 파란색 테두리의 보여주는 개수(게시글 보여주는 개수, 페이지 보여주는 개수) 또한 변수이므로 고려해야 합니다.

이로써 총 **3**개의 변수를 생각해야 합니다.

3개의 변수는 다음과 같이 나뉩니다.

- 서버용 변수 : 게시글 보여주는 개수, 페이지 보여주는 개수
- 클라이언트 요청 변수 : 페이지 번호

DB분석 : DB는 서버의 요청으로 데이터를 얼마큼 가져올 것인가를 생각해야 합니다. 그럼 어떻게 얼마큼 가져올것에 대해 설명을 하자면,

우선 **DB** 테이블은 **2차원**인데 우리 데이터를 가져오는것만이 목표이므로 **1차원(배열)**으로 생각해 해결해 나가는게 좋다 생각이 듭니다.

즉, 전체 테이블을 하나의 배열로 생각해 각 페이지별로 테이블구역을 정해 데이터를 가져오는게 **DB**의 역할입니다.

3-b. 페이징 공식 도출

- 변수 설정

- **limitBoardNum(> 0)** : 게시글 보여지는 개수(서버용 변수)
- **limitPageNum(> 0)** : 페이지 보여주는 개수(서버용 변수)
- **totalBoardNum(>= 0)** : 전체 게시글 갯수(서버용 변수, DB연동 필요)
- **reqPage(integer)** : 요청 페이지

***totalBoardNum 설명** : 이 변수는 제가 처음에 뭘 할지 모르다가 나중에 생각해낸 변수입니다. 최대페이지등을 구할 때 사용 됩니다.

페이징을 통해 구역을 일정하게 나누려면 당연히 나눌대상이 필요한데 이 대상이 **totalBoardNum** 입니다.

최대 페이지 도출 방법

totalBoardNum	totalBoardNum / limitBoardNum	maxPage
0 = limitBoardNum * 0	0	1
limitBoardNum * 0 + 1	0	1

~	~	~
limitBoardNum * 1	1	1
limitBoardNum * 1 + 1	1	2
~	~	~
limitBoardNum * 2	2	2
limitBoardNum * 2 + 1	2	3
~	~	~
limitBoardNum * 3	3	3

설명 : **totalBoardNum**을 생각할 때 단순히 **0, 1, 2, 3, 4, ...** 이렇게 보는게 아니라 구역을 나누기 위해 위와 같이 규칙을 두어 만드는데 좋습니다.

빨간색 테두리 영역은 개수가 **1**개라 일반적인 테두리 영역과 다르지만, 빨간색 테두리 영역이 있는데 끝부분만 남기고 나머지는 잘랐다고 생각하시면 됩니다.

그리고 **totalBoardNum / limitBoardNum** 을 표시한 이유는 전체구역을 limitBoardNum로 일정하게 나누는게 포인트이기 때문에 해당 값의 변화를 체크하기 위해 표시를 했습니다.

그리고 각 테두리 영역을 **n**이 (**>= 0**)라 가정을 한다면 다음과 같은 식이 나오게 됩니다.

$$\underline{\text{totalBoardNum} = \text{limitBoardNum} * (n - 1) + x, (1 \leq x \leq \text{limitBoardNum})}$$

그런다음 **totalBoardNum**을 **limitBoardNum**로 나머지 연산을 하게 되면

1 <= x < limitBoardNum 경우에는 나머지가 **0**이 아니며, **x = limitBoardNum** 인 경우에는 나머지가 **0**입니다.

즉, $\text{totalBoardNum} \% \text{limitBoardNum}$ 값이 0인지 아닌지에 따라 maxPage 가 결정됩니다.

따라서, maxPage 는 다음과 같이 쓸 수 있습니다.

$\text{maxPage} = (\text{totalBoardNum} / \text{limitBoardNum}) + 1$
when $\text{totalBoardNum} \% \text{limitBoardNum}$ is not 0.

$\text{maxPage} = \text{totalBoardNum} / \text{limitBoardNum}$
when $\text{totalBoardNum} \% \text{limitBoardNum}$ is 0.

가져오는 범위 도출 방법

range	reqPage
0 ~ $\text{limitBoardNum} - 1$	1
$\text{limitBoardNum} \sim 2 * \text{limitBoardNum} - 1$	2
$2 * \text{limitBoardNum} \sim 3 * \text{limitBoardNum} - 1$	3

위 표는 아래표와 같습니다.

range	reqPage
$(\text{reqPage} - 1) * \text{limitBoardNum} \sim \text{reqPage} * \text{limitBoardNum} - 1$	1
$(\text{reqPage} - 1) * \text{limitBoardNum} \sim \text{reqPage} * \text{limitBoardNum} - 1$	2
$(\text{reqPage} - 1) * \text{limitBoardNum} \sim \text{reqPage} * \text{limitBoardNum} - 1$	3

따라서,

$\text{startIdx} = (\text{reqPage} - 1) * \text{limitBoardNum}$
 $\text{endIdx} = \text{reqPage} * \text{limitBoardNum} - 1$

***추가설명 :** startIdx 와 endIdx 는 가져오는 범위를 나타내는데 전체를 가져오는 범위를 의미합니다. 이렇게 하면 **out of range** 와 같은 오류가 날 수가 있으나 **ORACLE DBMS** 같은 경우에는 **BETWEEN** 절에 **out of range**가 발생하더라도 오류가 나지 않으니 오류체크를 해도되고 안해도 됩니다.

시작 페이지 도출 방법

startPage	reqPage
$1 = \text{limitPageNum} * 0 + 1$	$1 = \text{limitPageNum} * 0 + 1$
~	~
$1 = \text{limitPageNum} * 0 + 1$	$\text{limitPageNum} * 1$
$\text{limitPageNum} * 1 + 1$	$\text{limitPageNum} * 1 + 1$
~	~
$\text{limitPageNum} * 1 + 1$	$\text{limitPageNum} * 2$
$\text{limitPageNum} * 2 + 1$	$\text{limitPageNum} * 2 + 1$
~	~
$\text{limitPageNum} * 2 + 1$	$\text{limitPageNum} * 3$

설명 : 테두리 영역의 시퀀스를 $n (\geq 0)$ 이라 가정한다면, **startPage = limitPageNum * n + 1** 이 됩니다. 하지만, n은 임의의 가정된 수 이기 때문에 n을 다른 실질적인 변수로 치환해줘야 합니다. 또한, 위의 표에서 다음과 같은 식을 얻을 수 있습니다.

$$\text{reqPage} - 1 = \text{limitPageNum} * n + x,$$

$$(0 \leq x < \text{limitPageNum})$$

reqPage - 1을 한 이유는 동일한 값을 얻기 위해서 입니다.

