목차

1. 추천기능 문제점 및 개선점

유저 — users

유저아이디	user_id	Domain	varchar2(20)	NOT NULL	Default value	Comment
의사면허코드	dr_lic_code	Domain	varchar2(20)	NULL	Default value	Comment
비밀번호	user_pw	Domain	varchar2(20)	NOT NULL	Default value	Comment
이름	user_name	Domain	varchar2(60)	NOT NULL	Default value	Comment
전화번호	user_tel	Domain	varchar2(20)	NOT NULL	Default value	Comment
주소	user_addr	Domain	varchar2(300)	NOT NULL	Default value	Comment
삭제여부	user_del	Domain	char(1)	NOT NULL	'n'	Comment
닉네임	user_nickname	Domain	varchar2(60)	NOT NULL	Default value	Comment
이메일	user_email	Domain	varchar2(30)	NOT NULL	Default value	Comment
생성일자	user_reg_date	Domain	date	NOT NULL	Default value	Comment
활동정지여부	user_activity	Domain	char(1)	NOT NULL	'n'	Comment
프로필이미지	user_image	Domain	varchar2(1000)	NULL	Default value	Comment

후기게시판 — reviewBoard

게시글아이디	rvb_id	Domain	number	NOT NULL	Default value	시퀀스 rvb_seq
유저아이디	user_id	Domain	varchar2(20)	NOT NULL	Default value	Comment
게시글제목	rvb_title	Domain	varchar2(60)	NOT NULL	Default value	Comment
게시글내용	rvb_content	Domain	varchar2(2400)	NOT NULL	Default value	Comment
게시글생성일자	rvb_reg_date	Domain	date	NOT NULL	Default value	Comment
조회수	rvb_view_cnt	Domain	number	NOT NULL	0	Comment
추천수	rvb_like_cnt	Domain	number	NOT NULL	0	Comment
삭제여부	rvb_del	Domain	char(1)	NOT NULL	'n'	Comment
카테고리	rvb_category	Domain	varchar2(20)	NOT NULL	'rvb'	Comment

문제점 : 일반적으로 추천기능은 **1**일 **1**회로 제한하는 경우가 대부분.

해결방법 : **ERD** 문제임. 한유저**->**여러 게시판 추천가능, 한 게시판**->**여러 유저 추천가능 이므로 **N:M** 임.

2-a. 게시판 관리관점

설명 : 게시판을 관리하는 관점은 본인이 생각하기에는 **2**가지 방법이 있다 판단됨. 이 **2**가지 관리방법에 아래와 같이 설명함.

- 분리해서 관리
    - 장점 : 데이터가 분산되어 있으므로 속도 및 조회빠름. 독립된 기능용이.
    - 단점 : 관리 어려움.
- 통합해서 관리
    - 장점 : 관리 용이함.
    - 단점 : 데이터가 한곳에 몰아 있으므로 속도 느림. 독립기능 설계 불편함.

통합관리 결정 이유 : 게시판 같은 경우에는 계속해서 늘어날 수 있는 여지가 있음. 이를 계속 분리된 테이블로서 관리하게되면 **ERD**가 복잡해질 가능성이 있을 뿐만 아니라 통합된 기능을 제공할 시에 쉽지 않을 거라 판단이 됨.

그리고 분해는 조립의 역순이라는 말이 있다. 분해->조립, 조립->분해 이런 관계가 형성이 된다는 말인데 각 순서과정이 항상 동일하다고는 말할 수 없다.

통합된 테이블에서 특정 데이터 그룹을 분리하는 것은 개인적인 생각으로 쉽다고 판단. 하지만 서로 다른 구조의 데이터를 합치는 것은 구조 및 데이터 충돌같은 변수가 많기 때문에 이러한 결정을 함.

