REST Project Presentation

REST 프로젝트 결과 발표

발표자 김영웅

팀원: 김동환 김현우 정윤정
박성준 장동필

프로젝트 기간:
24.08.13 ~ 24.09.09

CONTENTS

프로젝트 개요: 팀원 소개 및 역할

팀장 김영웅	엔터티 설계, 초성 검색 로직 구현, 검색 쿼리 최적화	팀원 박성준	마이페이지, 그룹 , 아티스트 페이지 구현
팀원 김동환	음원 상세 페이지, 검색 쿼리 작성, 검색 결과 페이지	팀원 장동필	앨범 차트 페이지, 좋아요 통계 테이블 구현
팀원 김현우	음원 차트, 정보 수정 페이지, 초성 검색 구현, AWS 배포	팀원 정윤정	로그인/회원가입 페이지, 그룹 업데이트 쿼리 작성

프로젝트 개요: 배경 및 목표

프로젝트 배경

- 대용량 음악 데이터 관리의 필요성
- 효율적인 검색 시스템의 중요성

프로젝트 목표

- 대용량 데이터베이스 구축
- 빠르고 정확한 검색 기능 구현
- 다양한 검색 방식 제공 (전체 검색, 초성 검색 등)

기술 스택


Spring Boot

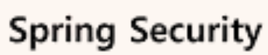

JPA


Query DSL

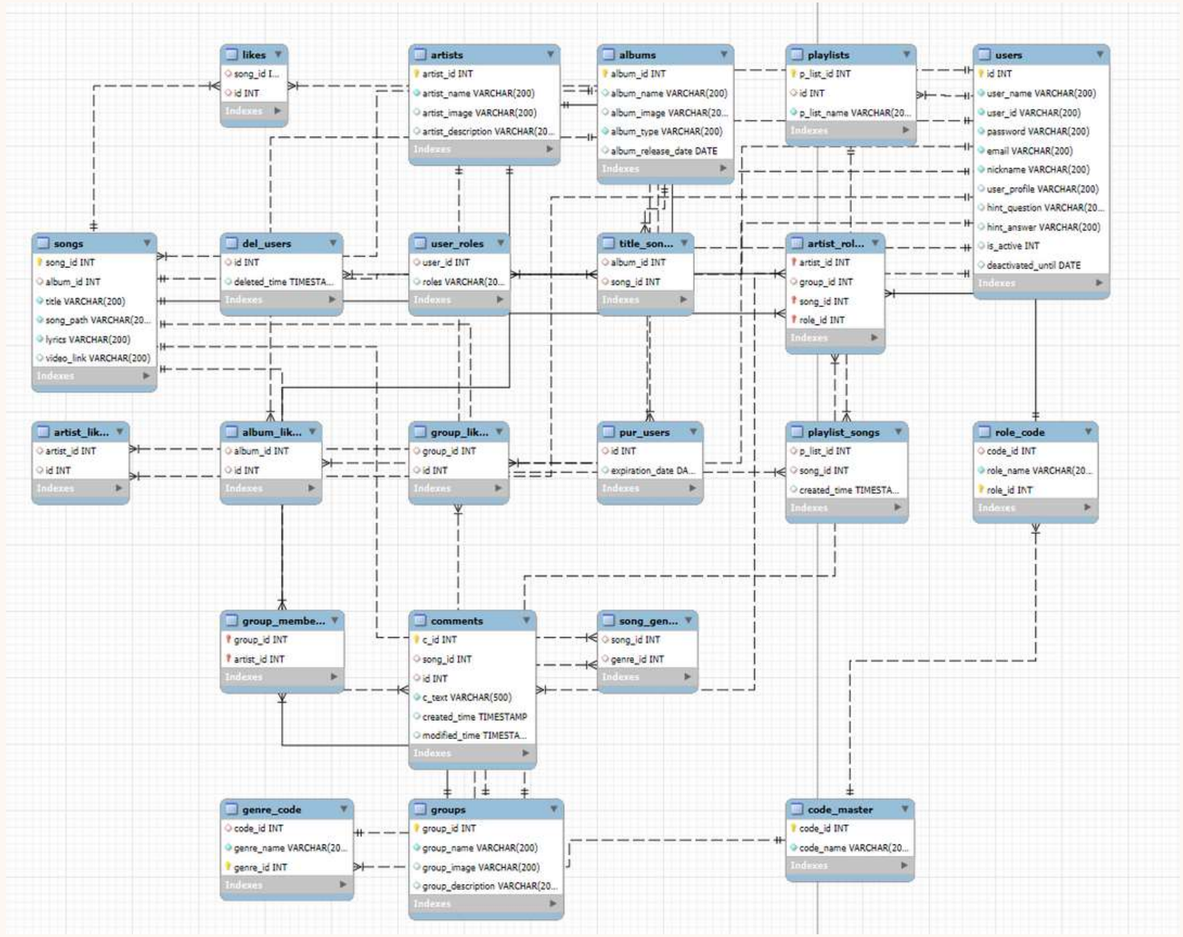

aws


AXIOS


HTML
CSS
JS


Spring Security

데이터 베이스 구조


likes
song_id I...
id INT
Indexes
artists
artist_id INT
artist_name VARCHAR(200)
artist_image VARCHAR(200)
artist_description VARCHAR(20...
Indexes
albums
album_id INT
album_name VARCHAR(200)
album_image VARCHAR(20...
album_type VARCHAR(200)
album_release_date DATE
Indexes
playlists
p_list_id INT
id INT
p_list_name VARCHAR(20...
Indexes
users
id INT
user_name VARCHAR(200)
user_id VARCHAR(200)
password VARCHAR(200)
email VARCHAR(200)
nickname VARCHAR(200)
user_profile VARCHAR(200)
hint_question VARCHAR(20...
hint_answer VARCHAR(200)
is_active INT
deactivated_until DATE
Indexes
songs
song_id INT
album_id INT
title VARCHAR(200)
song_path VARCHAR(20...
lyrics VARCHAR(200)
video_link VARCHAR(200)
Indexes
del_users
