

2

Chapter 2. 실전 SQL - 어떤 Query를 작성해야 할까? (1)

 학습 목표

 잠깐 ! 스터디 인증샷은 찍으셨나요? 

 2주차 주제

 2주차 본문

실제로 접할 수 있는 요구 사항

해시태그를 통한 책의 검색

페이징(Paging)

 목록 조회, 이대로 괜찮을까요?

Offset based 페이징

Offset paging의 단점

Cursor based 페이징

 핵심 키워드

 학습 후기

 스터디 진행 방법

 실습 체크리스트

 실습 인증

 미션

 미션 기록 (여기에 해도 되고 위의 미션에서 각 페이지 밑에 간단하게 블록 만들어서 하셔도 됩니다!)



 트러블 슈팅

 참고 자료

 **학습 목표**

1. 1주차 때 예시를 기반으로 여러가지 요구 사항에 대한 SQL 쿼리를 고민한다.
2. paging을 고려하여 쿼리를 작성한다.

 **잠깐 ! 스터디 인증샷은 찍으셨나요?** 

* 스터디리더께서 대표로 매 주차마다 한 장 남겨주시면 좋겠습니다!  

(사진을 저장해서 이미지 임베드를 하셔도 좋고, 복사+붙여넣기해서 넣어주셔도 좋습니다!)

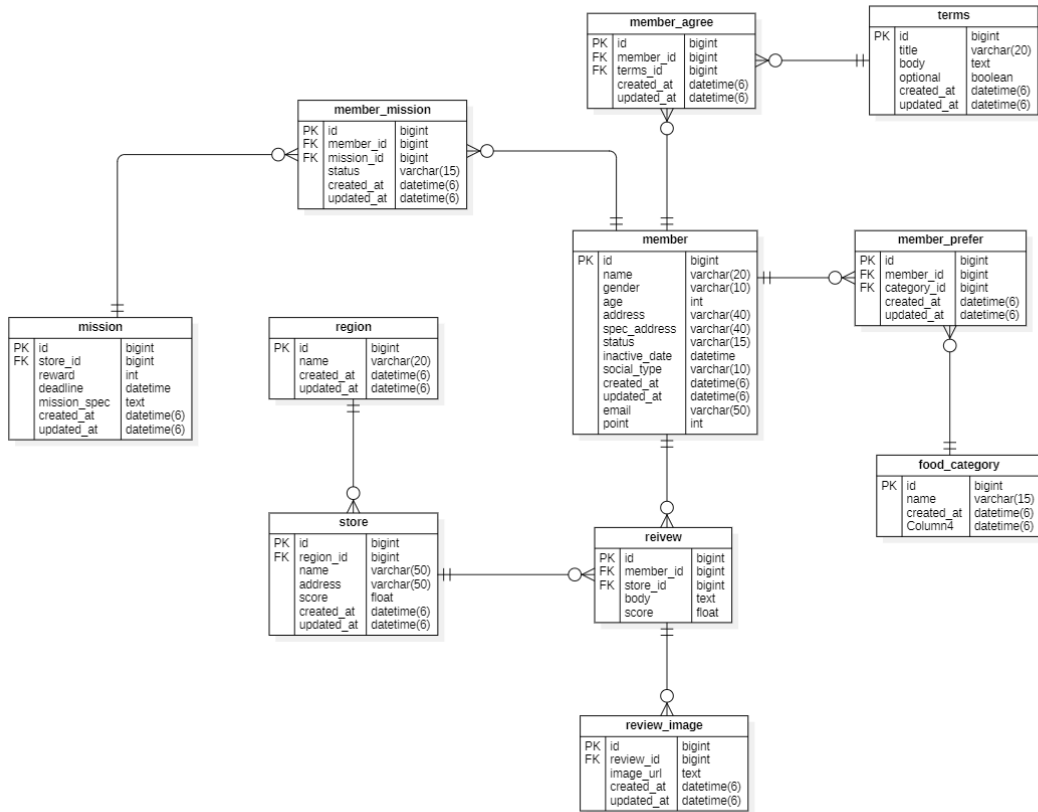


2주차 주제

다들 데이터베이스 설계(1주차 미션)는 잘 하셨나요?

▼ 저 같은 경우는 아래의 형태로 설계를 했습니다. (참고해주세요!)

! 1주차 미션에 포함되지 않는 부분은 설계를 하지 않았습니다!