2-a. 선택한 관리 관점 구현

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

ERD 설명 :

- 게시판 : 게시판 분류 코드 테이블
- 운영자 : 보안 및 관리 측면 고려하여 유저테이블과 분리.
- 게시글 내용
    - 내용 분리 이유 : 확장성 고려 및 글 및 **Multipart** 순서 고려.
- 게시글 삭제 : 게시글 삭제 분류 코드 테이블.
- 게시글 댓글
    - 게시글 답변 : 중첩레벨이 같음.(부모와 평행 존재)
    - 게시글 중첩 : 중첩레벨이 다름.(부모안에 존재)
    - 중첩레벨 : **N**중첩을 하기 위해 여러 부모 컬럼이 필요하지만 여러 컬럼 추가하면 복잡함. 그래서 직계관련 부모 아이디

컬럼만 추가함. 그러면 다음과 같은 레벨**1**과 레벨**2**의 차이인지 레벨 **3**과 레벨**4**의 차이인지 알 수가 없음. 이 레벨 차이를 알기위해 중첩레벨 컬럼 추가. 이렇게 컬럼이 추가 되면 클라이언트가 쉽게 파악할 수 있을거라 생각함.

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

하지만 이렇게 통합적으로 관리하게 되면 게시판이 추가 되어도 게시판 테이블에 인스턴스만 추가하면 됨.

ERD 추가 고려점 : 게시글과 댓글은 전체적으로 보면 서로 비슷한 구조임. 따라서 게시글 내용, 파일, 삭제 테이블 또한 댓글과 제약조건 추가해야함.

그리고 **Multipart,** 삭제와 같은 테이블도 게시판에서 게시판코드를 통해 게시글을 분리했듯이 이와 같은 분류가 필요함.

즉, 통합 관리 테이블을 만들어 분류해서 관리하는게 좋다고 판단.
하지만 세세한 건 생략함.

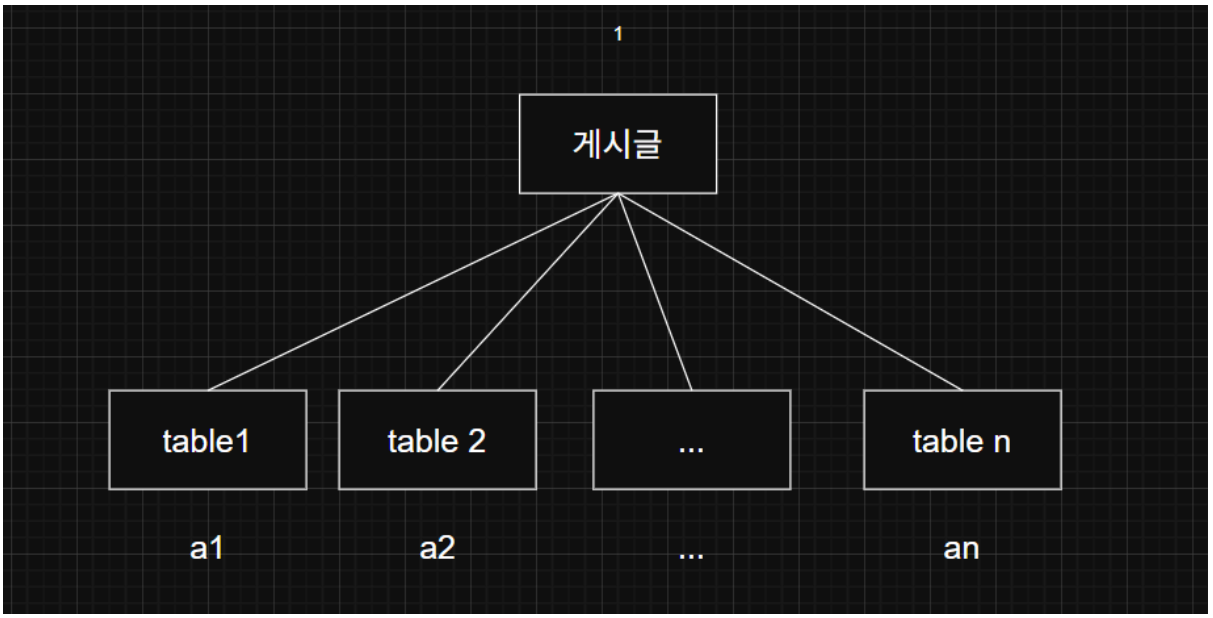
SQL 추가 고려점 : 하나의 테이블 **CRUD** 같은 경우에는 트랜잭션 및 비동기 처리에 그리 신경 쓸 일이 없지만, 테이블이 분리되면 비동기처리에 신경 쓸 여지가 생김.