그러면, 해당 식을 나누기(몫) 연산으로 나눠주면 됩니다. 나머지 연산이 아닌 나누기 연산을 하는 이유는 나머지 연산은 자연수 등차수열에 대해서 각각의 구역에 대해 일정한 값을 갖지 못하지만 나누기 연산은 그것이 가능하기 때문에 나누기 연산을 해야 합니다.

나누기 연산을 하게 되면 다음과 같은 식을 얻을 수 있습니다.

$$n = ((\text{reqPage} - 1) / \text{limitPageNum})$$

따라서, 시작 페이지는 다음의 식으로 쓸 수 있습니다.

$$\text{startPage} = \text{limitPageNum} * ((\text{reqPage} - 1) / \text{limitPageNum}) + 1$$

끝, 이전, 다음 페이지 도출 방법

endPage	reqPage
limitPageNum * 1	1 = limitPageNum * 0 + 1
~	~
limitPageNum * 1	limitPageNum * 1
limitPageNum * 2	limitPageNum * 1 + 1
~	~
limitPageNum * 2	limitPageNum * 2
limitPageNum * 3	limitPageNum * 2 + 1
~	~
limitPageNum * 3	limitPageNum * 3

설명 : 시작페이지 도출 방법과 유사하기 때문에 자세한 설명은 생략하겠습니다. 테두리 영역 시퀀스 n (≥ 1) 에 대하여,

$$\text{endPage} = \text{limitPageNum} * n,$$

$$\text{reqPage} - 1 = \text{limitPageNum} * (n - 1) + x, (0 \leq x < \text{limitPageNum}),$$

$$(\text{reqPage} - 1) / \text{limitPageNum} = n - 1$$

endPage = limitPageNum * (((reqPage - 1) / limitPageNum) + 1)

다음, 이전 페이지는 쉽습니다.

nextPage = endPage + 1

prevPage = startPage - 1

테스트 코드 : 위와 같이 우선 테스트 코드작성을 통해 페이지징이 제대로 작동하는지 먼저 테스트를 하고 프로젝트에 적용을 하였습니다. 테스트 코드는 다음과 같은 사진들이며, **Board.java**, **BoardClient.java(main함수 진입점)**, **BoardServer.java** 파일이 해당 테스트 코드파일들 입니다.

The screenshot shows an IDE with three files: BoardClient.java, BoardServer.java, and Board.java. The BoardClient.java file is open, showing a main method that calls reqBoards and prints the results. The output window shows the results of the reqBoards method call.

```
1 import org.json.JSONArray;
2 import org.json.JSONObject;
3 import java.util.*;
4
5 public class BoardClient {
6     public static void main(String[] args) {
7         BoardServer boardServer = new BoardServer( totalBoardNum: 31, limitBoardNum: 15, limitPageNum: 10);
8         Scanner sc = new Scanner(System.in);
9
10        JSONObject res = reqBoards(boardServer, page: 2);
11        // System.out.println(res);
12        System.out.println("boardList : " + res.get("boardList"));
13        System.out.println("page : " + res.get("page"));
14        System.out.println("startPage : " + res.get("startPage"));
15        System.out.println("endPage : " + res.get("endPage"));
16        System.out.println("nextPage : " + res.get("nextPage"));
17        System.out.println("prevPage : " + res.get("prevPage"));
18    }
19    @ public static JSONObject reqBoards(BoardServer boardServer, int page) {
20        JSONObject req = new JSONObject();
```

Run BoardClient

```
"C:\Program Files\Java\jdk-11.0.16.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IOEA Community Edition 2024.2.3\lib\idea_rt.jar=49332:C:\Program Files\JetBrains\IntelliJ IOEA Community Edition 2024.2.3\bin" -Didea.config.path=C:\Program Files\JetBrains\IntelliJ IOEA Community Edition 2024.2.3\conf -Didea.copyright.path=C:\Program Files\JetBrains\IntelliJ IOEA Community Edition 2024.2.3\copyright -Didea.home.path=C:\Program Files\JetBrains\IntelliJ IOEA Community Edition 2024.2.3\bin -Didea.platform.prefix=JDK -Didea.vendor.id=ioea -Didea.version=2024.2.3 -jar C:\Program Files\JetBrains\IntelliJ IOEA Community Edition 2024.2.3\bin\idea_rt.jar 49332
maxPage : 3
startIdx : 15
endIdx : 29
boardList : ["15","16","17","18","19","20","21","22","23","24","25","26","27","28","29"]
page : 2
startPage : 1
endPage : 3
nextPage : 3
```

```

public class BoardServer { 3 usages
    public BoardServer(int totalBoardNum, int limitBoardNum, int limitPageNum) { 1 usage
        this.totalBoardNum = totalBoardNum;
        this.limitBoardNum = limitBoardNum;
        this.limitPageNum = limitPageNum;

        for(int i = 0; i < totalBoardNum; i++) {
            String idx = "" + i;
            boards.add(new Board(idx));
        }
    }

    public JSONObject resBoards(JSONObject req) { 1 usage
        JSONObject res = new JSONObject();

        int maxPage;
        if(totalBoardNum <= 0) {
            maxPage = 1;
        }
        else if(totalBoardNum % limitBoardNum == 0) {
            maxPage = totalBoardNum / limitBoardNum;
        }
        else {
            maxPage = (totalBoardNum / limitBoardNum) + 1;
        }
        System.out.println("maxPage : " + maxPage);
        int reqPage = req.getInt( key, "page");
        if(reqPage < 1) {
            reqPage = 1;
        }
        else if(reqPage > maxPage) {
            reqPage = maxPage;
        }
    }
}

```

```

public class Board { 4 usages
    private String boardID; 3 usages

    public Board(String boardID) { 1 usage
        this.boardID = boardID;
    }

    public String getBoardIdx() { 1 usage
        return boardID;
    }

    public void setBoardID(String boardID) { no usages
        this.boardID = boardID;
    }
}

```

페이징 객체화 : 페이징은 게시물 및 댓글등 여러곳에 쓰일 수 있기 때문에 **PagingData**라는 객체를 만들어 해당 객체 안에 페이징 관련 데이터가 담길 수 있도록 하였습니다.