1주차 미션 ERD



위 ERD 사진 수정사항 안내

- member 테이블의 PK 또한 id여야 하는데 해당 내용이 누락되어 member 테이블의 PK가 phone_num이 아닌, id로 생각해주세요!
- store를 mission이 외래키로 참조하고 있어 1:n 연결선이 필요합니다! (mission:store = n:1)


이번 주차는 몇 가지 요구 사항에 대해 어떻게 쿼리를 만들어야 하는지 같이 고민해보는 주차입니다.

join, subquery는 알고 있다는 전제로 시작합니다!

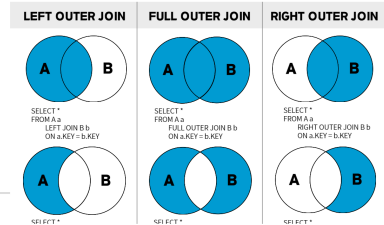
▼ 참고 자료

SQL 기본 문법: JOIN(INNER, OUTER, CROSS, SELF JOIN)

조인은 두 개의 테이블을 서로 묶어서 하나의 결과를 만들어 내는 것을 말한다. INNER JOIN(내부 조인)은 두 테이블을 조인할 때, 두 테이블에 모두 지정한 열의 데이터가 있어야 한다. OUTER JOIN(외부 조인)은 두 테

 <https://hongong.hanbit.co.kr/sql-기본-문법-joininner-outer-cross-self-join/>

OUTER JOIN



[SQL] 테이블 JOIN의 개념과 예제

테이블 JOIN의 개념과 예제


 <https://velog.io/@wijoonwu/JOIN>

[SQL] 테이블 JOIN 

INNER JOIN, OUTER JOIN

[MYSQL]  테이블 조인(JOIN) - 그림으로 알기 쉽게 정리

SQL JOIN JOIN은 데이터베이스 내의 여러 테이블에서 가져온 레코드를 조합하여 하나의 테이블이나 결과 집합으로 표현해 주는, Relation Database 에서 가장 많이 쓰이는 녀석이다. (INNER)

 <https://inpa.tistory.com/entry/MYSQL-테이블-조인-그림으로-알기쉽게-정리>




SQL / MySQL 서브쿼리(SubQuery)


서브쿼리(Subquery) 서브쿼리(Subquery)란 하나의 SQL 문 안에 포함되어 있는 또 다른 SQL문을 말한다. 서브쿼리는 메인쿼리가 서브쿼리를 포함하는 종속적인 관계이다. #메인쿼리 SELECT

 <https://snowple.tistory.com/360>



[MYSQL]  서브쿼리 개념 & 문법 ¹⁰⁰ 정리

서브쿼리(Subquery) 서브쿼리(subquery)란 다른 쿼리 내부에 포함되어 있는 SELETE 문을 의미한다. 서브쿼리를 포함하고 있는 쿼리를 외부쿼리(outer query)라고 부르며, 서브쿼리는 내부

 <https://inpa.tistory.com/entry/MYSQL-서브쿼리-정리>



 mysql 기준으로 진행합니다.