예를 들어, **INSERT** 문 같은경우에는 테이블이 분리되어 있으므로 각각의 테이블에 오류없이 **INSERT** 할 수 있도록 하나의 트랜잭션으로 묶어서 처리하거나 이러한 트랜잭션 격리 수준을 고려해야함.

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

2-c. 조인 데이터 한번 요청 가능성

설명 : 게시글 데이터와 관련된 모든 조인 데이터를 한번에 가져오는게 가능한가? 가능하다. 하지만 비효율적이라 판단함.



위의 그림은 게시글의 자식 테이블이다. 단순 비용을 정리하면 다음과 같다. (본인이 생각하는 비용이므로 틀릴 수 있음. 환경은 동일하다 가정.)

- 데이터 한번 요청 비용 = **a1 * a2 … * an** (비용 **A**)
- 데이터 분리 요청 비용 = **a1 + a2 … + an** (비용 **B**)

비용 **A**와 비용 **B** 중 어느 비용이 더 작은가? 우위를 가릴 수는 없다. 관련 테이블의 데이터가 **1**개 이하(**0**이면 **0**이지만 **0**이어도 없다는 표시를 해야함)이면 비용 **A**의 값은 **1**이다.

하지만 이런 경우는 매우 특수한 경우이다. 우리는 일반적인 경우를 생각해야 한다. 그리고 이렇게 한번에 조인을 하게 되면 데이터가 중복되는 경우가 생김.

직관적으로 두 개의 식을 볼 때, 비용 **B**가 비용 **A**보다 더 값이 작다고 생각된다. 왜냐하면, 비용 **A**는 곱셈 연산이 너무많이 들어가있기 때문이다. 따라서 데이터 요청은 분리요청을 통해 처리하는게 좋다고 판단한다.

2-d. 데이터(댓글)로딩 설명 및 추가 고려점

설명 : **sql for integrated board.sql** 파일 참고바람. 댓글로딩 기준임

- **load nest level 1 comments :** 레벨 **1** 댓글 가져오기.
- **load nest level 2 comments :** 우선 안에 서브쿼리가 있는데 이 서브쿼리는 데이터를 **WHERE** 절로 먼저 필터링 한것임. 데이터 전체를 계층형 쿼리로 만든다음 **WHERE** 절로 필터링 한게 아니라 **WHERE** 절로 먼저 부분 데이터를 필터링 한다음에 계층 필터링을함. 즉, 부분데이터 추출->계층쿼리화.
- **load nest level 2 comments by using list :** 위의 레벨 **2**를 리스트로 가져오는 상황이 되면 이렇게 쓰는게 좋지 않을까 해서 추가.

추가 고려점 : 데이터 로딩을 할 때, 따로 가져오는게 아니라 한번에 가져오면 이 방법이 아마 더 좋은 방법일 것임. 근데 앞서 설명 했듯이 한번에 가져오는건 좋지만 가져오는 비용을 무시해서는 안됨.

2-d. 기능 추가 고려점

설명 : 요즘은 스크롤 페이징기법을 많이 쓰는 것 같다. 그런데 이러한 스크롤 페이징은 페이지 검색기능이 거의 없는데 페이지 검색을 통해 너무많이 뒤로가면 클라이언트에 너무 스크롤 데이터가 너무 쌓여서 일부러 안넣었는지 모르겠다.

하지만, 이러한 단점들을 스크롤 데이터가 특정 기준 만큼 쌓이면 초기화를 한다던지 아니면 헤더파일형식으로 간단히 저장하던지 하면 좋을 것 같다.

그리고 페이지 저장 기능도 추가해 매번 처음부터 보는게 아니라 특정 페이지부터 보는 기능도 추가하면 좋을 것 같음.

3. 데이터 동기화 문제점

설명 : 데이터 동기화를 클라이언트에서 해줘야 할 필요가 있다. 부모 요소에 자식 데이터를 넣으려고 하는데 부모 요소가 없으면 넣을 수가 없으니 말이다.

4. api 문제점

설명 : **api** 요청을 보면 매우 더럽다. **Restful** 하게 할 필요가 보인다.