쿼리 개선 작업

들어가며

- 사이드 프로젝트를 개발하는데 있어서 JPA를 사용했는데, JPA의 대한 학습 부족과 시간의 촉박함이라는 이유(핑계)로 좋은 쿼리문을 작성하지 못했었습니다.
- 따라서 더 좋은 애플리케이션이 되기 위해서 쿼리를 전반적으로 점검하고 개선하기로 하였습니다.
- 목표는 아래와 같습니다.
  - **성능 측정 도구를 활용**하여 성능을 측정하고 **개선 정도를 확인**한다.
  - 개선작업을 진행하며 **JPA에 대해 더 깊게 이해**한다.
  - **테이블 설계에 대한 고민**을 한번 더 해본다.

쿼리 성능 측정 도구 후보

- 쿼리 성능 측정을 위해 고려해본 도구들은 아래 **세가지와 같습니다.**
- **프로파일러**
  - Profiler, YourKit를 사용해서 JPA 쿼리의 성능을 분석한다.
  - 애플리케이션 전체의 메모리 사용, 스레드 활동, CPU의 사용량도 확인할 수 있다.
- **APM 활용**
  - Datadog, Elastic APM 활용
  - 프로파일러와 동일하게 애플리케이션 전반적인 성능을 모니터링하고 분석할 수 있다.
  - 추가적으로 에러 로깅 및 추적이 가능하다.
- **JPQL 또는 Criteria 쿼리의 Explain Plan 사용**
  - DataGrip 툴에서 제공하는 Explain Plan을 사용한다.
  - 순수 쿼리 자체에 대한 분석을 제공하고 시각화 해준다.
- 결론: 현재는 애플리케이션의 성능을 측정하는 것이 목표가 아니기 때문에, 순수 쿼리 자체에 대한 분석을 위해 Explain Plan을 사용하기로 했습니다. (물론 애플리케이션 자체도 모니터링이 필요하기는 합니다만, 추후 고려해볼 예정이고 현재로서는 비용이 발생한다는 점도 크게 한몫 했습니다!)

Explain Plan의 정보 분석

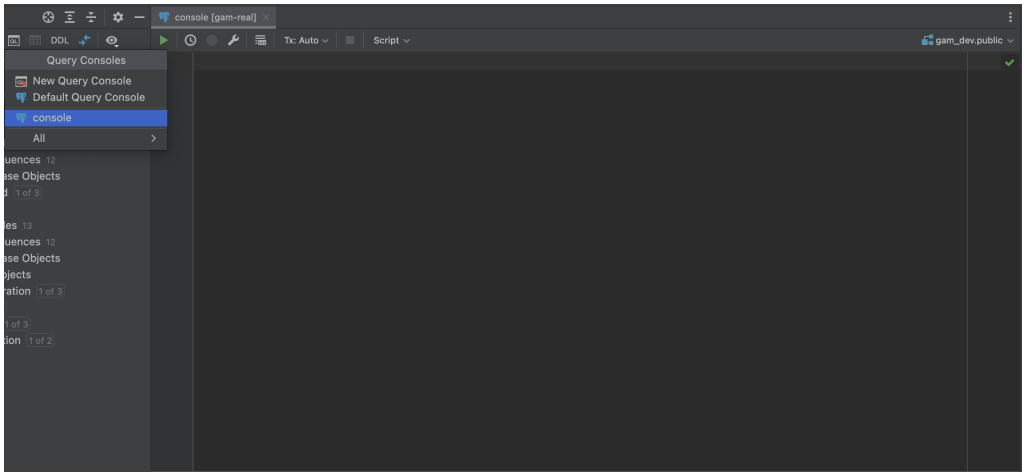
- 모든 쿼리마다 Explain Plan을 사용할 예정인데, 해당 분석에서 제공하는 정보에 대한 설명을 적어두려고 합니다.

- **Operation**: 실행된 작업의 종류를 나타냅니다. 여기서는 **Index Scan**이라는 작업이 표시되고 있는데, 이는 인덱스를 사용하여 데이터를 조회하는 작업을 의미합니다.