4-a. 답변형 게시판 예시 그림

게시글 1
게시글 1-1 (게시글 1에 대한 답변글1)
게시글 1-1-1 (게시글 1-1에 대한 답변글1)
게시글 1-1-2 (게시글 1-1에 대한 답변글2)
게시글 1-2 (게시글 1에 대한 답변글2)
게시글 1-3 (게시글 1에 대한 답변글3)
게시글 2
게시글 2-1 (게시글 2에 대한 답변글1)
게시글 2-2 (게시글 2에 대한 답변글2)

답변형 게시판 예시그림입니다.

4-b. 추상화 및 부분 구조 분석

구조 파악 및 설명 : 가장 먼저 고려해야 할 점이 있는데, 바로 일반게시판과 답변형게시판과의 차이점입니다. 이 둘의 차이점은 바로 게시글의 종속성 및 독립성 즉, 존재과정에 차이점이 있습니다.

일반게시글을 생각해 보면, 게시글이 존재하기 위한 조건에 대해 생각을 했을 때, 그냥 글쓰기를 하면 바로 게시글이 존재하게 되며 각 게시글은 서로 독립적입니다. 즉, 일반게시판은 게시글 존재과정이 독립적입니다.

그럼 답변형 게시판은 어떤가?라고 생각했을 때 답변형 게시판도 일반게시판처럼 게시글 존재과정이 독립적일수 있습니다.

하지만 답글이 달리기 때문에 이 답글이 독립적인지 종속적인지 판단할 필요가 있습니다.

그러면 한 번 판단을 해보는 것입니다. 답글이 달리기 위한 조건에 대해 생각을 할 때, 답글이 답글을 달 대상(게시글)이 없으면 존재할 수가 없습니다. 따라서, 답글은 답글을 달 대상이 필수이며 이는 종속적이라고 말할 수 있습니다.

이러한 종속적 관계를 간단히 말하면 부모-자식 관계라 할 수 있습니다.

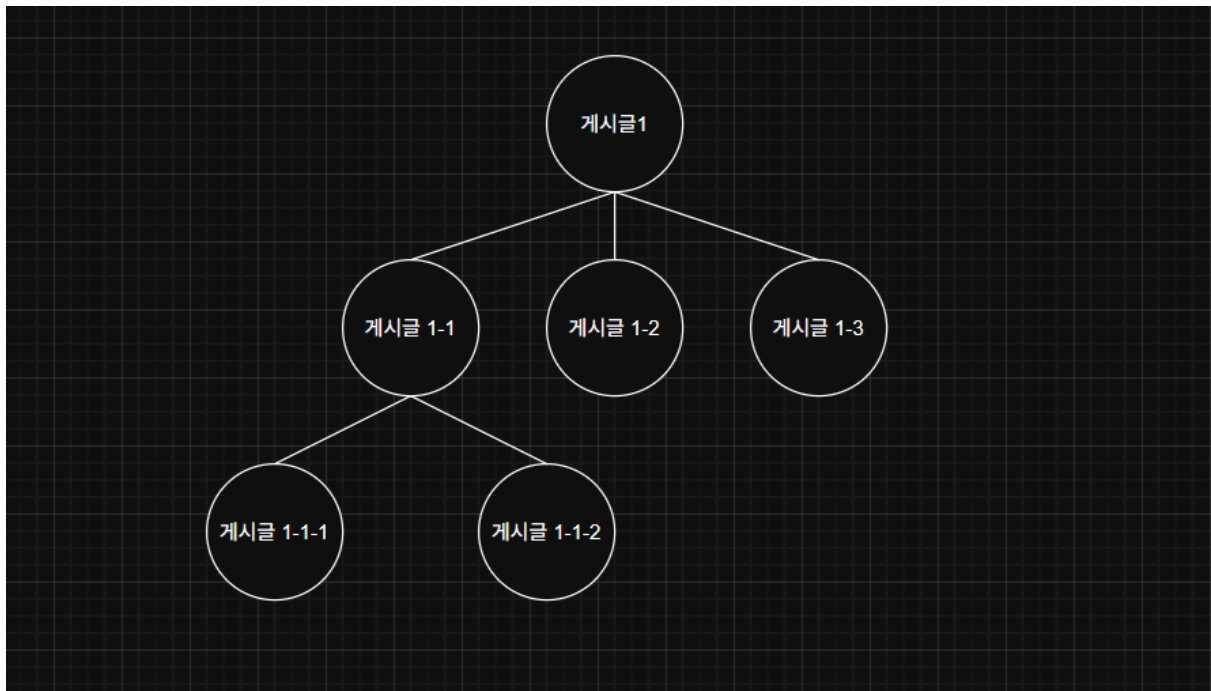
추상화 및 단순화 : 구조에 대한 파악이 끝났으니 해당 구조를 어떻게 다뤄야하는지 생각을 할 필요가 있습니다. 위의 게시판 구조는 일단 복잡합니다.

어떤 복잡한 문제나 상황에 대해 단순하게 만들 수 있다면 그건 더할 나위 없이 좋은 방법입니다. 먼저 해당 구조의 요소들에 대해 간단한 요소로 추상화를 진행하고 그런 다음, 이러한 요소들을 다시 연결시켜 주면 단순화가 될 거라 생각했습니다.

그럼 한번 이러한 생각을 바탕으로 해당구조를 다음과 같이 볼 수 있습니다.