id INT
deleted_time TIMESTA...
Indexes
user_roles
user_id INT
roles VARCHAR(20...
Indexes
title_son...
album_id INT
song_id INT
Indexes
artist_rol...
artist_id INT
group_id INT
song_id INT
role_id INT
Indexes
artist_lik...
artist_id INT
id INT
Indexes
album_lik...
album_id INT
id INT
Indexes
group_lik...
group_id INT
id INT
Indexes
pur_users
id INT
expiration_date DA...
Indexes
playlist_songs
p_list_id INT
song_id INT
created_time TIMESTA...
Indexes
role_code
code_id INT
role_name VARCHAR(20...
role_id INT
Indexes
group_membe...
group_id INT
artist_id INT
Indexes
comments
c_id INT
song_id INT
id INT
c_text VARCHAR(500)
created_time TIMESTAMP
modified_time TIMESTA...
Indexes
song_gen...
song_id INT
genre_id INT
Indexes
genre_code
code_id INT
genre_name VARCHAR(20...
genre_id INT
Indexes
groups
group_id INT
group_name VARCHAR(200)
group_image VARCHAR(200)
group_description VARCHAR(20...
Indexes
code_master
code_id INT
code_name VARCHAR(20...
Indexes

프로젝트 진행: 대용량 데이터 준비

더미 데이터 삽입	-실제 음원 정보를 대용량으로 만들기에는 기술적으로 불가능 - MySQL의 작업 프로시저를 이용해 더미 데이터 삽입

↓

앨범 67000개, 음원 670,000개, 그룹 67000개, 아티스트 201,000명 삽입

프로젝트 진행: 검색 결과 출력

전체 검색 최적화

검색 결과 출력

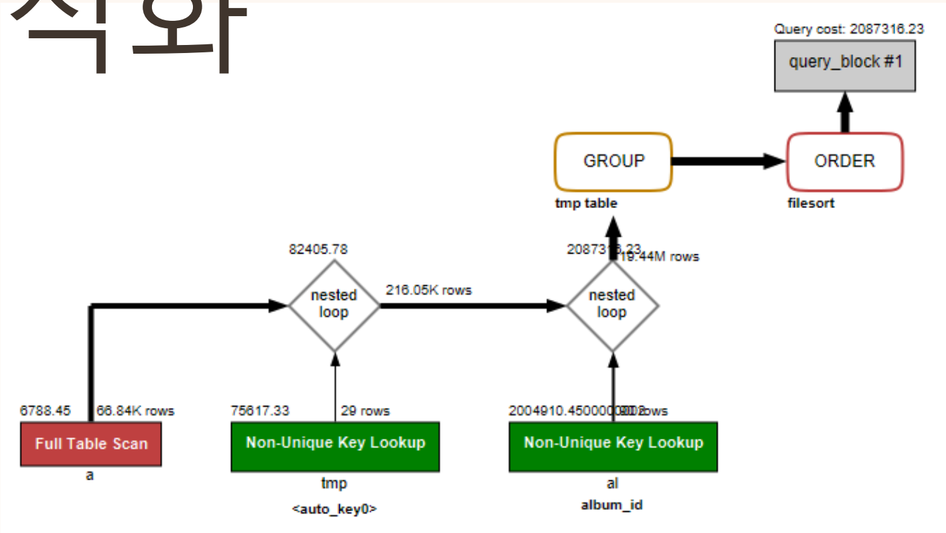
- 검색 쿼리 후 추가 데이터 가공을 통해 결과 출력.
- 아티스트가 검색된 경우: 아티스트가 참여한 모든 곡 출력.
- 그룹이 검색된 경우: 아티스트 탭에 그룹 멤버를 함께 출력.
- 앨범이 검색된 경우: (1) 앨범 수록곡을 결과에 포함
  (2) 가장 많이 참여한 아티스트 하나만 출력
  (3) 해당 아티스트가 그룹으로 참여했다면 그룹 이름으로 출력