- **Params**: 쿼리 실행에 사용된 파라미터를 나타냅니다. 예를 들어, 인덱스 이름이나 스캔 조건 등이 여기에 포함될 수 있습니다.
- **Rows**: 예상되는 결과 행 수입니다. 즉, 쿼리 실행 결과로 반환될 것으로 **예상되는 행의 수**를 나타냅니다.
- **Actual Rows**: **실제로 반환된 결과 행 수**입니다. 이는 `Rows` 값과 다를 수 있으며, 실행 계획의 정확성을 평가하는 데 사용될 수 있습니다.
- **Total Cost**: 쿼리를 완료하는 데 **예상되는 총 비용**입니다. 이 비용은 PostgreSQL의 내부 비용 모델을 기반으로 하며, 일반적으로 디스크 I/O 비용과 CPU 비용을 포함합니다.
- **Actual Total Time**: 쿼리가 **실제로 실행되어 완료되는 데 걸린 시간**입니다.
- **Startup Cost**: 쿼리의 **첫 번째 행을 반환하기까지의 예상 비용**입니다. 이는 쿼리가 결과를 생성하기 시작할 때까지 얼마나 많은 리소스가 소모될 것인지를 나타냅니다.
- **Actual Startup Time**: **실제로 첫 번째 결과 행을 반환하는 데 걸린 시간**입니다.
- **Raw Desc**: 추가적인 정보를 제공할 수 있는 설명이나 메타데이터를 포함할 수 있습니다. 예를 들어, `Parallel Aware`는 쿼리가 병렬 처리를 지원하는지 여부를 나타냅니다.
- 해당 항목 중에서 집중적으로 **Actual Total Time(+Actual Startup Time)**를 확인해보겠습니다.

작업 목록

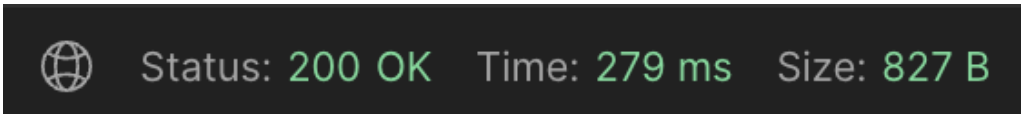
- ~~소셜로그인~~
- 발견 - 유저 포트폴리오 상세보기 /api/v1/user/portfolio/{userId}
- 영감 매거진
- 매거진 - 발견
- 스크랩 - 매거진 스크랩 뷰
- 스크랩 - 매거진 스크랩 하기 안하기
- 마이페이지 - 유저 포트폴리오 보기
- 마이페이지 - 작업물 삭제
- 마이페이지 - 작업물 추가
- 매거진 - 매거진 세부



1번 케이스

개선 전

개선 전 API 통신 비용



- 총 실행 시간: 279ms (0.279s)

개선 전 쿼리

- 쿼리 개수: 11개
- 총 실행 시간: 258ms (0.258s)

1번째 쿼리

JPA

```
Hibernate:
    select
        user0_.user_id as user_id1_8_0_,
        user0_.created_at as created_2_8_0_,
        user0_.modified_at as modified3_8_0_,
        user0_.behance as behance4_8_0_,
        user0_.detail as detail5_8_0_,
        user0_.device_token as device_t6_8_0_,
        user0_.email as email7_8_0_,
        user0_.first_work_id as first_wo8_8_0_,
        user0_.info as info9_8_0_,
        user0_.instagram as instagr10_8_0_,
        user0_.magazine_view_count as magazin11_8_0_,
        user0_.notion as notion12_8_0_,
        user0_.role as role13_8_0_,
        user0_.scrap_count as scrap_c14_8_0_,
        user0_.selected_first_at as selecte15_8_0_,
        user0_.tag as tag16_8_0_,
        user0_.user_name as user_na17_8_0_,
        user0_.user_status as user_st18_8_0_,
        user0_.view_count as view_co19_8_0_,
        user0_.work_thumb_nail as work_th20_8_0_
    from
        "user" user0_
    where
        user0_.user_id=?
```

- API를 요청한 유저의 user 정보를 가져오는 쿼리

SQL

```
SELECT
    user_id, created_at, modified_at, behance, detail, device_token,
    email, first_work_id, info, instagram, magazine_view_count,
    notion, role, scrap_count, selected_first_at, tag, user_name,
    user_status, view_count, work_thumb_nail
FROM
    "user"
WHERE
```

```
    user_id = <사용자 ID>;
```

Explain

- **Actual Total Time**: 0.015

2번째 쿼리

```
Hibernate:
    select
        authprovid0_.auth_provider_id as auth_pro1_0_,
        authprovid0_.created_at as created_2_0_,
        authprovid0_.modified_at as modified3_0_,
        authprovid0_.provider_type as provider4_0_,
        authprovid0_.user_id as user_id5_0_
    from
        "auth_provider" authprovid0_
    where
        authprovid0_.user_id=?
```