- 각 게시글 : 노드
- 답변 관계 : 간선 표현

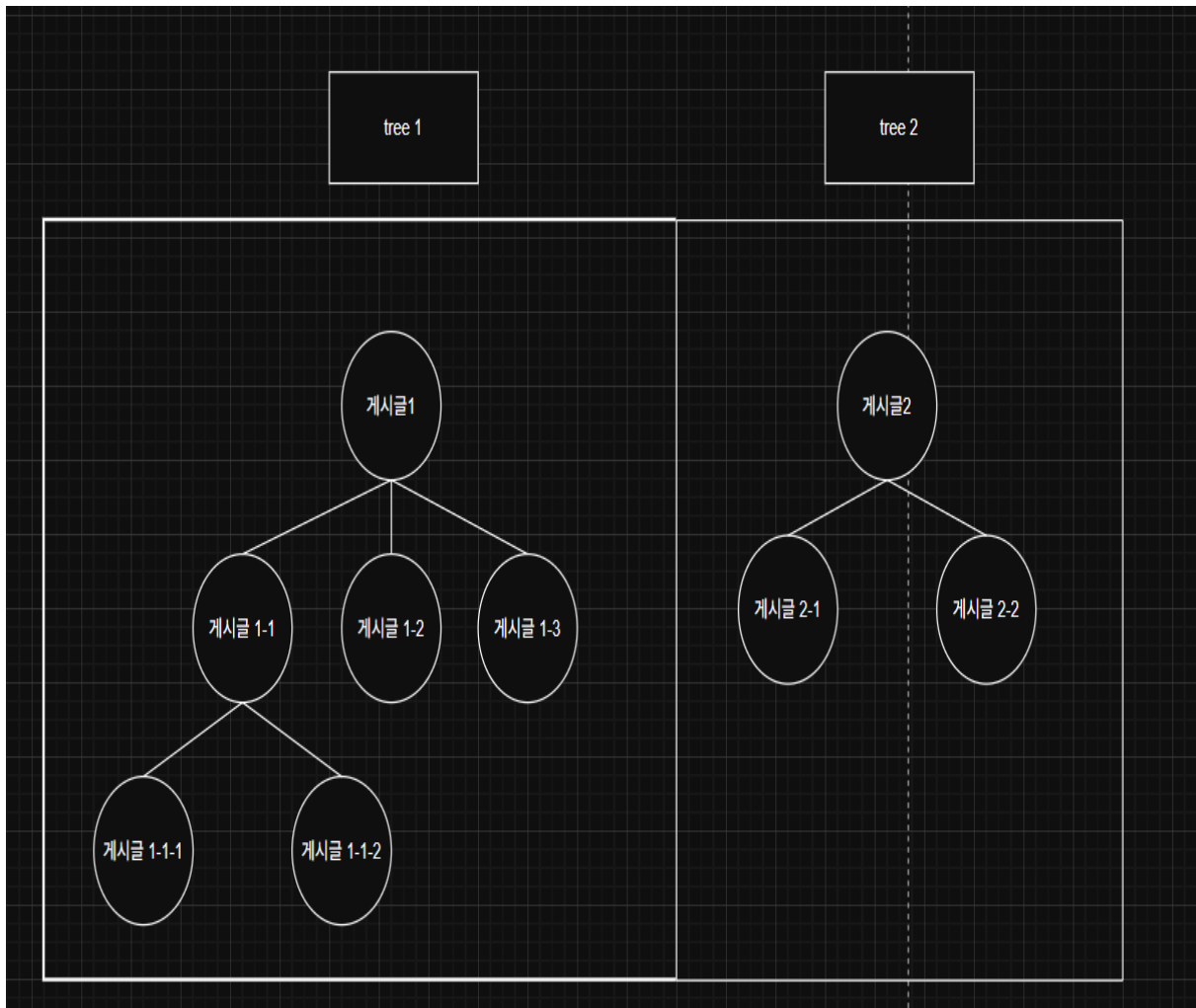
우선 게시글 **2**와 관련된 노드들을 제외한 게시글**1**과 관련된 노드들만으로 이루어진 구조를 표현하자면 다음과 같습니다.



이와 같은 구조를 보니 **트리구조**와 굉장히 비슷하다 생각하였습니다.
그리고 탐색과정은 **깊이 우선 탐색(DFS)** 입니다.

4-c. 구조 확대 및 탐색 방법

구조 확대 : 아직 다 끝난게 아닙니다. 게시글 **2**가 남아있습니다. 게시글 **2**를 포함한 전체 구조를 보면 다음과 같습니다.



제가 게시글 1 사진과 전체 사진을 구분해서 보여드린 이유는 다음과 같습니다.

위 사진에서 볼 수 있듯이, 각 트리의 루트 노드는 독립적, 루트 노드를 제외한 나머지 노드들은 종속적이며 한 트리의 탐색이 끝나면 다음 트리를 탐색하는 구조입니다.

탐색 방법 : 탐색처리는 서버에서 하는게 아니라 **DBMS**에서 하는게 좋습니다. 왜냐하면, 서버에서 하게되면 데이터 연동이 증가할 뿐만 아니라 서버의 작업시간도 늘어나기 때문에 **DBMS**에서 처리하는 게 좋다 판단이 됩니다.

DBMS에서 탐색 방법은 다양합니다. 기본적인 **DML**문을 사용해 탐색을 해도 좋고, **PL/SQL**등을 사용해서 탐색을 해도 됩니다.

하지만 이런 탐색방법들은 복잡하고 그다지 효율적이지 못하다고 생각을 했습니다. 그럼 어떻게 하면 되느냐하면, 운이 좋게도 계층형 쿼리를 사용하면 됩니다. 계층형 쿼리가 위와 같이 탐색을 하기 때문입니다.

한 가지 주의할 점은, 테이블에 부모PK 컬럼이 추가되어야 탐색을 할 수 있습니다. 그리고, 자기 자신을 참조 하는 것이므로 **1:N** 자기참조 비식별 관계로 설정해 놓는 편이 좋다 판단됩니다.

***비식별 관계인 이유** : 식별관계로 놓으면 **PK**가 **null**이 되지 않아야 하는데 위의 구조를 보면 게시글이 독립적으로 존재할 수가 있으므로 부모**PK** 컬럼이 **null**이 될 수 있기 때문입니다.

5-a. 게시판 종류 및 설명

설명 : 총 다섯가지의 게시판이 있습니다. 그리고 삭제표시 되지 않은 게시판만 보여줍니다.




- **질문게시판** : 의사에게 질문을 할 수 있는 게시판입니다. 답글 및 답글의 답글등이 구현되어 있으며 의사회원은 질문을 할 수 없고 답변만 할 수 있습니다.
- **자유게시판** : 자유롭게 게시글을 이용할 수 있는 게시판 입니다. 중첩댓글은 없고 댓글만 있습니다.
- **고객센터** : 불량유저등이나 불만사항을 접수할 수 있는 게시판입니다. 운영자만 답글을 달 수 있습니다.
- **후기 게시판** : 후기를 남길 수 있는 게시판입니다. 추천하기 기능이 추가되어 있습니다. 중첩댓글은 없고 댓글만 있습니다.
- **의료정보 게시판** : 의료정보를 게시할 수 있는 게시판입니다.