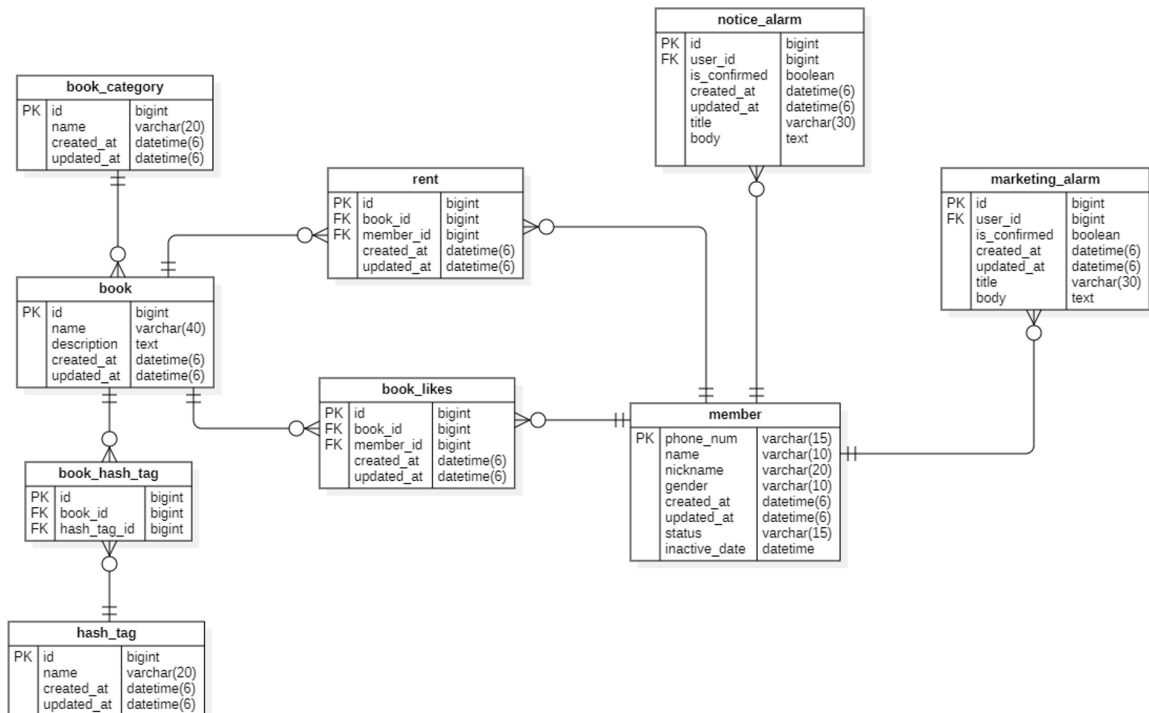
2주차 본문

기존 1주차 워크북 본문서 요구하던 요구 사항,
그리고 전 미션(1주차 미션)에서 요구했던 요구 사항 일부에 대해
어떻게 쿼리를 만드는 것이 좋을지 고민을 해봅시다.

query 역시 Database 설계처럼 **정해진 답이 없고,**

join을 사용할 지 혹은 subquery를 사용할 지 선택하시면 됩니다!

그리고 너무 무리해서 **한번에 거대한 query를 보내기** 보다는 **힘들 경우, 2개로 쪼개서 보내는 것**
도 좋습니다.



1주차 워크북의 ERD

우선 아래의 요구 사항에 대해 어떻게 데이터를 줄지 고민해봅시다.



책이 받은 좋아요 개수를 보여준다.

만약, 책 테이블에 좋아요 개수를 둔다면 아래와 같은 query로 충분하겠죠

```
select likes from book;
```

그런데, 만약 **likes 칼럼없이 집계**를 한다면 아래와 같은 query가 필요합니다

```
select count(*) from book_likes where book_id = {대상 책 아이디};
```

위의 요구 사항에서 추가로 아래와 같은 요구 사항이 생긴다면 어떻게 해야 할까요?



책의 좋아요 갯수를 계산하는데, 내가 차단한 사용자의 좋아요는 집계를 하지 않는다.

| block | | |
|-------|------------|-------------|
| PK | id | bigint |
| FK | owner_id | bigint |
| FK | target_id | bigint |
| | created_at | datetime(6) |
| | updated_at | datetime(6) |

차단 테이블

우선 차단 테이블이 위와 같다고 해봅시다.

그러면 아래와 같은 쿼리가 필요하게 됩니다.

책의 아이디가 3, 내 아이디가 2라고 가정합니다!

```
select count(*) from book_like where book_id = 3
and user_id not in (select target_id from block where owner_id = 2);
```

위의 쿼리문을 아래와 같은 **left join**을 사용하도록 변경이 가능합니다.

```
select count(*)
from book_like as bl
left join block as b on bl.user_id = b.target_id and b.owner_id = 2
where bl.book_id = 3 and b.target_id is null;
```



참고!

저 같은 경우, join 연산보다는 **sub query**가 더 가독성이 좋다고 생각해 sub query를 자주 사용합니다!

원하는 스타일의 query를 선택하여 사용하시면 될 것 같습니다.

그리고 제가 굳이 차단에 대한 요구 사항을 예시로 든 이유는



TIP)

커뮤니티 성격의 app의 경우 신고, 차단 기능이 없을 경우 reject를 받아 런칭이 안됩니다.

따라서 처음에 쿼리를 간단하게 작성했다가 reject를 고려해 모든 쿼리를 수정했던 좋지 않은 경험을 했기에, 여러분들에게 처음부터 아예 신고, 차단을 고려하여 쿼리를 작성 하는 것을 추천 드립니다. 만약 기획이 나오지 않았다면 PM에게 상황을 설명하며 요청을 하시기 바랍니다.

이처럼 이번 주차는 **실제로 접할 수 있는 요구 사항들에 대해 어떻게 접근할지**

몇 가지 예시를 보여 드리겠습니다.

실제로 접할 수 있는 요구 사항

해시태그를 통한 책의 검색

쿼리를 작성 할 때 N : M 관계로 인해 가운데 매핑 테이블이 추가 된 경우는 쉬운 쿼리로 데이터를 가져오기가 힘듭니다.

UMC라는 이름을 가진 해시태그가 붙은 책을 찾도록 해봅시다.