- API를 요청한 유저의 third party user 정보를 가져오는 쿼리

SQL

```
SELECT
    user_id, created_at, modified_at, behance, detail, device_token,
    email, first_work_id, info, instagram, magazine_view_count,
    notion, role, scrap_count, selected_first_at, tag, user_name,
    user_status, view_count, work_thumb_nail
FROM
    "user"
WHERE
    user_id = <사용자 ID>;
```

Explain

- Actual Total Time: 0.009

3번째 쿼리

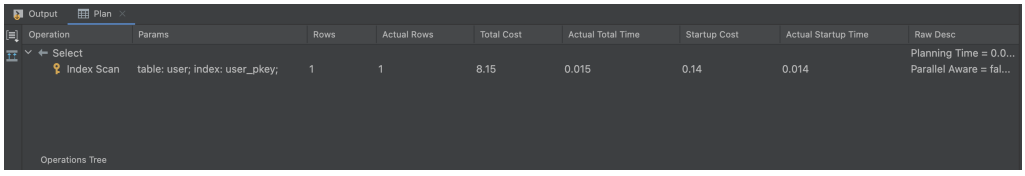
```
Hibernate:
    select
        user0_.user_id as user_id1_8_0_,
        user0_.created_at as created_2_8_0_,
        user0_.modified_at as modified3_8_0_,
        user0_.behance as behance4_8_0_,
        user0_.detail as detail5_8_0_,
        user0_.device_token as device_t6_8_0_,
        user0_.email as email7_8_0_,
        user0_.first_work_id as first_wo8_8_0_,
        user0_.info as info9_8_0_,
        user0_.instagram as instagr10_8_0_,
        user0_.magazine_view_count as magazin11_8_0_,
        user0_.notion as notion12_8_0_,
        user0_.role as role13_8_0_,
        user0_.scrap_count as scrap_c14_8_0_,
        user0_.selected_first_at as selecte15_8_0_,
        user0_.tag as tag16_8_0_,
        user0_.user_name as user_na17_8_0_,
        user0_.user_status as user_st18_8_0_,
        user0_.view_count as view_co19_8_0_,
        user0_.work_thumb_nail as work_th20_8_0_
    from
        "user" user0_
    where
        user0_.user_id=?
```

- API를 요청한 유저의 user 정보를 가져오는 쿼리

SQL

```
SELECT
    user_id, created_at, modified_at, behance, detail, device_token,
    email, first_work_id, info, instagram, magazine_view_count,
    notion, role, scrap_count, selected_first_at, tag, user_name,
    user_status, view_count, work_thumb_nail
FROM
    "user"
WHERE
    user_id = <사용자 ID>;
```

Explain



- **Actual Total Time**: 0.015

4번째 쿼리

```
Hibernate:
    select
        user0_.user_id as user_id1_8_0_,
        user0_.created_at as created_2_8_0_,
```

```
        user0_.modified_at as modified3_8_0_,
        user0_.behance as behance4_8_0_,
        user0_.detail as detail5_8_0_,
        user0_.device_token as device_t6_8_0_,
        user0_.email as email7_8_0_,
        user0_.first_work_id as first_wo8_8_0_,
        user0_.info as info9_8_0_,
        user0_.instagram as instagr10_8_0_,
        user0_.magazine_view_count as magazin11_8_0_,
        user0_.notion as notion12_8_0_,
        user0_.role as role13_8_0_,
        user0_.scrap_count as scrap_c14_8_0_,
        user0_.selected_first_at as selecte15_8_0_,
        user0_.tag as tag16_8_0_,
        user0_.user_name as user_na17_8_0_,
        user0_.user_status as user_st18_8_0_,
        user0_.view_count as view_co19_8_0_,
        user0_.work_thumb_nail as work_th20_8_0_
    from
        "user" user0_
    where
        user0_.user_id=?
```

- 요청할 대상인 유저의 유저 정보를 가져오는 쿼리

SQL

```
SELECT
    user_id, created_at, modified_at, behance, detail, device_token,
    email, first_work_id, info, instagram, magazine_view_count,
    notion, role, scrap_count, selected_first_at, tag, user_name,
    user_status, view_count, work_thumb_nail
FROM
    "user"
WHERE
    user_id = <사용자 ID>;
```