5-b. 게시판 **CRUD**

776	frbNotice1	관리자	2025-01-23	12
807	게시글작성	nickName1	2025-02-13	97
806	게게시	nickName1	2025-02-13	1
805	tjhahsgshg	nickName1	2025-02-12	4
804	sdfhdsfhs	nickName1	2025-02-12	0
802	fhadfhadf	nickName1	2025-02-12	1

- **create** : 게시글 생성 기능입니다. 관리자는 글쓰면 공지사항이 되게했습니다(관리자글을 **SQL**에서 **UNION ALL**을 통해 위쪽으로 배치 나머지는 아래로 배치해 구현했습니다. 기존 게시글에서 관리자 글만 뺀것이니 **PK**는 중복되지 않습니다).
- **read** : 게시글 조회 기능입니다. 게시글 조회시 조회수가 1씩 증가하게 됩니다. 조회에 필요한 게시글 **PK**를 서버로 보내야 합니다. 추천기능이 있어 게시글을 추천할 수 있습니다.
- **update** : 게시글 수정 기능입니다. **read**와 동일하게 게시글 **PK**가 필요하며 추가적으로 유저아이디가 필요합니다. 세션에서 해당 작성자에 대한 검증이 필요하기 때문입니다.
- **delete** : 게시글 삭제 기능입니다. **update**와 기능은 다르나 프로세스가 같아 **update**에 보내는 것과 같은 유형의 데이터가 필요합니다. 실제로는 삭제가 되지 않고 삭제했다는 표시만 됩니다.

5-c. 답변형 게시판(고객센터) **CRUD**

3497	df	nickName1	2025-02-12	1
3495	게시글	nickName1	2025-02-12	0
3493	ssssssssssssssssssssss	nickName1	2025-02-12	16
 3494	dddddddddddddddddd	관리자	2025-02-12	2
3471	csTitle234	nickName1	2025-02-04	7
 3490	csTitle234-2	관리자	2025-02-06	2
 3472	csTitle234-1	관리자	2025-02-04	3
3470	csTitle233	nickName1	2025-02-04	0

- **create** : 답변형 게시판같은 경우에는 계층형 쿼리를 사용하기 위해 답글을 달 대상인 부모 게시글 PK가 필요합니다. 운영자만 답글을 달 수 있습니다.
- **read, update** : 일반적인 게시글 **read, update**와 동일합니다. 답변형 게시글을 포함한 게시판 이지만, 읽거나 수정하는데는 일반적인 게시판 글과 동일하기 때문입니다.
- **delete** : **delete**연산같은 경우에는 자식노드(답글)가 있는 부모 노드를 삭제하게 되면 자식노드(답글)이 엉뚱한 게시글의 답글이 되므로 부모 노드를 삭제 하면 자식 노드(답글)가 삭제되게 하였습니다. 실제로는 삭제가 되지 않고 삭제되었다 표시만 됩니다.

5-d. 게시글 검색

"1" 으로 검색하신 결과

게시글아이디	제목	작성자	작성일자	조회수
805	tjahsgshg	nickName1	2025-02-12	4
785	forTestTitle1	nickName2	2025-01-29	442
781	gk1	nickName1	2025-01-26	13
778	1	nickName3	2025-01-26	1
767	frbTitle231	nickName1	2025-01-18	0
757	frbTitle221	nickName1	2025-01-18	0
755	frbTitle219	nickName1	2025-01-18	0
754	frbTitle218	nickName1	2025-01-18	0

설명 : ERD를 보시면 아시겠지만, 테이블이 서로 나뉘어져 있습니다. 페이징을 여태 한 테이블을 기준으로 해왔는데 전체 테이블을 대상으로 한다면 상황이 막막하기만 하지만 여러 테이블을 합쳐 한 개의 테이블로 다룬다면 페이징이 되지 않을까 생각을 했습니다.

하나의 테이블로 다루는 **SQL**문을 저는 **UNION ALL** 연산을 이용했습니다. **UNION ALL** 은 중복허용 및 공통컬럼필요 라는 특징을 가지고 있는데, 각 게시판 테이블은 구조가 비슷하니 공통컬럼(게시글 아이디, 게시글 내용, 방문수, 작성일자등)만 가져왔습니다. 공통컬럼 가져와서 위와같이 표시하기 위함입니다.

하지만 **UNION ALL**을 쓰면 **PK**가 중복되는 경우가 생기니 **PK중복**에 대한 설명은 다음과 같습니다.

각 게시판 **PK**가 중복이 되는 경우에는, 각 테이블에 카테고리라는 컬럼을 추가해 해결했습니다. 그리고 가져오는 공통컬럼에 카테고리 컬럼을 추가하였습니다. 서버는 클라이언트로부터 **PK**와 카테고리를 받고 카테고리를 통해 어느 테이블인지, **PK**를 통해 테이블의 어느 데이터인지를 식별할 수 있습니다.

5-e. 댓글 및 대댓글(중첩페이징)

The image shows a web application interface with two identical form panels. Each panel contains the following elements:

- Header: `idName2` and a button labeled `새`.
- Yellow bar: `답변2` and `2025-03-29`.
- Buttons: `댓글보기` and `댓글작성하기`.
- Text input: `부모 0000`.
- Sub-header: `idName1/2025-03-29`.
- Buttons: `답` and `새`.
- Text input: `2-3`.
- Buttons: `1` and `이동`.