(DataBase에 #을 붙여서 저장하거나 아니면 # 없이 저장하는 것은 프론트 개발자 분과 상의를 해서 결정을 해야 하고 저 같은 경우는 저번 프로젝트에서 후자로 했기에 후자로 예시를 드리겠습니다.)

이 역시 직감적으로 조인 연산 혹은 서브 쿼리가 필요함을 느낄 수 있죠

```
select * from book where id in
  (select book_id from book_hash_tag
    where hash_tag_id = (select id from hash_tag where name = 'UMC' ));
```

이 쿼리는 다시 아래처럼 join 연산을 사용하도록 변경이 가능합니다.

```
select b.*
from book as b
inner join book_hash_tag as bht on b.id = bht.book_id
inner join hash_tag as ht on bht.hash_tag_id = ht.id
where ht.name = 'UMC';
```

자, 이제 책들의 목록을 최신 순으로 조회하는 쿼리를 만들어 봅시다.

간단하게 아래와 같이 만들 수 있겠죠?

편의를 위해 모든 데이터를 다 가져온다고 합시다.

```
select * from book order by created_at desc;
```

자, 이번에는 조금 더 어렵게 좋아요 개수 순으로 목록 조회를 한다고 해봅시다.
book 자체에 likes 칼럼이 없기에 단순한 select로는 조회가 안됩니다.

```
select * from book as b
join (select count(*) as like_count
      from book_likes
      group by book_id) as likes on b.id = likes.book_id
order by likes.like_count desc;
```

이런 형태의 쿼리를 통해 인기 순 정렬이 가능해집니다.

페이징(Paging)

🤔 목록 조회, 이대로 괜찮을까요?

사실 목록 조회는 저렇게만 하면 안됩니다.

예를 들어 책이 100만 권이 있다고 할 때, 저 쿼리를 그대로 사용하게 된다면
엄청난 렉이 발생하게 되겠죠, 한번에 100만 개를 다 가져오게 되니...🤯

따라서 Database 자체에서 끊어서 가져오는 것이 필요하고 이를

paging 이라고 합니다.

Paging에도 사실 **2가지 형태**가 존재합니다.

Offset based 페이징

이는 우리가 자주 봤던 페이징 입니다.

이렇게 직접 페이지 번호를 찾아내어 이동하는 페이지징이죠.

그러면 저런 쿼리는 어떻게 만들까요?

paing 쿼리는 sql마다 상이하며 **UMC는 다시 한 번 말하지만 mysql 기반으로 합니다!**

```
select *  
from book  
order by likes desc  
limit 10 offset 0;
```

위와 같이 limit를 통해 한 페이지에서 보여줄 데이터의 개수를 정하고
offset으로 몇 개를 건너뛴지를 정합니다.

그러면 아래와 같이 추론이 가능하죠.

페이지 x번에 대하여 한 페이지에 y개를 보여준다면

```
select *  
from book  
order by likes desc  
limit y offset(x - 1) * y;
```

이렇게 쿼리를 만들 수 있겠죠?

왜 (x - 1)이냐면 보통 1페이지가 첫 페이지이기 때문이죠!

이제 offset paging을 적용해서 위의 목록 조회 query를 만들어 봅시다!

페이지 n번에 대해 한번에 15개씩 보여준다고 합시다.

```
select * from book  
order by created_at desc  
limit 15 offset (n - 1) * 15;
```

당연히 (n - 1) * 15 이렇게 쿼리에 적으면 안되고... 숫자 계산해서 넣어야 합니다..

이제 다음으로 인기 순 정렬에 대한 쿼리를 만들어 봅시다.

```
select * from book as b
join (select count(*) as like_count
      from book_likes
      group by book_id) as likes on b.id = likes.book_id
order by likes.like_count desc
limit 15 offset (n - 1) * 15;
```

Offset paging의 단점

offset paging은 직접 여러 개의 데이터를 넘어가서 가져온다는 느낌이기에 페이지가 뒤로 갈수록 넘어가는 데이터가 많아져 성능 상의 이슈가 있고, 결정적으로 **아래와 같은 문제 상황이 발생할 수 있습니다.**



사용자가 1 페이지에서 2 페이지로 넘어가려는 찰나, 게시글 6개가 추가가 되었다.

이러면 어떻게 될까요??

😬 **분명 2 페이지로 왔는데 1 페이지에서 봤던 게 또 보이네??** 😬

이런 단점을 보완한 페이징 기법이 있습니다.

Cursor based 페이징

cursor paging의 경우 이름에서 유추 할 수 있듯이 커서로 무언가를 가르켜 페이징을 하는 방법입니다.