Explain

- **Actual Total Time**: 0.015

5번째 쿼리

```
Hibernate:
    update
        "user"
    set
        created_at=?,
        modified_at=?,
        behance=?,
        detail=?,
        device_token=?,
        email=?,
        first_work_id=?,
```

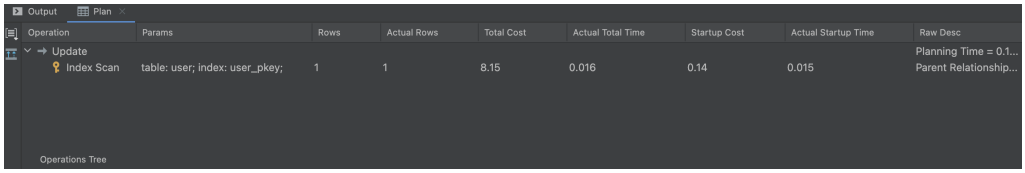
```
        info=?,
        instagram=?,
        magazine_view_count=?,
        notion=?,
        role=?,
        scrap_count=?,
        selected_first_at=?,
        tag=?,
        user_name=?,
        user_status=?,
        view_count=?,
        work_thumb_nail=?
    where
        user_id=?
```

- user의 viewCount를 늘리는 update 쿼리

SQL

```
UPDATE "user"
SET
    view_count = <업데이트된 조회수>;
WHERE
    user_id = <사용자 ID>;
```

Explain



- **Actual Total Time**: 0.016

6번째 쿼리

```
Hibernate:
    select
        work0_.work_id as work_id1_12_,
        work0_.created_at as created_2_12_,
        work0_.modified_at as modified3_12_,
        work0_.detail as detail4_12_,
        work0_.is_active as is_activ5_12_,
        work0_.is_first as is_first6_12_,
        work0_.photo_url as photo_ur7_12_,
        work0_.title as title8_12_,
        work0_.user_id as user_id10_12_,
        work0_.view_count as view_cou9_12_
    from
        "work" work0_
    left outer join
        "user" user1_
            on work0_.user_id=user1_.user_id
    where
        user1_.user_id=?
        and work0_.is_first=?
        and work0_.is_active=?
```

```
    order by
        work0_.created_at desc
```

- 유저의 작업물들을 가져오는 쿼리

SQL

```
SELECT
    work_id, is_active, is_first,
    photo_url, title
FROM
    "work"
LEFT OUTER JOIN
    "user" ON work.user_id = "user".user_id
WHERE
    "user".user_id = <사용자 ID> AND
    work.is_first = true AND
    work.is_active = true
ORDER BY
    work.created_at DESC
```

Explain

전체

- **Actual Total Time**: 0.033

Nested Loops (Inner):

- **Actual Total Time**: 0.028
- 두 테이블('work'과 'user') 간의 조인을 수행하는 데 걸린 실제 시간입니다.

Full Scan (on table: work):

- **Actual Total Time**: 0.011
- 'work' 테이블에서 풀 스캔을 수행하는 데 걸린 시간입니다. (풀 스캔은 인덱스를 사용하지 않고 테이블의 모든 레코드를 검사)

Index Scan (on table: user, index: user_pkey):

- **Actual Total Time**: 0.015초
- 'user' 테이블에서 기본 키 인덱스('user_pkey')를 사용하여 인덱스 스캔을 수행하는 데 걸린 시간입니다.

7번째 쿼리

```
Hibernate:
    select
        work0_.work_id as work_id1_12_,
        work0_.created_at as created_2_12_,
        work0_.modified_at as modified3_12_,
        work0_.detail as detail4_12_,
        work0_.is_active as is_activ5_12_,
        work0_.is_first as is_first6_12_,
        work0_.photo_url as photo_ur7_12_,
        work0_.title as title8_12_,
```

```
        work0_.user_id as user_id10_12_,
        work0_.view_count as view_cou9_12_
    from
        "work" work0_
    left outer join
        "user" user1_
            on work0_.user_id=user1_.user_id
    where
        user1_.user_id=?
        and work0_.is_first=?
```

- 유저의 작업물들중 가장 첫번째(대표작)을 가져오는 쿼리

SQL

```
SELECT
    work_id, is_active, is_first,
    photo_url, title
FROM
    "work"
LEFT OUTER JOIN
    "user" ON work.user_id = "user".user_id
WHERE
    "user".user_id = <사용자 ID> AND
    work.is_first = true AND
    work.is_active = true
ORDER BY
    work.created_at DESC
```

Explain

Nested Loops (Inner):

- **Actual Total Time**: 0.065
- 두 테이블('work'과 'user') 간의 조인을 수행하는 데 걸린 실제 시간입니다.