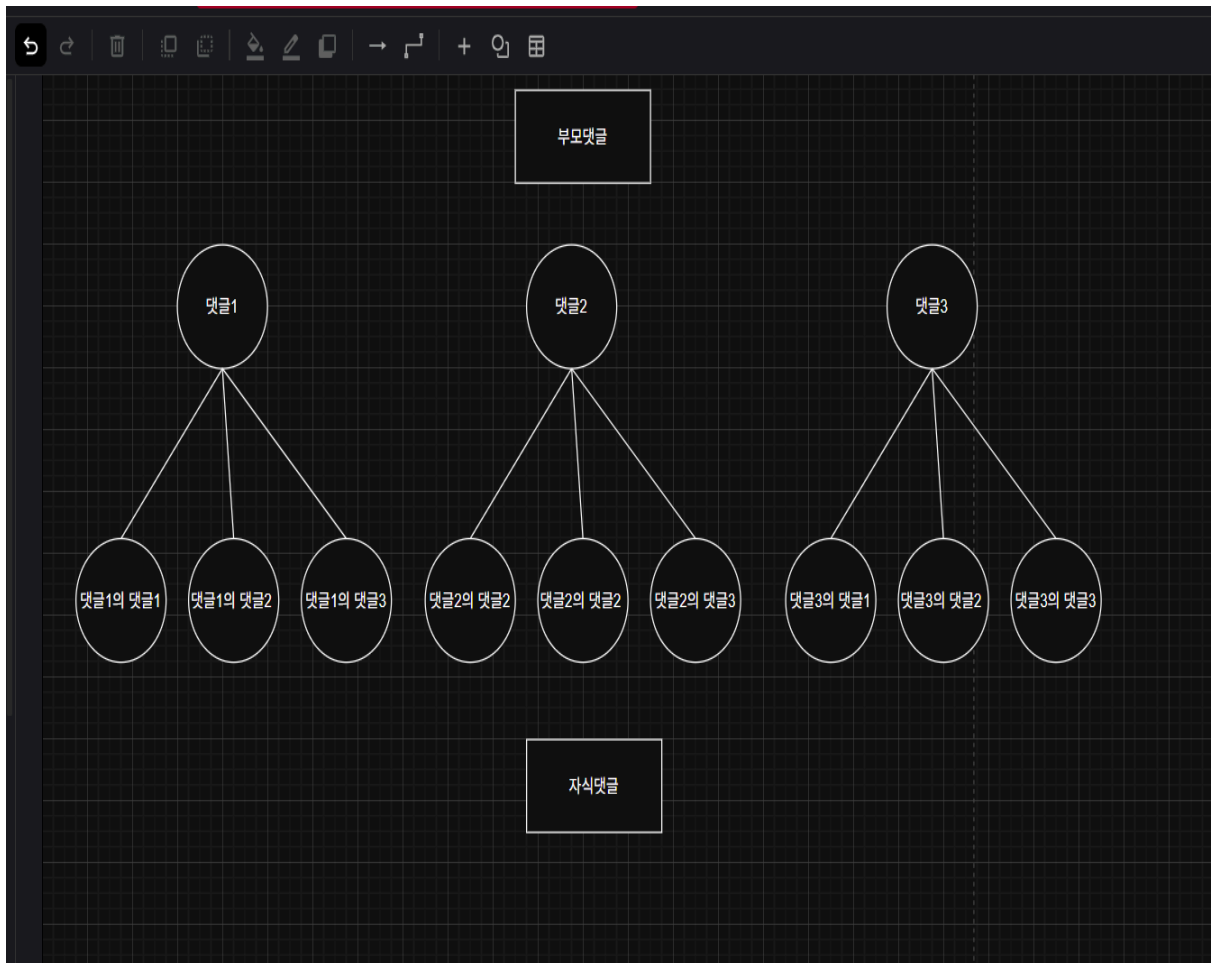
The bottom of the page features a `검색해보기` input and an `이동` button.

구현계기 : 팀원 한 분이 댓글에 댓글을 달 수 있는 시스템을 만들 수 있는지에 대해 요청을 하였습니다. 처음 시도해 보는 것이기도 하고 만들 수 있을지 확신이 들지는 않았지만 한 번 만들기로 도전하였고 해당 기능을 구현하였습니다.

설명 : 페이지안에 페이지가 존재하는 구조입니다. 일반적인 페이지는 부모**PK**가 필요가 없으나 중첩 페이지는 부모**PK**등이 필수로 필요합니다.

왜냐하면 자식 데이터가 부모 데이터 안에 포함되기 때문입니다.

예시 그림 :



클라이언트 구현 과정 설명 : 우선 게시글에 대한 부모 댓글들을 **API** 요청을 하여 로딩을 합니다. 부모 댓글들의 로딩이 끝난 뒤에 요청을 해야 합니다. 왜냐하면, 자식 댓글들을 요청하려면 부모 댓글 데이터가 먼저 도착해야하기 때문입니다. 부모 댓글 데이터가 도착하면 해당 데이터(**PK 포함**)를 배열로 저장합니다. 그 다음에 해당 배열을 **FOR**문을 통해 순서대로 요청을 하고 받은 자식 데이터들을 부모 데이터 안에 넣어주면 됩니다.

서버 및 DB 구현 과정 설명 : 서버랑 **DB**쪽은 패러미터 받은걸로 기존의 페이징 처리를 하면 됩니다. 단, 기존의 페이징 처리에 당연히 해당 부모댓글에 대한 페이징 처리를 해야하기 때문에 **WHERE** 부모 댓글 **id = #클라이언트로부터 받은 패러미터#** 절을 추가해야 합니다.

6-a. 내가 쓴 글 확인

마이페이지

회원정보 수정

내가 쓴 게시글

내가 쓴 댓글

탈퇴

내가 쓴 게시글

전체

자유게시판	게시글작성	2025-02-13
자유게시판	게게시	2025-02-13
자유게시판	tjahsgshgl	2025-02-12
자유게시판	sdfhdsfhsj	2025-02-12
자유게시판	fhadfhadff	2025-02-12
자유게시판	agasfhfasdfj	2025-02-12
자유게시판	adgadl	2025-02-12
자유게시판	asdasddddddddddddd	2025-02-12
자유게시판	gk1	2025-01-26
자유게시판	하	2025-01-26
자유게시판	안녕하세요	2025-01-21
자유게시판	frbTitle234	2025-01-18
자유게시판	frbTitle233	2025-01-18
자유게시판	frbTitle232	2025-01-18
자유게시판	frbTitle231	2025-01-18

페이지 이동 :

이동

1 2 3 4 5 6 7 8 9 10 다음

설명 : (마감시간에 의해 디자인을 제대로 못한 것에 대해 미리 사죄드립니다....)

- 회원정보 수정 : 회원정보를 수정할 수 있습니다. 유저아이디는 **PK**이므로 수정이 안되게 하고 비밀번호나 전화번호, 이메일 등을 수정할 수 있습니다.
- 내가 쓴 게시글 및 댓글 : 내가 쓴 게시글이나 댓글을 클릭시 해당 게시판으로 이동합니다.

회원정보 수정 구현 : 이전과 같은 비밀번호, 전화번호, 이메일등을 사용하는지를 체크합니다. 전화번호나 이메일 등은 해당 데이터를 사용하는 다른 사람이 없는지를 체크합니다.

게시글 및 댓글이동 구현 : 요청이 마이페이지에서 왔는지 아니면 일반게시판이나 검색에서 왔는지에 대한 판별이 필요합니다. 이에 대해 자세한 설명을 하자면 다음과 같습니다.

우선 일반게시판을 통해 오든 검색에서오든 마이페이지에서 오든 보는 화면은 같습니다. 다만, 마이페이지일 경우에는 다른 기능을 하게 만들어야 합니다.