프로젝트 진행: 검색 결과 출력(앨범)

```sql
SELECT
    a.album_id,
    a.album_name,
    a.album_image,
    a.album_type,
    a.album_release_date,
    mp.participant_name AS artist_name,
    mp.participant_id AS artist_id,
    mp.participant_type AS artist_type,
    COUNT(al.album_id) AS like_count
FROM albums a
LEFT JOIN (
    SELECT
        tmp.album_id,
        tmp.participant_name,
        tmp.participant_id,
        tmp.participant_type,
        tmp.participation_count
    FROM (
        SELECT
            s.album_id,
            COALESCE(g.group_name, art.artist_name) AS participant_name,
            COALESCE(g.group_id, art.artist_id) AS participant_id,
            CASE
                WHEN g.group_id IS NOT NULL THEN 'group'
                ELSE 'artist'
            END AS participant_type,
            COUNT(*) as participation_count,
            ROW_NUMBER() OVER (PARTITION BY s.album_id ORDER BY COUNT(*) DESC) as rn
        FROM songs s
        JOIN artist_roles ar ON s.song_id = ar.song_id
        LEFT JOIN artists art ON ar.artist_id = art.artist_id
        LEFT JOIN `groups` g ON ar.group_id = g.group_id

        GROUP BY s.album_id, participant_name, participant_id, participant_type

    ) AS tmp
    WHERE tmp.rn = 1
) mp ON a.album_id = mp.album_id
LEFT JOIN album_likes al ON a.album_id = al.album_id
WHERE a.album_name LIKE CONCAT('%', '소녀', '%')
GROUP BY a.album_id, a.album_name, a.album_image, a.album_type, a.album_release_date,
         mp.participant_name, mp.participant_id, mp.participant_type
ORDER BY a.album_name, like_count DESC
LIMIT 18 OFFSET 0;
```

- 앨범 테이블과 음원 테이블의 정보를 가져오는 쿼리
- 앨범이 검색된 경우 앨범의 수록곡을 검색 결과에 포함
- 가장 많이 등장한 아티스트를 출력
- 가장 많이 등장한 아티스트가 그룹으로 참여한 경우 그룹 이름 출력
- 검색 대상 문자열 길이가 짧은 순으로 정렬
- 문자열 길이가 동일한 경우 좋아요 개수가 많은 순으로 정렬

프로젝트 진행: 앨범 검색 쿼리 최적화



ɔum_id	album_name	album_image	album_type	album_release_date	artist_name	artist_id	artist_type	like_cou
)63	소곡집 I	잔나비 소곡집 I_cover.jpg	EP	2020-11-06	최정훈	201288	artist	0
)64	소곡집 II: 초록...	잔나비 소곡집 II.jpg	EP	2022-05-10	최정훈	201288	artist	0

2 row(s) returned 16.656 sec / 0.000 sec

풀 테이블 스캔:

- 앞 와일드카드 검색인 경우 모든 테이블을 스캔할 수 밖에 없음
- 앞 와일드카드 검색이 필요하다고 판단.
- 최적화 할 수 없는 부분

Non-Unique Key Lookup: 조건부 조인 최적화 필요

- 조인이 자주 일어나면 쿼리 코스트가 높게 책정됨
- 전체 테이블에 대한 조인이 이루어지고 있음.
  - 데이터 개수가 670,000개인 songs 테이블 JOIN
- 특정 조건에만 조인하도록 최적화 할 수 있음.

집계함수 count: 사전 집계 최적화

- 검색할 때 마다 count 함수를 실행하므로 성능에 문제가 있음
- 집계 함수 테이블(*)을 생성해 최적화 할 수 있음
  * 집계 함수를 미리 실행해 저장해두는 테이블

사전 집계 최적화: 계획

최적화 방향성 설정

사전 집계 최적화

- 정의: SUM, COUNT, AVG 등의 집계 함수 결과를 미리 계산하여 저장한 테이블을 활용.
- 장점: 쿼리 실행 시 연산 부담 감소, 성능 향상.

최적화 적용 근거

- 문제:
  - 검색 시 매번 COUNT 함수 실행, 성능 저하.
  - 정렬 시 좋아요 정보 조회로 인한 빈번한 JOIN 발생.
- 해결:
  - 좋아요 통계 테이블 생성: 앨범, 음원, 아티스트 등 다양한 좋아요 집계 데이터 저장.

사전 집계 최적화:적용

집계 테이블 적용 방법

적용 방법 (집계 테이블 데이터)	• 구성: TYPE, ITEM_ID, COUNT 컬럼 - type: 데이터 출처 (앨범, 음원 등) - item_id: 가져온 열의 데이터의 PK 아이디 - count: 가져온 열의 좋아요 개수 • 업데이트: 테이블 변경시 실시간 반영, 이용자가 많을 경우 새벽 시간대 업데이트 가능.

사전 집계 최적화:테이블 구조

집계 테이블 저장 예시

stats_id	type	item_id	like_count
1	album	90	91
91	song	90	90
181	group	90	90
271	artist	90	90
364	album	2	0
366	song	19	1
370	song	14	1
371	song	12	1
372	song	17	1
374	song	11	1