Full Scan (on table: work):

- **Actual Total Time**: 0.047
- 'work' 테이블에서 풀 스캔을 수행하는 데 걸린 시간입니다. (풀 스캔은 인덱스를 사용하지 않고 테이블의 모든 레코드를 검사)

Index Scan (on table: user, index: user_pkey):

- **Actual Total Time**: 0.015초
- 'user' 테이블에서 기본 키 인덱스('user_pkey')를 사용하여 인덱스 스캔을 수행하는 데 걸린 시간입니다.

8번째 쿼리

```
Hibernate:
    select
        userscraps0_.user_id as user_id6_10_0_,
        userscraps0_.user_scrap_id as user_scr1_10_0_,
        userscraps0_.user_scrap_id as user_scr1_10_1_,
```

```
        userscraps0_.created_at as created_2_10_1_,
        userscraps0_.modified_at as modified3_10_1_,
        userscraps0_.status as status4_10_1_,
        userscraps0_.target_id as target_i5_10_1_,
        userscraps0_.user_id as user_id6_10_1_
    from
        "user_scrap" userscraps0_
    where
        (
            userscraps0_.status = true
        )
        and userscraps0_.user_id=?
```

- 유저의 스크랩들을 가져오는 쿼리

SQL

```
SELECT
    user_scrap_id, created_at, modified_at, status, target_id, user_id
FROM
    "user_scrap"
WHERE
    status = TRUE AND
    user_id = <사용자 ID>;
```

Explain

- **Actual Total Time**: 0.064

9~11번째 쿼리

```
Hibernate:
    update
        "work"
    set
        created_at=?,
        modified_at=?,
        detail=?,
        is_active=?,
        is_first=?,
        photo_url=?,
        title=?,
        user_id=?,
        view_count=?
    where
        work_id=?
```

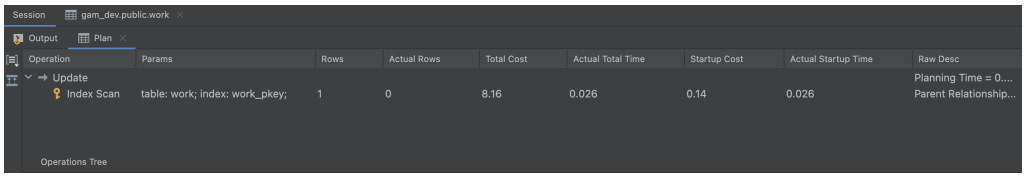
- 유저가 조회한 유저의 작업물들의 조회수를 추가하는 쿼리

SQL

```
UPDATE
    "work"
```

```
SET
    created_at = <새 생성 일시>, modified_at = <새 수정 일시>, view_count = <새 조회수
WHERE
    work_id = <작업 ID>;
```

Explain



- **Actual Total Time**: 0.026 (쿼리 한개당, 0.026N)

