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 대부분입니다. 위와 같이 게시판에 추천수를 넣게 되면 **1**일 **1**회로 제한을 구현하는데 어려움이 따릅니다.

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

2-a. 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.

2-a. 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 들고,

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

- 게시글 삭제 : 게시글 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 : N중첩을 하기 위해 여러 부모 컬럼이 필요하지만 여러 컬럼을 추가하면 복잡합니다. 그래서 직계관련 부모 아이디 컬럼만 추가했습니다.
    그러면 다음과 같은 레벨1과 레벨2의 차이인지 레벨 3과 레벨4의 차이인지 알 수가 없는데, 이 레벨 차이를 알기위해 중첩레벨 컬럼 추가하였습니다. 이렇게 컬럼이 추가 되면 쉽게 파악할 수 있을거라 생각이 들었습니다.

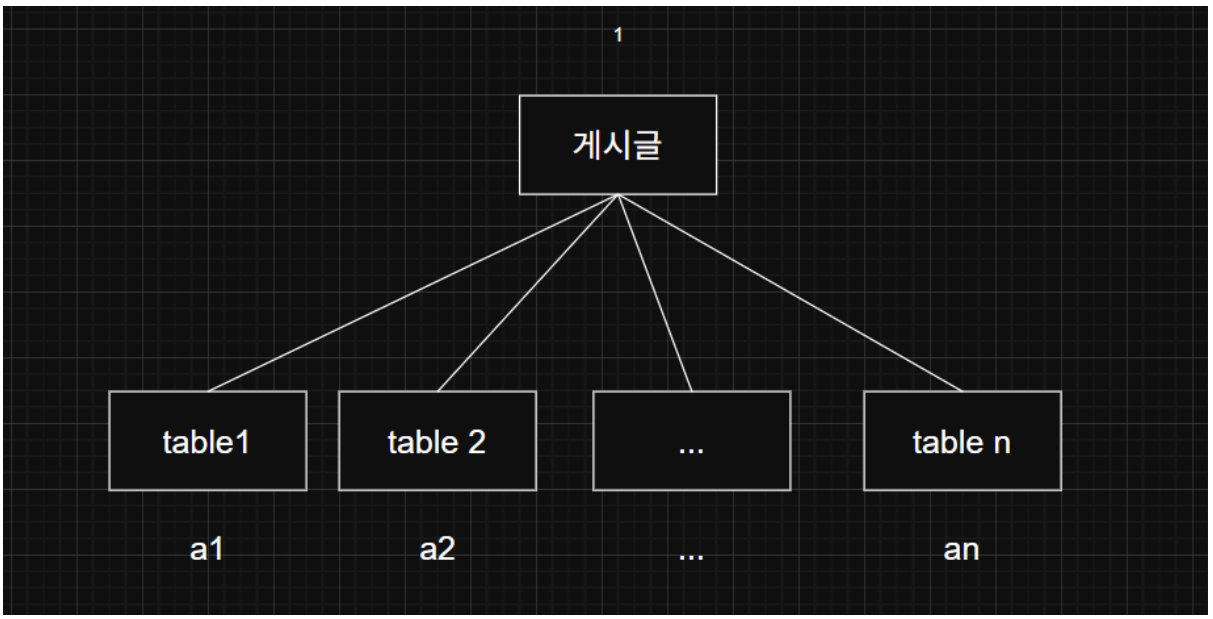
이렇게 만든 이유 : 앞서 말했다시피 관리하기 쉽게 만들었습니다.
기존의 게시판 같은 경우에는 게시판을 추가하려면 또 다른 테이블을 만들거나 제약조건같은걸 설정해야합니다.

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

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.

2-c. 조인 데이터 한번 요청 가능성

설명 : 게시글 데이터와 관련된 모든 조인 데이터를 한번에 가져오는게 가능한가?라고 생각했을 때, 가능은 하지만, 비효율적이라 생각합니다.



위의 그림은 게시글의 자식 테이블입니다. 단순 비용을 정리하면 다음과 같이 생각해 볼 수 있습니다. (본인이 생각하는 비용이므로 틀릴 수 있습니다. 환경은 동일하다 가정했습니다.)

- 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 작은가요**?** 우위를 가릴 수는 없습니다. 관련 테이블의 데이터가 **1**개 이하**(0**이면 **0**이지만 **0**이어도 없다는 표시를 해야합니다**)**이면 비용 **A**의 값은 **1**입니다.

하지만 이런 경우는 매우 특수한 경우이고 일반적인 경우를 생각해야 하지 않나 생각이 들었습니다.

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

2-d. 데이터(댓글)로딩 설명 및 추가 고려점

설명 : **sql for integrated board.sql** 파일 참고하시면 이해하시기 편합니다. 댓글로딩을 구현한 쿼리입니다.

- **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 **2**를 리스트로 가져오는 상황이 되면 이렇게 쓰는게 좋지 않을까 해서 추가하였습니다.

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