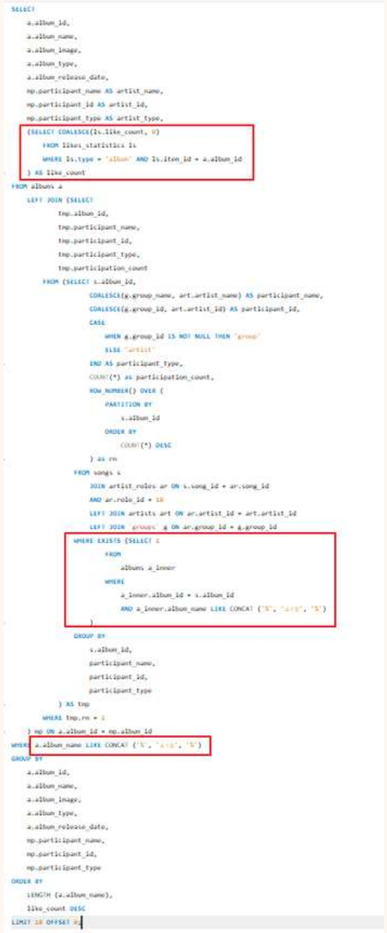
조건부 JOIN 최적화: 계획

최적화 계획

- JOIN 조건 설정
- EXISTS 함수를 이용해 설정한 조건이 TRUE인 경우만 JOIN

keyword로 검색된 결과만 JOIN이 되도록 조건 설정

조건부 JOIN 최적화: 최적화 적용 쿼리

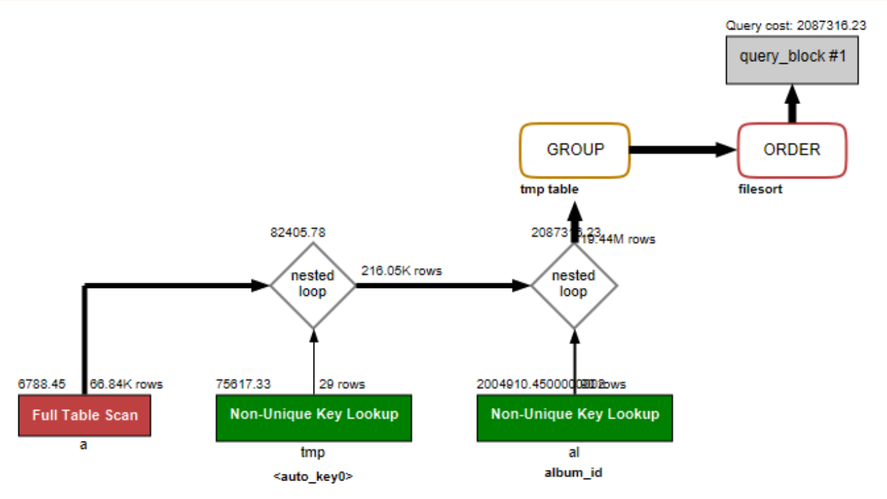


```
SELECT
    a.album_id,
    a.album_name,
    a.album_image,
    a.album_type,
    a.album_release_date,
    mp.participant_name AS artist_name,
    mp.participant_id AS artist_id,
    mp.participant_type AS artist_type,
    (SELECT COALESCE(lc.like_count, 0)
        FROM likes_statistics lc
        WHERE lc.type = 'album' AND lc.item_id = a.album_id
    ) AS like_count
FROM albums a
    LEFT JOIN (SELECT
            tmp.album_id,
            tmp.participant_name,
            tmp.participant_id,
            tmp.participant_type,
            tmp.participation_count
        FROM (SELECT s.album_id,
                COALESCE(g.group_name, art.artist_name) AS participant_name,
                COALESCE(g.group_id, art.artist_id) AS participant_id,
                CASE
                    WHEN g.group_id IS NOT NULL THEN 'group'
                    ELSE 'artist'
                END AS participant_type,
                COUNT(*) as participation_count,
                ROW_NUMBER() OVER (
                    PARTITION BY
                        s.album_id
                    ORDER BY
                        COUNT(*) DESC
                ) as rn
            FROM songs s
                JOIN artist_roles ar ON s.song_id = ar.song_id
                AND ar.role_id = 10
                LEFT JOIN artists art ON ar.artist_id = art.artist_id
                LEFT JOIN `groups` g ON ar.group_id = g.group_id
            WHERE EXISTS (SELECT 1
                    FROM
                        albums a_inner
                    WHERE
                        a_inner.album_id = s.album_id
                        AND a_inner.album_name LIKE CONCAT ('%', '아이', '%')
                )
            GROUP BY
                s.album_id,
                participant_name,
                participant_id,
                participant_type
        ) AS tmp
        WHERE tmp.rn = 1
    ) mp ON a.album_id = mp.album_id
WHERE a.album_name LIKE CONCAT ('%', '아이', '%')
GROUP BY
    a.album_id,
    a.album_name,
    a.album_image,
    a.album_type,
    a.album_release_date,
    mp.participant_name,
    mp.participant_id,
    mp.participant_type
ORDER BY
    LENGTH (a.album_name),
    like_count DESC
LIMIT 10 OFFSET 0;
```

- Exists 함수로 해당 조건이 만족할 경우만 JOIN
- 서브 쿼리를 사용해 데이터를 가져옴
- 직접 JOIN문이 사라짐

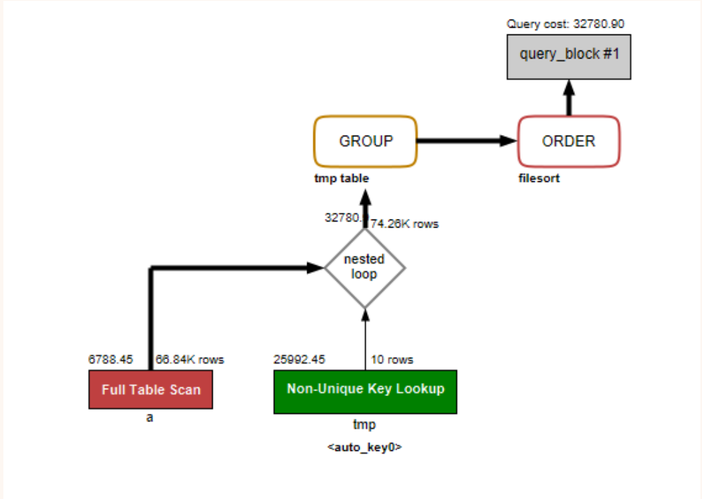
최적화 결과: 성능 비교

사전 집계 최적화 및 조건부 JOIN 최적화 적용 효과



ɔum_id	album_name	album_image	album_type	album_release_date	artist_name	artist_id	artist_type	like_cou
)63	소곡집 I	잔나비 소곡집 I_cover.jpg	EP	2020-11-06	최정훈	201288	artist	0
)64	소곡집 II: 초록...	잔나비 소곡집 II.jpg	EP	2022-05-10	최정훈	201288	artist	0