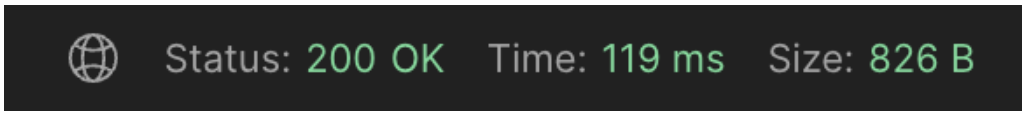
불필요한 쿼리 정리

- spring security단에서 유저를 authorizate할 때 user 테이블, auth_provider 테이블에서 조회하여 @AuthenticationPrincipal에 넣어준다.
    - 다만 userId 정보만 넣어주기 때문에, 유저 엔티티의 다른 부분을 조회하기 위해서 user 테이블을 한번 더 조회하는 작업을 진행하고 있다. (3번쿼리)
    - authorizate할 때, userId 정보만 넣어주지말고 User 엔티티 객체를 모두 넣어주어서 불필요하게 쿼리를 한번 더 실행하지 않도록 변경이 필요하다.
- user의 작업물인 work들을 가져올때 한번더 쿼리가 날라가고 있다.
    - user와 work의 경우 @OneToMany관계이기 때문에 fetchJoin을 통해서 user를 조회할 때 한번에 가져오도록 변경하여 해당 쿼리를 실행하지 않도록 변경이 필요하다. (6번 쿼리)
- 유저의 대표작을 가져와서 가장 앞단에 두기 위해서 불필요하게 user의 작업물인 work들을 가져오는 쿼리를 한번 더 실행한다. (7번 쿼리)
    - 굳이 쿼리를 사용해 I/O 비용을 감수하지 않고, 배열의 최대 4개 요소만이 들어가기 때문에 서버 자원을 사용해서 정렬하고 정제한다.
- user의 스크랩 리스트인 user_scrap들을 가져올때 한번더 쿼리가 날라가고 있다.
    - user와 user_scrap의 경우 @OneToMany관계이기 때문에 fetchJoin을 통해서 user를 조회할 때 한번에 가져오도록 변경하여 해당 쿼리를 실행하지 않도록 변경이 필요하다. (8번 쿼리)
- 작업물들의 배열 works를 돌면서 하나씩 work를 업데이트하는 쿼리가 날아가고 있는데, 한번에 반영될 수 있도록 변경이 필요하다. (9~11번)
    - 벌크 연산를 활용해보자.

```
// e.g
@Modifying
@Query("UPDATE User u SET u.status = :status WHERE u.id IN :ids")
void updateUsersStatus(String status, List<Long> ids);
```

개선 후

개선 후 API 통신 비용



- 총 실행 시간: 119ms (0.119s)

개선 후 쿼리

- 쿼리 개수: 6개
- 총 실행 시간: 77ms (0.077s)

1번째 쿼리 (개선 전과 동일)

JPA

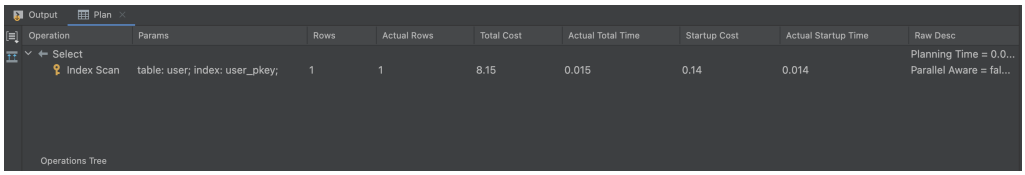
```
Hibernate:
    select
        user0_.user_id as user_id1_8_,
        user0_.created_at as created_2_8_,
        user0_.modified_at as modified3_8_,
        user0_.behance as behance4_8_,
        user0_.detail as detail5_8_,
        user0_.device_token as device_t6_8_,
        user0_.email as email7_8_,
        user0_.first_work_id as first_wo8_8_,
        user0_.info as info9_8_,
        user0_.instagram as instagr10_8_,
        user0_.magazine_view_count as magazin11_8_,
        user0_.notion as notion12_8_,
        user0_.role as role13_8_,
        user0_.scrap_count as scrap_c14_8_,
        user0_.selected_first_at as selecte15_8_,
        user0_.tag as tag16_8_,
        user0_.user_name as user_na17_8_,
        user0_.user_status as user_st18_8_,
        user0_.view_count as view_co19_8_,
        user0_.work_thumb_nail as work_th20_8_
    from
        "user" user0_
    where
        user0_.user_id=?
```

- API를 요청한 유저의 user 정보를 가져오는 쿼리

SQL

```
SELECT
    user_id, created_at, modified_at, behance, detail, device_token,
    email, first_work_id, info, instagram, magazine_view_count,
    notion, role, scrap_count, selected_first_at, tag, user_name,
    user_status, view_count, work_thumb_nail
FROM
    "user"
WHERE
    user_id = <사용자 ID>;
```

Explain



- **Actual Total Time**: 0.015

2번째 쿼리 (개선 전과 동일)

```
Hibernate:
    select
        authprovid0_.auth_provider_id as auth_pro1_0_,
        authprovid0_.created_at as created_2_0_,
        authprovid0_.modified_at as modified3_0_,
        authprovid0_.provider_type as provider4_0_,
        authprovid0_.user_id as user_id5_0_
    from
        "auth_provider" authprovid0_
    where
        authprovid0_.user_id=?
```

- API를 요청한 유저의 third party user 정보를 가져오는 쿼리

SQL

```
SELECT
    user_id, created_at, modified_at, behance, detail, device_token,
    email, first_work_id, info, instagram, magazine_view_count,
    notion, role, scrap_count, selected_first_at, tag, user_name,
    user_status, view_count, work_thumb_nail
FROM
    "user"
WHERE
    user_id = <사용자 ID>;
```