여기서 커서는? **마지막으로 조회한 콘텐츠입니다.**



마지막으로 조회한 대상 그 다음부터 가져와!

이렇게 생각하면 됩니다.

이제 cursor paing 쿼리의 예시를 보여드릴게요.

cursor paging은 조금 더 어렵습니다. (파이팅!)

마지막으로 조회한 책의 좋아요가 20이라면 (물론 예시여서 좋아요 칼럼이 있다고 한 거예요!)

```
select * from book where book.likes < 20 order by likes desc limit 15;
```

이런 형태이며 실제로는 저렇게 보다는 마지막으로 조회한 책의 아이디를 가져와서

```
select * from book where book.likes <
    (select likes from book where id = 4)
    order by likes desc limit 15;
```

이런 형태의 쿼리가 됩니다.

이제 책 목록 조회 쿼리를 커서 페이징으로 해보겠습니다.

```
select * from book where created_at <
    (select created_at from book where id = 3)
    order by created_at desc limit 15;
```

이번에는 인기순 커서 페이징 입니다.

```
select * from book as b
    join (select count(*) as like_count
          from book_likes
          group by book_id) as likes on b.id = likes.book_id
    where likes.like_count < (select count(*) from book_likes where book_id = b.id)
    order by likes.like_count desc limit 15;
```



! 질문 !

이 인기 순 커서 페이징 쿼리가 과연 잘 작동할까요?

아뇨,



이유는, 예를 들어 **좋아요가 0개인 게시글이 400개**이고,

공교롭게 마지막으로 조회한 책의 좋아요가 0개라면,

그리고

아직 뒤에 조회를 하지 않은 책이 있다면

다음 페이지에 책이 목록으로 조회가 될까요?

**이런 경우가 있기에 인기 순 정렬 같이 같은 값이 있을 수 있는 경우
정렬 기준이 하나 더 있어야 합니다.**

따라서 위의 예시에서 좋아요 수가 같을 경우 PK값을 정렬 기준을 추가해보겠습니다.

```
select * from book as b
  join (select count(*) as like_count
        from book_likes
        group by book_id) as likes on b.id = likes.book_id
 where likes.like_count < (select count(*) from book_likes where book_id = b.id)
 order by likes.like_count desc, b.id desc limit 15;
```

이러면 성공 한 걸까요?

이 경우 에도 정렬 기준이 추가 된것 일 뿐

좋아요 갯수가 같은 book이 15개가 넘어가면 그 이상의 것은 무시가 됩니다.

둘의 조건을 한번에 분류할 수 있는 기준이 필요합니다.

밑에 것은 두 조건을 한번에 cursor 값으로 지정할 수 있는 코드입니다.

좋아요 갯수와, id를 하나의 문자열로 엮어서 cursor 값이 완전히 고유한 값이라고 볼 수 있습니다. (MySQL을 기준으로 썼고 SQL마다 문법이 조금씩 다를 수 있습니다.) (10자리를 기준으로 엮은 것이고 더 길어질 수도 있습니다.)

```
SELECT b.*,
       CONCAT(LPAD(likes.like_count, 10, '0'), LPAD(b.id, 10, '0')) AS cursor_value
FROM book AS b
JOIN (SELECT book_id, COUNT(*) AS like_count
      FROM book_likes
      GROUP BY book_id) AS likes ON b.id = likes.book_id
HAVING cursor_value < (SELECT CONCAT(LPAD(like_count_sub.like_count, 10, '0'), LPAD(3, 10, '0'))
                      FROM (SELECT book_id, COUNT(*) AS like_count
                            FROM book_likes
                            GROUP BY book_id) AS like_count_sub
                      WHERE like_count_sub.book_id = 3) # 여기에 cursor_value 값이 들어가
ORDER BY likes.like_count DESC, b.id DESC
LIMIT 15;
```

cursor_value를 위해 having을 썼지만 집계 함수를 쓰지 않고 having 절을 쓰기는 조금 비효율적 일 수 있으니

where 절로 구현한다면 이렇게 할 수도 있겠네요.

```
SELECT b.*,
       CONCAT(LPAD(likes.like_count, 10, '0'), LPAD(b.id, 10, '0')) AS cursor_value
FROM book AS b
JOIN (SELECT book_id, COUNT(*) AS like_count
      FROM book_likes
      GROUP BY book_id) AS likes ON b.id = likes.book_id
WHERE CONCAT(LPAD(likes.like_count, 10, '0'), LPAD(b.id, 10, '0')) <
```

```
(SELECT CONCAT(LPAD(like_count_sub.like_count, 10, '0'), LPAD(like_cour
FROM (SELECT book_id, COUNT(*) AS like_count
      FROM book_likes
      GROUP BY book_id) AS like_count_sub
WHERE like_count_sub.book_id = 3) # 여기에 cursor_value 값이 들어가면 됨.
ORDER BY likes.like_count DESC, b.id DESC
LIMIT 15;
```