2 row(s) returned 16.656 sec / 0.000 sec



album_id	album_name	album_image	album_type	album_release_date	artist_name	artist_id	artist_type	like_count
67063	소곡집 I	잔나비 소곡집 I_cover.jpg	EP	2020-11-06	잔나비	201287	artist	NULL
67064	소곡집 II: 초록...	잔나비 소곡집 II.jpg	EP	2022-05-10	잔나비	201287	artist	NULL

2 row(s) returned 0.078 sec / 0.000 sec

최적화 결과: 적용 효과

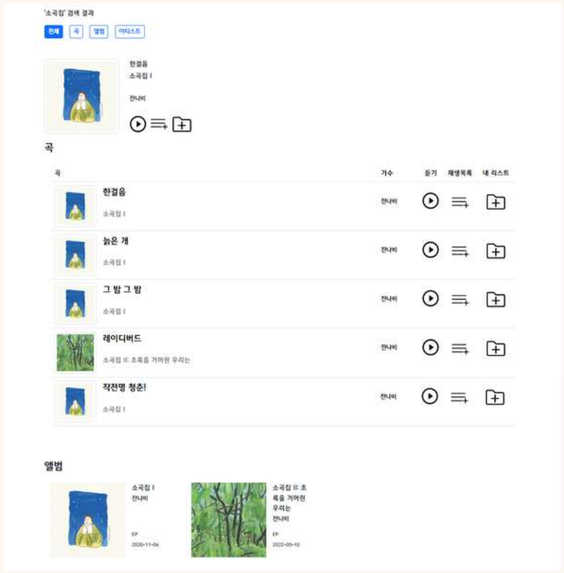
사전 집계 최적화 및 조건부 JOIN 최적화 적용 효과

최적화 적용 효과	• 쿼리 실행 시간 단축 (16.66 S -> 0.078 S) • 쿼리 코스트 감소 (2,087,816 -> 32,780)