Explain

- Actual Total Time: 0.009

3번째 쿼리

```
Hibernate:
    select
        user0_.user_id as user_id1_8_0_,
```

```
        works1_.work_id as work_id1_12_1_,
        user0_.created_at as created_2_8_0_,
        user0_.modified_at as modified3_8_0_,
        user0_.behance as behance4_8_0_,
        user0_.detail as detail5_8_0_,
        user0_.device_token as device_t6_8_0_,
        user0_.email as email7_8_0_,
        user0_.first_work_id as first_wo8_8_0_,
        user0_.info as info9_8_0_,
        user0_.instagram as instagr10_8_0_,
        user0_.magazine_view_count as magazin11_8_0_,
        user0_.notion as notion12_8_0_,
        user0_.role as role13_8_0_,
        user0_.scrap_count as scrap_c14_8_0_,
        user0_.selected_first_at as selecte15_8_0_,
        user0_.tag as tag16_8_0_,
        user0_.user_name as user_na17_8_0_,
        user0_.user_status as user_st18_8_0_,
        user0_.view_count as view_co19_8_0_,
        user0_.work_thumb_nail as work_th20_8_0_,
        works1_.created_at as created_2_12_1_,
        works1_.modified_at as modified3_12_1_,
        works1_.detail as detail4_12_1_,
        works1_.is_active as is_activ5_12_1_,
        works1_.is_first as is_first6_12_1_,
        works1_.photo_url as photo_ur7_12_1_,
        works1_.title as title8_12_1_,
        works1_.user_id as user_id10_12_1_,
        works1_.view_count as view_cou9_12_1_,
        works1_.user_id as user_id10_12_0__,
        works1_.work_id as work_id1_12_0__
    from
        "user" user0_
```

- API를 요청한 유저의 user 정보를 가져오는 쿼리, work를 fetchJoin해서 가져온다.

SQL

```
SELECT
    user0_.user_id AS user_id1_8_0_,
    works1_.work_id AS work_id1_12_1_,
    user0_.created_at AS created_2_8_0_,
    user0_.modified_at AS modified3_8_0_,
    user0_.behance AS behance4_8_0_,
    user0_.detail AS detail5_8_0_,
    user0_.device_token AS device_t6_8_0_,
    user0_.email AS email7_8_0_,
    user0_.first_work_id AS first_wo8_8_0_,
    user0_.info AS info9_8_0_,
    user0_.instagram AS instagr10_8_0_,
    user0_.magazine_view_count AS magazin11_8_0_,
    user0_.notion AS notion12_8_0_,
    user0_.role AS role13_8_0_,
    user0_.scrap_count AS scrap_c14_8_0_,
    user0_.selected_first_at AS selecte15_8_0_,
    user0_.tag AS tag16_8_0_,
    user0_.user_name AS user_na17_8_0_,
    user0_.user_status AS user_st18_8_0_,
    user0_.view_count AS view_co19_8_0_,
    user0_.work_thumb_nail AS work_th20_8_0_,
    works1_.created_at AS created_2_12_1_,
    works1_.modified_at AS modified3_12_1_,
    works1_.detail AS detail4_12_1_,
```

```
    works1_.is_active AS is_activ5_12_1_,
    works1_.is_first AS is_first6_12_1_,
    works1_.photo_url AS photo_ur7_12_1_,
    works1_.title AS title8_12_1_,
    works1_.user_id AS user_id10_12_1_,
    works1_.view_count AS view_cou9_12_1_,
    works1_.user_id AS user_id10_12_0__,
    works1_.work_id AS work_id1_12_0__
FROM
    "user" user0_
INNER JOIN
```

Explain

Nested Loops (Inner):

- **Actual Total Time**: 0.028
- 두 테이블('work'과 'user') 간의 조인을 수행하는 데 걸린 실제 시간입니다.

Full Scan (on table: work):

- **Actual Total Time**: 0.018
- 'work' 테이블에서 풀 스캔을 수행하는 데 걸린 시간입니다. (풀 스캔은 인덱스를 사용하지 않고 테이블의 모든 레코드를 검사)

Index Scan (on table: user, index: user_pkey):

- **Actual Total Time**: 0.07초
- 'user' 테이블에서 기본 키 인덱스('user_pkey')를 사용하여 인덱스 스캔을 수행하는 데 걸린 시간입니다.