마지막 코드의 경우 실제 서버 구현 때는 cursor 값을 그대로 받는 것이 아닌 정렬 기준이 되는 값들을 받아서 cursor 값을 만들어 적용시키면 더욱 좋을 것 같습니다! (또한 PK 값은 보통 Long으로 설정되는 경우가 많으니 너무 값이 커질 수 있어서 생성 시간과 같은 것으로 해도 좋을 것 같습니다!)

마지막 커서 기반 페이지네이션 SQL문이 너무 어려워 보인다면 참고 자료에 잘 설명된 블로그 글을 첨부해놓았으니 확인바랍니다.

핵심 키워드



주요 내용들에 대해 조사해보고, 자신만의 생각을 통해 정리해보세요!
레퍼런스를 참고하여 정의, 속성, 장단점 등을 적어주셔도 됩니다.
조사는 공식 홈페이지
Best, 블로그(최신 날짜) **Not Bad**

이번 주차는 키워드가 딱히 없습니다!

원하실 경우 join연산에 대해 더 알아보는 정도면 될 것 같습니다.

학습 후기

- 이번 주차 워크북을 해결해보면서 어땠는지 회고해봅시다.
- 핵심 키워드에 대해 완벽하게 이해했는지? 혹시 이해가 안 되는 부분은 무엇인지?



! 스터디 진행 방법

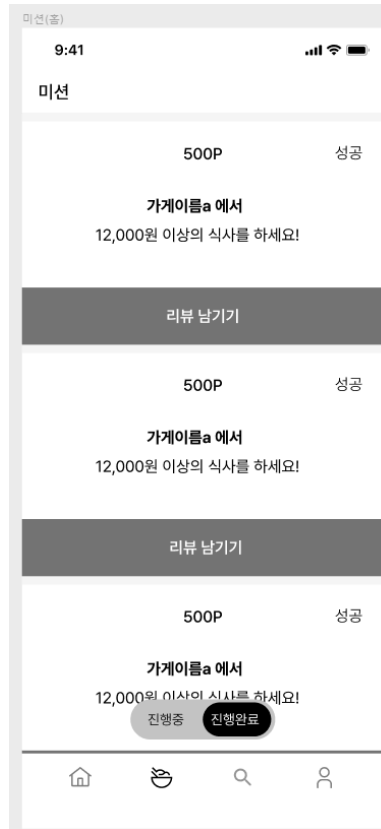
1. 스터디를 진행하기 전, 워크북 내용들을 모두 채우고 스터디에서는 서로 모르는 내용들을 공유해주세요.
2. 미션은 워크북 내용들을 모두 완료하고 나서 스터디 전/후로 진행해보세요.
3. 다음주 스터디를 진행하기 전, 지난주 미션을 서로 공유해서 상호 피드백을 진행하시면 됩니다.