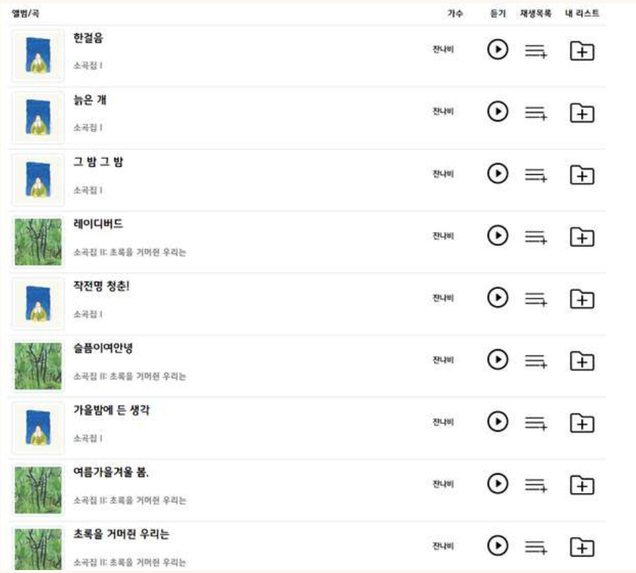
최적화 결과: 앨범 검색 구현

앨범 검색 출력 구현: “소곡집” 검색 결과

통합 검색



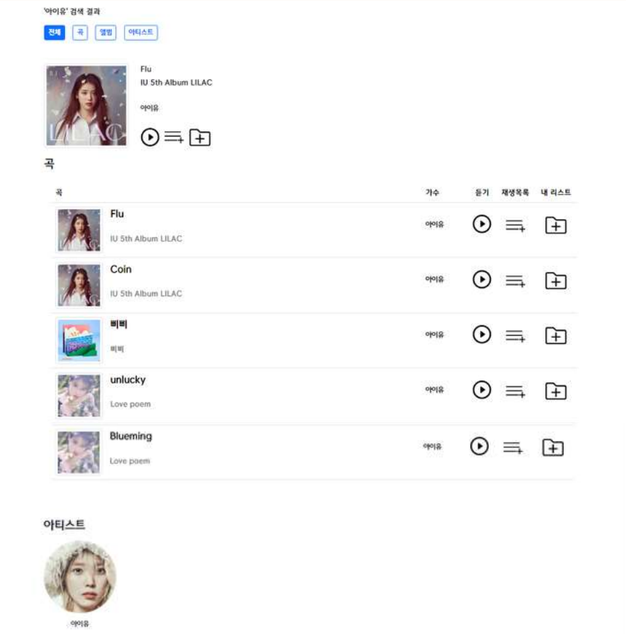
곡 카테고리



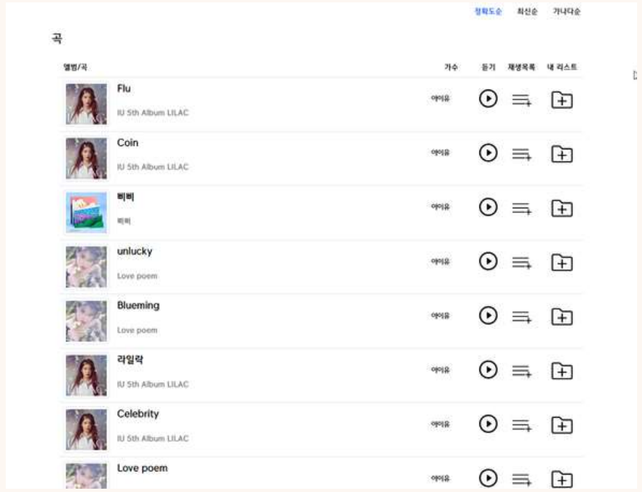
최적화 결과: 아티스트 검색 구현

아티스트 검색 출력 구현: “아이유” 검색 결과

통합 출력



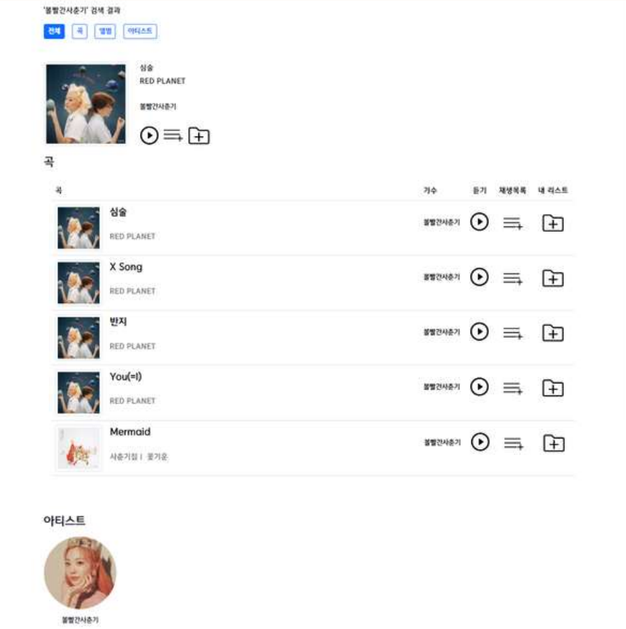
곡 카테고리



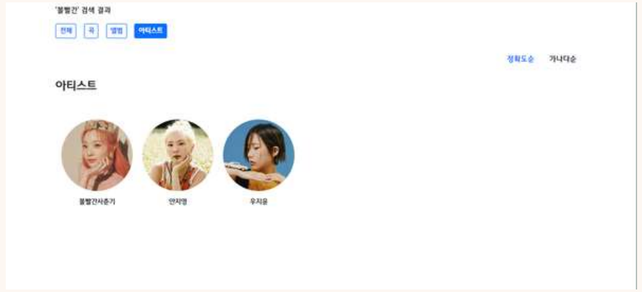
최적화 결과: 그룹 검색 구현

그룹 검색 출력 구현: “볼빨간 사춘기” 검색

통합 출력



아티스트 카테고리



초성 검색: 구현 방법 2 가지

MySQL 함수 사용을 통한 초성 검색	• MYSQL의 함수를 이용해 테이블의 컬럼을 초성으로 변환 후 검색.
초성 테이블을 미리 생성하여 검색	• 자바 초성 변환 로직을 구현해 초성으로 변환된 테이블을 생성하고 이를 검색.

초성 검색: 구현 방법 2 가지 (1)

방법 1:

MySQL
함수 사용을 통한
초성 검색

- MYSQL의 함수를 이용해 테이블의 컬럼을 초성으로 변환 후 검색.

- 장점:
  - MySQL에서 바로 초성 검색 가능.
  - 간단한 구현.

- 단점:
  - 데이터가 많아질수록 함수 실행 속도가 느려짐.
  - MySQL에만 의존적인 방법.

초성 검색: 구현 방법 2 가지 (2)

방법 2:

초성 테이블을 미리 생성하여 검색

- 초성 변환 로직을 사용해 초성으로 변환된 테이블을 생성하고 이를 검색.
- 쿼리: 초성 테이블을 통해 검색.
- 장점:
  - 빠른 검색 성능.
  - 대량의 데이터데이터를 효율적으로 검색
  - DB 의존성 줄일 수 있음.
- 단점:
  - 초성 테이블을 미리 생성하고 유지해야 함.
  - 구현 복잡성 증가.

초성 검색: 선택한 방법

최종 선택:
방법 2
(초성 테이블 사용)