4번째 쿼리

```
Hibernate:
    update
        "work"
    set
        view_count=view_count+1
    where
        work_id in (
            ? , ? , ?
        )
```

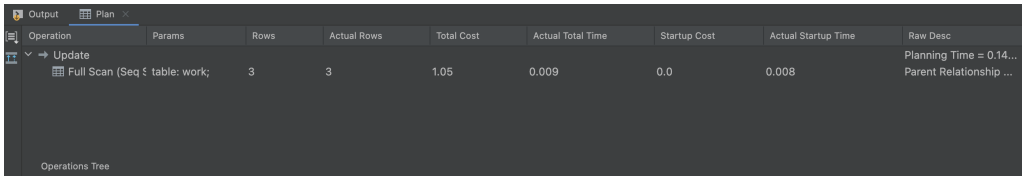
- work id를 기준으로 찾아서 viewCount를 하나씩 올려주는 쿼리

SQL

```
update
    "work"
set
    view_count=view_count+1
```

```
where
  work_id iHibernate:
    update
        "work"
    set
        view_count=view_count+1
    where
        work_id in (
            <작업물 ID 1> , <작업물 ID 2> , <작업물 ID 3>
        )
```

Explain



- **Actual Total Time**: 0.009

5번째 쿼리 (개선 전과 동일)

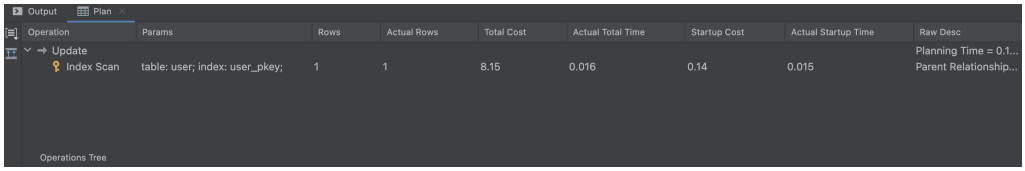
```
Hibernate:
    update
        "user"
    set
        created_at=?,
        modified_at=?,
        behance=?,
        detail=?,
        device_token=?,
        email=?,
        first_work_id=?,
        info=?,
        instagram=?,
        magazine_view_count=?,
        notion=?,
        role=?,
        scrap_count=?,
        selected_first_at=?,
        tag=?,
        user_name=?,
        user_status=?,
        view_count=?,
        work_thumb_nail=?
    where
        user_id=?
```

- user의 viewCount를 늘리는 update 쿼리

SQL

```
UPDATE "user"
SET
    view_count = <업데이트된 조회수>;
WHERE
    user_id = <사용자 ID>;
```

Explain



- **Actual Total Time**: 0.016

6번째 쿼리

```
Hibernate:
    select
        userscrap0_.user_scrap_id as user_scr1_10_,
        userscrap0_.created_at as created_2_10_,
        userscrap0_.modified_at as modified3_10_,
        userscrap0_.status as status4_10_,
        userscrap0_.target_id as target_i5_10_,
        userscrap0_.user_id as user_id6_10_
    from
        "user_scrap" userscrap0_
    left outer join
        "user" user1_
            on userscrap0_.user_id=user1_.user_id
    where
        user1_.user_id = <사용자 ID>
        and userscrap0_.target_id = <타겟 사용자 ID>
```

- 유저의 작업물 스크랩을 가져오는 쿼리

SQL

```
SELECT
    userscrap0_.user_scrap_id AS user_scr1_10_,
    userscrap0_.created_at AS created_2_10_,
    userscrap0_.modified_at AS modified3_10_,
    userscrap0_.status AS status4_10_,
    userscrap0_.target_id AS target_i5_10_,
    userscrap0_.user_id AS user_id6_10_
FROM
    "user_scrap" userscrap0_
WHERE
    userscrap0_.user_id = <사용자 ID>  AND
    userscrap0_.target_id = <타겟 사용자 ID>
```

Explain

- **Actual Total Time**: 0.011

개선 결과

	총 실행 시간	쿼리 수행 시간	쿼리 개수
개선 전	279ms (0.279s)	258ms (0.258s)	11개
개선 후	119ms (0.119s)	77ms (0.077s)	6개

- 위와 같은 개선 결과를 얻었습니다.
- 현재는 데이터가 개발용으로 많지 않기 때문에 차이가 적어보이지만, 데이터가 많아질수록 개선 정도가 더 커질 것입니다.