✓ 실습 체크리스트

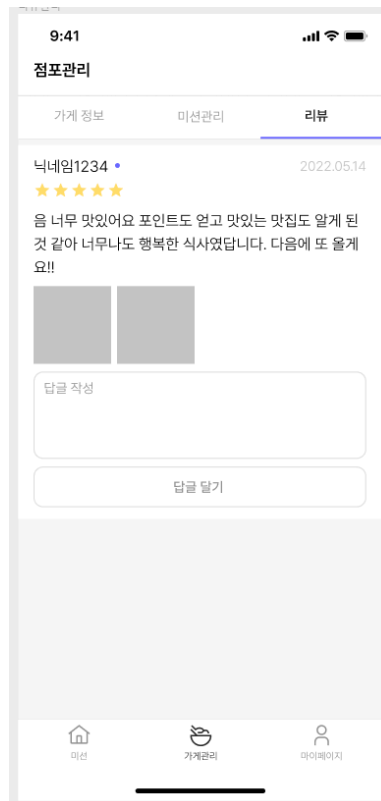
✓ 실습 인증

🔥 미션

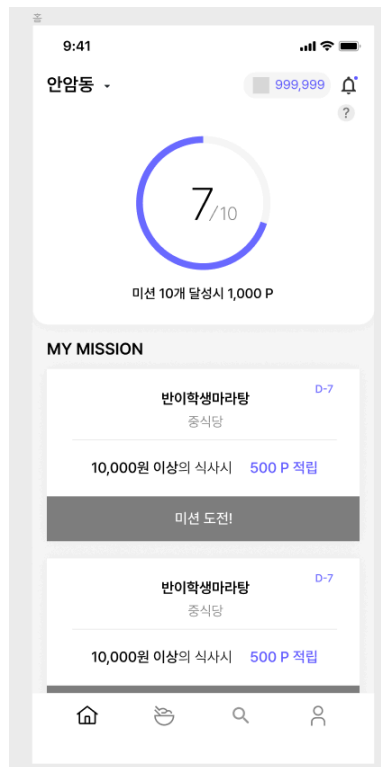
1. 1주차 때 설계한 데이터베이스를 토대로 아래의 화면에 대한 쿼리를 작성



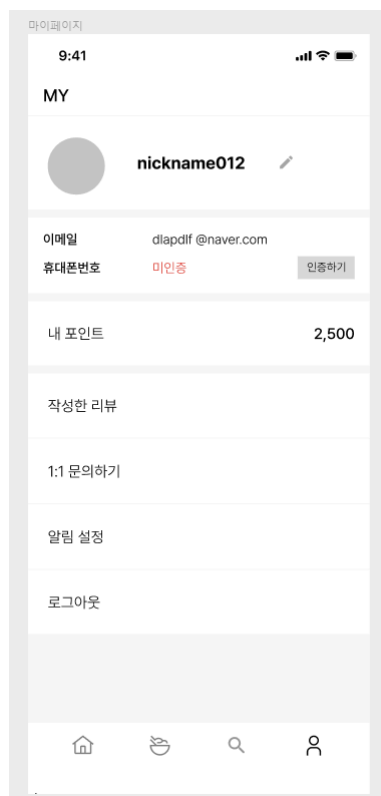
내가 진행중, 진행 완료한 미션 모아서 보는 쿼리(페이징 포함)



리뷰 작성하는 쿼리,
* 사진의 경우는 일단 배제




홈 화면 쿼리
(현재 선택 된 지역에서 도전이 가능한 미션 목록, 페이징 포함)



마이 페이지 화면 쿼리

< 시니어 미션 >

😊 시니어 미션

 **미션 기록 (여기에 해도 되고 위의 미션에서 각 페이지 밑에 간단하게 블록 만들어서 하셔도 됩니다!)**



미션 기록의 경우, 아래 미션 기록 토글 속에 작성하시거나, 페이지를 새로 생성하여 해당 페이지에 기록하여도 좋습니다!

하지만, 결과물만 올리는 것이 아닌, **중간 과정 모두 기록하셔야 한다는 점!** 잊지 말아주세요.

▼ 미션 기록

1. 내가 진행중인 미션, 진행 완료한 미션에 모아서 보기
 - 가게에 방문하는 것이 미션이므로 가게 방문여부를 받는 visit 컬럼을 가져오고 미션 완료시점과, 미션 완료여부컬럼이 있는 회원미션 테이블과 조인하여 데이터를 가져옴.
 - 미션완료한 가게데이터(boolean = 1)를 가져오고 미션 완료하지 않은 가게 데이터(boolean = 0)를 가져옴.
 - offset으로 limit 은 10개로 했다.

```

240 ✓ select
241     m.visit as 미션이름,
242     um.created_at,
243     um.point
244 from usermission as um
245     join missions as m
246         on um.미션id = m.id
247         and um.지역id = m.지역id
248         and um.가게id = m.가게id
249 where um.회원id = ?
250     and um.is_completed = 1
251 order by um.created_at desc
252 limit 10 offset 0; -- ← 페이지 1
253
254 select
255     m.visit as 미션이름,
256     um.created_at,
257     um.point
258 from usermission as um
259     join missions as m
260         on um.미션id = m.id
261         and um.지역id = m.지역id
262         and um.가게id = m.가게id
263 where um.회원id = ?
264     and um.is_completed = 0
265 order by um.created_at desc
266 limit 10 offset 0; -- ← 페이지 1

```

2. 리뷰 작성하는 쿼리(사진은 배제)

- 리뷰를 작성하는 작업이기 때문에 데이터를 넣어주는 insert into 를 사용했고 그 데이터를 reviews에 저장함. 회원 테이블의 id를 primary key 로 하고 데이터를 넣을때 sql에서 자동으로 회원아이디 가게 아이디 지역아이디정보가 저장되도록 auto_increment로 설정해주었음. 사용자는 리뷰내용과 별점만 입력함. 리뷰 작성일시, 수정일시 보여주기 위해 생성시작은 now로 추가해줌.

```

insert into reviews(
    회원id,
    가게id,
    지역id,
    review,
    review_star,
    created_at,
    upload_at
) values(
    ?, ?, ?, ?, ?, created_at now(), upload_at now()
);