마이페이지에서 해당 게시글이나 댓글을 누르면 스위치가 동작한다는 표시(**ON**)를 브라우저 세션에 해놓고, 브라우저(**JSP**)에서 데이터를 로딩할 때, 해당 **ON**표시를 체크를 합니다.

ON표시가 없으면 일반적인 루트(게시글 및 검색)를 통해 온것이고 **ON**표시가 있으면 마이페이지로부터 온 것이라 할 수 있습니다.

ON표시를 확인했으면 해당 서버에서 받은 페이지를 가지고 데이터 로딩이 끝날 때, 해당 페이지를 클릭하는 기능을 넣었습니다.

이렇게 추가적인 클릭기능을 넣은이유는 기존의 기능은 그대로 유지하되, 해당 기능만 추가하여 사용하기 위해 위와같이 클릭하는 기능을 넣었습니다.

그다음, 한가지 중요한 처리를 해야하는데, 바로 **ON**표시를 **OFF**표시로 해놓아야 합니다. **OFF**표시로 하지 않으면 브라우저는 모든 경로가 전부 마이페이지로 온 거로 간주하기 때문입니다.

조금 더 이해를 돕기 위해 비슷한 예시를 들어 설명을 하자면, 마치 화장실을 이용하는 과정과 비슷합니다. 화장실을 이용하기 위해서 해당 화장실칸이 잠금이 되지 않았는지(**ON**표시확인) 잠금이 되지 않았으면 잠금장치를 잠그고 볼 일을 보고 난뒤 다시 잠금을 풀고(**OFF**표시) 다시 와도 방금 예시의 프로세스들이 반복되는 행위와 비슷하다고 생각하시면 됩니다.

PK로 페이지를 구하는 연산 방법 : 위의 과정에서 서버는 해당 게시글이나 댓글의 **PK**를 통해 페이지를 구해야 합니다. **ORACLE SQL**에서는 **ROWNUM**을 이용하면 해당 **PK**가 몇 번째에 해당되는지 알 수 있기에 쉽게 알 수 있습니다. 즉, **PK의 순서번호**를 통해 페이지를 구할 수 있습니다.

1페이지당 **15**개 표시 기준 과 **0~14**의 표시 기준으로 설명을 하면 다음과 같습니다.

순서번호	페이지
0 ~ 14	1
15 ~ 29	2
30 ~ 44	3

15개 표시기준을 변수화(**limCol**로 변수화)를 한뒤에 표시를 하면 다음과 같습니다.

순서번호	페이지
limCol * 0 ~ limCol * 1 - 1	1
limCol * 1 ~ limCol * 2 - 1	2
limCol * 2 ~ limCol * 3 - 1	3

각각의 섹션을 **n(>= 0)**이라 하고 이를 좀더 수식화를 하면 다음과 같습니다.

순서번호 = **limCol * n + x, (0 <= x < limCol)**.

순서번호 / **limCol** = **n** 이므로 **n + 1**한게 바로 페이지 번호가 됩니다. 즉, 해당 **PK**에 대한 순서번호를 **limCol**로 나눈 것에 **+1**을 하여 클라이언트로 해당 페이지를 보내면 됩니다.

7-a. 글 및 회원관리

관리자 페이지

회원목록	회원목록
공지사항 관리	전체 ▼
고객센터	
답변관리	
	userId9 nickName9 일반회원 세부정보 정지상태아님 탈퇴
	userId8 nickName8 의사회원 세부정보 정지상태아님 탈퇴
	userId7 nickName7 일반회원 세부정보 정지상태아님 탈퇴
	userId6 nickName6 의사회원 세부정보 정지상태아님 탈퇴
	userId5 nickName5 일반회원 세부정보 정지상태아님 탈퇴
	userId4 nickName4 의사회원 세부정보 정지상태아님 탈퇴
	userId3 nickName3 일반회원 세부정보 정지상태아님 탈퇴
	userId2 nickName2 의사회원 세부정보 정지상태아님 탈퇴
	userId1 nickName1 일반회원 세부정보 정지상태아님 탈퇴
	페이지 이동 : <input type="text"/> 이동 1

설명 : (마감시간에 의해 디자인을 제대로 못한 것에 대해 미리 사죄드립니다....)

- 회원목록
 - 드롭다운식으로 전체, 질문, 자유게시판등으로 보기가 가능합니다.
 - 세부정보 : 해당 유저 정보가 나타납니다.
 - 정지상태관리 : 해당 유저의 정지상태를 결정 -> 보는건 가능하지만 글쓰기등이 제한됩니다.
 - 탈퇴 : 해당 유저를 탈퇴시키는 기능입니다.
- 공지사항 및 고객센터 답변 관리 : 각 게시판의 공지사항 및 고객센터 답변리스트가 표시되며 클릭하면 해당 공지사항으로 이동합니다.

8-a. 추천기능 문제점 및 개선점



문제점 : 일반적으로 추천기능은 **1일 1회**로 제한하는 경우가 대부분입니다. 위와 같이 게시판에 추천수를 넣게 되면 **1일 1회**로 제한을 구현하는데 어려움이 따릅니다.

해결방법 : **ERD**문제입니다. 한유저->여러 게시판 추천이 가능하고, 한 게시판->여러 유저 추천가능하므로 **N:M**이므로 교차테이블을 사용해야 합니다.



8-b. 게시판 관리관점

설명 : 게시판을 관리하는 관점은 본인이 생각하기에는 **2**가지 방법이 있다 판단이 됩니다. 이 **2**가지 관리방법에 아래와 같이 설명하겠습니다.

- 분리해서 관리
 - **장점** : 데이터가 분산되어 있으므로 속도 및 조회가 빠릅니다. 또한 분리 되어있으니 독립된 기능을 만들기도 용이합니다.
 - **단점** : 분리되어 있으니 통합된 기능을 개발하거나 관리하기가 까다롭습니다.
- 통합해서 관리
 - **장점** : 데이터를 한곳에서 관리 할 수 있으니 관리가 용이합니다.
 - **단점** : 데이터가 한곳에 몰아 있으므로 속도가 느릴 수 밖에 없습니다. 왜냐하면 데이터가 **100**개 있는 곳에서 특정 데이터를 찾는것보다 데이터가 백만개 있는 곳에서 특정 데이터를 찾는 과정이 더 복잡하기 때문입니다.