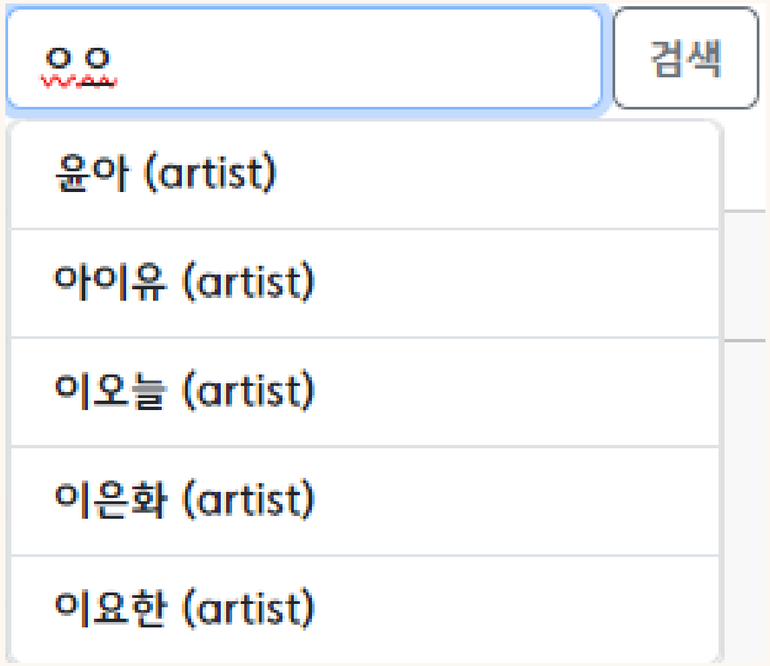
- 선택 이유:
  - 검색 성능이 더 높음
  - MySQL 초성 변환 함수는 데이터 양이 많아질수록 성능 저하.
  - MySQL에만 의존하지 않고 다른 DB에서도 적용 가능.

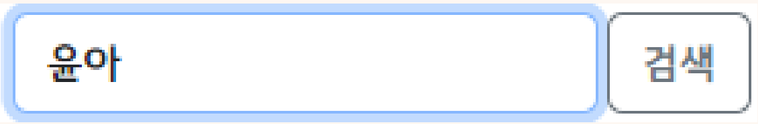
초성 검색: 초성 검색 쿼리

```
SELECT
    'album' AS type,
    a.album_id AS id,
    a.album_name AS name,
    a.album_release_date AS release_date,
    COALESCE(al.like_count, 0) AS like_count -- 앨범 좋아요 개수 가져오기
FROM albums a
JOIN albums_first_con afc ON a.album_id = afc.album_id
LEFT JOIN album_likes al ON a.album_id = al.album_id -- 좋아요 데이블을 LEFT JOIN
WHERE afc.album_first_con_name LIKE CONCAT('%', 'ㅇㄴ', '%')

UNION ALL

SELECT
    'song' AS type,
    s.song_id AS id,
    s.title AS name,
    NULL AS release_date,
    COALESCE(sl.like_count, 0) AS like_count -- 곡 좋아요 개수 가져오기
FROM songs s
JOIN songs_first_con sfc ON s.song_id = sfc.song_id
LEFT JOIN likes sl ON s.song_id = sl.song_id -- 좋아요 데이블을 LEFT JOIN
WHERE sfc.song_first_con_name LIKE CONCAT('%', 'ㅇㄴ', '%')

UNION ALL

SELECT
    'group' AS type,
    g.group_id AS id,
    g.group_name AS name,
    NULL AS release_date,
    COALESCE(gl.like_count, 0) AS like_count -- 그룹 좋아요 개수 가져오기
FROM `groups` g
JOIN groups_first_con gfc ON g.group_id = gfc.group_id
LEFT JOIN group_likes gl ON g.group_id = gl.group_id -- 좋아요 데이블을 LEFT JOIN
WHERE gfc.group_first_con_name LIKE CONCAT('%', 'ㅇㄴ', '%')

UNION ALL

SELECT
    'artist' AS type,
    art.artist_id AS id,
    art.artist_name AS name,
    NULL AS release_date,
    COALESCE(artl.like_count, 0) AS like_count -- 아티스트 좋아요 개수 가져오기
FROM artists art
JOIN artists_first_con afc ON art.artist_id = afc.artist_id
LEFT JOIN artist_likes artl ON art.artist_id = artl.artist_id -- 좋아요 데이블을 LEFT JOIN
WHERE afc.artist_first_con_name LIKE CONCAT('%', 'ㅇㄴ', '%')

ORDER BY
    LENGTH(name),  -- name(혹은 title) 짧은게 데이터 상위로 정렬
    like_count DESC;  -- 같은 검색일 경우 좋아요 개수 높은 순으로 정렬
```

- 생성한 초성 테이블에서 정보를 가져옴
- 4개의 테이블에 대해서 각 각 검색
- UNION ALL 을 이용해 각 검색 결과를 이어 붙임
- 문자열 길이와 좋아요 개수를 이용해 정렬
- exists와 비슷한 역할을 하는 서브 쿼리로 조건부 JOIN 최적화 적용

초성 검색: 초성 검색 구현


ㅇㅇ
검색
윤아 (artist)
아이유 (artist)
이오늘 (artist)
이은화 (artist)
이요한 (artist)


윤아
검색

그룹 테이블 추가: 문제 상황

문제 상황: 그룹 활동과 솔로 활동 아티스트 처리

문제	• 그룹 멤버가 솔로로도 활동하거나 작곡, 작사에 참여하는 경우, 아티스트 페이지에서 일관성 있게 정보가 출력되지 않음. • 예: 안지영(볼빨간 사춘기 멤버)이 작곡에 참여했을 때, 그룹과 솔로 활동을 명확히 구분하지 못함.
해결 방안	• 그룹 테이블 생성: 그룹 정보를 저장하는 테이블 추가. • 그룹 멤버 테이블 생성: 그룹 멤버와 개별 활동을 연계. • 아티스트 역할 테이블 수정: 그룹으로 참여한 지 저장하는 컬럼 추가