```

3. 홈화면 작성하는 쿼리

- left join 연산을 활용해서 회원미션테이블의 미션 달성한 수 um.id로 count해서 가져온다.
- group by 연산자를 활용해서 미션별로 묶음.
- 지역구에따라서 그 지역구의 미션과 포인트를 데이터를 가져옴
- 페이징을 하므로 한 페이지의 10개로 limit 설정함.

```

select
    m.id,
    m.visit as 미션이름,
    m.mission_point as 포인트,
    ⚡ count(um.id) as 달성미션수
from missions as m
left join usermission as um
    on m.id = um.미션id
    and m.지역id = um.지역id
    and m.가게id = um.가게id
where m.지역id = ?
group by m.id, m.visit, m.mission_point
order by m.id desc
limit 10 offset 0;

```

4. 마이페이지 쿼리

- 회원이름과 회원의 정보를 가져오고 회원미션의 누적 포인트를 가져옴.
- 회원테이블과 회원 미션 테이블의 join연산을 통해 회원미션 테이블의 회원 포인트 데이터를 가져오고 획득포인트들의 합을 sum연산자를 통해 표현함.
- group by 연산자를 활용해서 회원 별로 누적된 포인트를 계산할 수 있게 함.

```

select
    m.user_name as 이름,
    m.user_id as 아이디,
    sum(um.point) as 누적포인트

from members as m
    left join usermission as um
        on m.id = um.회원id

where m.id = ?
group by m.id, m.user_name, m.user_id;

```

⚡ 트러블 슈팅



실습하면서 생긴 문제들에 대해서, **이슈 - 문제 - 해결** 순서로 작성해주세요.



스스로 해결하기 어렵다면? 스터디원들에게 도움을 요청하거나 **너디너리의 지식 IN 채널에 질문**해보세요!

▼ ⚡ 이슈 작성 예시 (이슈가 생기면 아래를 복사해서 No.1, No.2, No.3 ... 으로 작성해서 트러블 슈팅을 꼭 해보세요!)

이슈

👉 앱 실행 중에 노래 다음 버튼을 누르니까 앱이 종료되었다.

문제

👉 노래클래스의 데이터리스트의 Size를 넘어서 NullPointerException이 발생하여 앱이 종료된 것이었다.

해결

👉 노래 다음 버튼을 눌렀을 때 데이터리스트의 Size를 검사해 Size보다 넘어가려고 하면 다음으로 넘어가는 메서드를 실행시키지 않고, 첫 노래로 돌아가게끔 해결

참고레퍼런스

- 링크

▼ ⚡ 이슈 No.1

이슈

👉 [트러블이 생긴 상태 작성]

리뷰를 작성하는 쿼리에서 회원별로 리뷰를 작성하려면 회원 id, 가게id등 사용자가 적지 않아도 되는 데이터들의 정보가 필요하지만 처음엔 이를 사용자가 입력하도록 요구하는 쿼리를 만듦.

문제

👉 [어떤 이유로 해당 이슈가 일어났는지 작성]

sql상에서 구현하려면 사용자가 아이디, 가게, 등 정보를 다 입력해야함.

해결

👉 [해결 방법 작성]

리뷰테이블의 id를 auto_increment로 설정하여 sql상에서 자동으로 가게id등 사용자가 입력하지 않아도 되는 정보를 자동으로 할당해줌.

참고레퍼런스

- [문제 해결 시 참고한 링크]

▼ ⚡ 이슈 No.1

이슈

👉 [트러블이 생긴 상태 작성]

문제

👉 [어떤 이유로 해당 이슈가 일어났는지 작성]

해결

👉 [해결 방법 작성]

참고레퍼런스

- [문제 해결 시 참고한 링크]

▼ https://diyinfo.tistory.com/entry/리뷰-테이블-생성-상품-리뷰-정보를-저장하기-위한-테이블을-생성합니다#google_vignette

참고 자료

2주차

| Aa 이름 | 🕒 생성일 | ⋮ 태그 |
|----------------------------|------------------------|------|
| <u>커서 기반 페이지 네이션 참고 자료</u> | @2025년 3월 15일 오후 11:26 | |

Copyright © 2023 최용욱(뜰이) All rights reserved.

Copyright © 2024, 2025 제이미(김준환) All rights reserved.