통합관리 결정 이유 : 게시판 같은 경우에는 계속해서 늘어날 수 있는 여지가 있습니다. 이를 계속 분리된 테이블로서 관리하게되면 **ERD**가 복잡해질 가능성이 있을 뿐만 아니라 통합된 기능을 제공할 시에 쉽지 않을 거라 판단이 되었습니다. 그리고 통합된 곳에서 무엇인가를 빼는 작업이 분리된 걸 하나로 통합하는 과정보다 쉽다고 판단해서 이렇게 결정하였습니다.

8-c. 선택한 관리 관점 ERD 구현

ERD 설계 : <https://www.erdcloud.com/d/5Fa2MYfS9Dxat87RY>

ERD 설명 :

- **게시판** : 게시판 분류 코드 테이블입니다. 어떠한 게시글의 종류를 판별하는 테이블 입니다.
- **운영자** : 기존에는 유저와 운영자모두 유저테이블에서 관리하였지만 보안 및 관리 측면 고려하여 유저테이블과 분리했습니다.
- **게시글 내용**
 - **내용 분리 이유** : 확장성 고려 및 글 및 **Multipart** 순서 고려하여 만들었습니다. 이렇게 만든 이유는 **ORACLE SQL**에서는 **VARCHAR2(4000)**이 최대이기 때문에 이 사이즈 안에 여러 문장들을 넣기에 상당히 부족하다 생각이 들었고,

게다가 게시물 내용에 순서를 고려한 멀티미디어 파일을 표시하기 위한 표시문장 또한 넣을 필요가 있다 생각이 들어 분리하였습니다.

- **게시글 삭제** : 게시물 삭제 분류 코드 테이블입니다. 삭제된 이유를 관리하기 위해 분리했습니다
- **게시글 댓글**
 - **게시글 답변** : 중첩레벨이 같습니다.(부모와 평행 존재)
 - **게시글 중첩** : 중첩레벨이 다릅니다.(부모안에 존재)
 - **중첩레벨** : **N**중첩을 하기 위해 여러 부모 컬럼이 필요하지만 여러 컬럼을 추가하면 복잡합니다. 그래서 직계관련 부모 아이디 컬럼만 추가했습니다.
그러면 다음과 같은 레벨**1**과 레벨**2**의 차이인지 레벨 **3**과 레벨**4**의 차이인지 알 수가 없는데, 이 레벨 차이를 알기위해 중첩레벨 컬럼 추가하였습니다. 이렇게 컬럼이 추가 되면 쉽게 파악할 수 있을거라 생각이 들었습니다.

이렇게 만든 이유 : 앞서 말했다시피 관리하기 쉽게 만들었습니다.
기존의 게시판 같은 경우에는 게시판을 추가하려면 또 다른 테이블을 만들거나 제약조건같은걸 설정해야합니다.

하지만 이렇게 통합적으로 관리하게 되면 게시판이 추가 되어도 게시판 테이블에 인스턴스만 추가하면 되어 관리하기가 쉽지 않을까 생각을 했습니다.

게시글 댓글 설명 : 기존의 댓글의 댓글 같은 경우에는 테이블을 하나 추가해서 기능을 추가하였는데 여러 댓글안에 계속 해서 댓글들이 달릴 수 있습니다. 물론 이렇게 까지 하는건 약간좀 무리가 있지만 그래도 확장성을 고려하여 테이블 **1**개로도 충분히 동작하도록 만들었습니다.

코드 : 테이블 생성 및 데이터 삽입은 **init integrated board.sql**
파일참고 해주시면 됩니다. 위에서부터 순서대로 실행하면 됩니다. 조회 코드는 **sql for integrated board.sql** 파일참고 하시면 됩니다.

8-d. 데이터(댓글)로딩 설명 및 추가 고려점

설명 : sql for integrated board.sql 파일 참고하시면 이해하시기 편합니다. 댓글로딩을 구현한 쿼리입니다. 해당 게시글에 대한 모든 중첩 댓글들을 로딩합니다.

- **load nest level 1 comments** : 중첩레벨 1 댓글로딩 가져오는 쿼리입니다.
- **load nest level 2 comments** : 우선 안에 서브쿼리가 있는데 이 서브쿼리는 데이터를 **WHERE** 절로 먼저 필터링 하였습니다. 그런 다음 계층 필터링을 하였습니다.
- **load nest level 2 comments by using list** : 위의 레벨 2를 리스트로 가져오는 상황이 되면 이렇게 쓰는데 좋지 않을까 해서 추가하였습니다.

결과 : load nest level 2 comments by using list의 결과 화면입니다.

	P.CMT_ID	POST_ID	USER_ID	P.CMT_RE_P_ID	P.CMT_NEST_P_ID	P.CMT_NEST_LVL	P.CMT_CONTENT	P.CMT_CREATE_DATE	ROWNUM
1	16	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글1	25/10/25	1
2	17	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글2	25/10/25	2
3	18	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글3	25/10/25	3
4	19	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글4	25/10/25	4
5	20	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글5	25/10/25	5
6	31	5	5userId1	20	5	2	2-게시판2-게시글2-댓글5-댓글5-답변1	25/10/25	6
7	32	5	5userId1	20	5	2	2-게시판2-게시글2-댓글5-댓글5-답변2	25/10/25	7
8	33	5	5userId1	20	5	2	2-게시판2-게시글2-댓글5-댓글5-답변3	25/10/25	8
9	34	5	5userId1	20	5	2	2-게시판2-게시글2-댓글5-댓글5-답변4	25/10/25	9
10	35	5	5userId1	20	5	2	2-게시판2-게시글2-댓글5-댓글5-답변5	25/10/25	10
11	21	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글6	25/10/25	11
12	22	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글7	25/10/25	12
13	23	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글8	25/10/25	13
14	24	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글9	25/10/25	14
15	25	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글10	25/10/25	15
16	26	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글11	25/10/25	16
17	27	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글12	25/10/25	17
18	28	5	5userId2	(null)	5	2	2-게시판2-게시글2-댓글5-댓글13	25/10/25	18

9. 프로젝트 느낀점

느낀점 : 처음보는 사람들과 같이 수업을 듣고 초반에는 서먹서먹 했지만, 같이 프로젝트를 진행하면서 대화도 자주 하면서 같이 회의도 하고 서로 만든 것에 버그나 오류가 생기면 같이 고치기도 하였습니다. 이렇게 초반 팀 프로젝트를 하면서 힘들었지만 재미나 성취감을 느끼게 가장 좋았습니다.