그룹 테이블 추가: 기존 테이블 구조

문제 상황: 그룹 활동과 솔로 활동 아티스트 처리

볼빨간 사춘기 (멤버: 안지영, 우지윤)

트랙	곡명	재생시간	작사	작곡	편곡
01	우주를 줄게 TITLE	3:33	안지영, 우지윤	안지영, 바닐라맨	바닐라맨
02	싸운날	3:09	안지영		
03	You(=I)	2:49			
04	심술	2:37			
05	나만 안되는 연애 TITLE	3:39			
06	조울릿	3:10	안지영, 우지윤	안지영, 바닐라맨	
07	프리지아	3:06	안지영		황중하
08	X Song	3:46	바닐라맨		바닐라맨
09	반지	2:56	안지영, 우지윤		
10	사랑에 빠졌을 때	3:52	안지영		

볼빨간 사춘기와 멤버를 연결하는 관계 X

그룹 테이블 추가: 추가 과정

적용 과정

- 그룹 테이블 생성: 그룹 정보를 저장하는 테이블 추가.
- 그룹 멤버 테이블 생성: 그룹 멤버와 개별 활동을 연계.
- 아티스트 역할 테이블 수정: 그룹으로 참여한 지 저장하는 컬럼 추가

그룹 테이블 데이터 예시

group_id	group_name	group_description	group_image
67001	ABBA	ABBA.text	ABBA.jpg
67002	봄빨간사춘기	봄빨간사춘기설명.txt	봄빨간사춘기사진.jpg
67003	AKMU (악뮤)	악뮤설명.txt	악뮤사진.jpg
67004	NewJeans	NewJeans 설명	NewJeans사진.jpg
67005	kissOfLife	kissOfLife 설명	kissOfLife사진.jpg
67006	babymonster	babymonster 설명	babymonster사진.jpg
67007	illit	illit 설명	illit사진.jpg
67008	VIVIZ	VIVIZ 설명	VIVIZ사진.jpg

그룹 멤버 테이블 데이터 예시

group_id	artist_id
67002	201270
67002	201450
67003	201271
67003	201451

아티스트 역할 테이블 데이터 예시

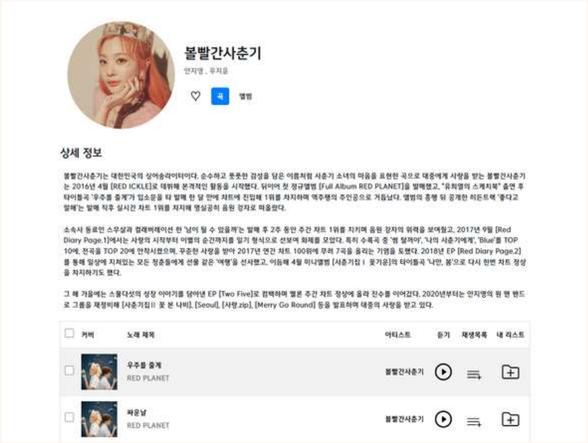
artist_id	group_id	song_id	role_id
201270	67002	670217	10
201270	67002	670218	10
201270	67002	670219	10
201270	67002	670220	10
201270	67002	670221	10
201270	67002	670222	10
201270	67002	670223	10
201270	67002	670224	10

그룹 테이블 추가: 적용 결과

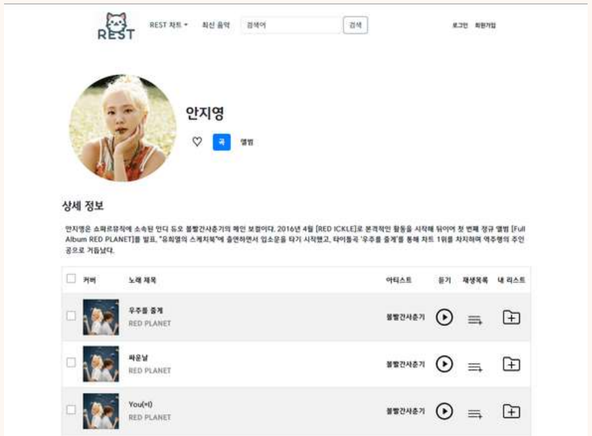
적용 결과

- 아티스트 페이지: 모든 활동(그룹 및 솔로)에서의 참여 음원이 정확히 표시됨.
- 그룹 페이지: 그룹 멤버 및 그룹 앨범 정보 정확히 출력.

그룹 페이지



아티스트 페이지



시현

프로젝트 시현

프로젝트 주요 성과

주요 성과	• 100만 건에 대한 검색 기능 구현 • 초성 검색 로직 구현 • 그룹 테이블 및 그룹 페이지 구현 • 아티스트 페이지의 참여 음원 정보 정확성 향상

프로젝트 개선 방향

외부 검색 엔진	• 데이터가 더 늘어난 경우 LIKE 검색은 한계가 있음. • 성능 좋은 알고리즘을 사용한 검색 기능을 이용할 수 있음.
차트 페이지 최적화	• 음원 차트 페이지의 쿼리 최적화는 개발 시간 부족으로 하지 못함 • 음원 데이터를 불러올 때 대기 시간이 생김 • 차트 페이지에서 사용하는 쿼리를 최적화 하여 대기 시간을 줄여야 함.

Q & A

질문과 답변

프로젝트에 대해 궁금한 점은 질문해주세요

THANK YOU

감사합니다

발표자 김영웅