취향 맞춤 추천 알고리즘 최적화

1. 기존 구현 방향

1-1. 기존 Query

```
@Query(
        value = "WITH user_specialities AS (" +
                "SELECT sm.speciality_id " +
                "FROM speciality_mappings sm " +
                "WHERE sm.user_id = :userId" +
                ") , liked_tags AS (" +
                "SELECT DISTINCT tm.tag_id " +
                "FROM user_specialities us " +
                "JOIN tag_maps tm ON tm.specialty_id = us.speciality_id " +
                "WHERE tm.weight = 1" +
                ") , disliked_tags AS (" +
                "SELECT DISTINCT tm.tag_id " +
                "FROM user_specialities us " +
                "JOIN tag_maps tm ON tm.specialty_id = us.speciality_id " +
                "WHERE tm.weight = 0" +
                ") , liked_count AS (" +
                "SELECT COUNT(*) AS cnt FROM liked_tags" +
                ") , disliked_count AS (" +
                "SELECT COUNT(*) AS cnt FROM disliked_tags" +
                ") " +
                "SELECT r.restaurant_id AS restaurantId, " +
                "COALESCE(1.0 * COUNT(DISTINCT lt.tag_id) / " +
                "NULLIF(lc.cnt + COUNT(DISTINCT rtm.tag_id) - COUNT(DISTINCT lt.tag_id), 0), 0)
AS likeScore, " +
                "COALESCE(1.0 * COUNT(DISTINCT dt.tag_id) / NULLIF(dc.cnt, 0), 0) AS dislikePen
alty, " +
                "COALESCE(1.0 * COUNT(DISTINCT lt.tag_id) / " +
                "NULLIF(lc.cnt + COUNT(DISTINCT rtm.tag_id) - COUNT(DISTINCT lt.tag_id), 0), 0)
" +
                "- 0.4 * COALESCE(1.0 * COUNT(DISTINCT dt.tag_id) / NULLIF(dc.cnt, 0), 0) AS fin
alScore " +
                "FROM restaurants r " +
                "LEFT JOIN restaurant_tag_maps rtm ON r.restaurant_id = rtm.restaurant_id " +
                "LEFT JOIN liked_tags lt ON lt.tag_id = rtm.tag_id " +
                "LEFT JOIN disliked_tags dt ON dt.tag_id = rtm.tag_id " +
                "CROSS JOIN liked_count lc " +
                "CROSS JOIN disliked_count dc " +
```

```
            "WHERE r.status = 'OPEN' " +
            "AND (lc.cnt + dc.cnt) > 0 " +
            "GROUP BY r.restaurant_id, lc.cnt, dc.cnt " +
            "ORDER BY finalScore DESC",
        nativeQuery = true
    )
    List<RestaurantSimilarityProjection> findRestaurantsByUserTagSimilarity(
        @Param("userId") Long userId
    );
```

1-2. 기존 Query의 개선할 점

1. **DISTINCT COUNT 최소화**

- 현재 `COUNT(DISTINCT rtm.tag_id)` / `COUNT(DISTINCT lt.tag_id)` / `COUNT(DISTINCT dt.tag_id)`가 반복됨
- `restaurant_id`별 태그를 먼저 집계한 서브쿼리로 바꿔서 DISTINCT 연산 횟수를 줄이는 게 가장 큼

2. **CTE를 줄이거나 재사용**

- `liked_tags` / `disliked_tags`는 한번 만들고 계속 쓰지만,
  MySQL에서는 CTE가 "임시 테이블화"되면서 느려질 수 있음
- 서브쿼리로 합치거나 캐시(임시 테이블)로 빼면 개선

3. **조건 필터가 너무 늦음**

- `WHERE r.status = 'OPEN' AND (lc.cnt + dc.cnt) > 0`가 마지막에 적용됨
- 이 조건을 더 앞에서 걸어 조인 대상 자체를 줄이기가 중요

4. **CROSS JOIN 최소화**

- `liked_count/disliked_count`를 CROSS JOIN 하고 있는데,
  상수처럼 쓰이는 값이라 변수(파라미터)로 미리 계산해두면 더 좋음
  → 서비스에서 liked/disliked count를 먼저 구해서 쿼리에 전달하면 계산 줄어듦

2. 최적화 진행

2-1. index 설정

```
show index from tag_maps;

CREATE INDEX idx_tag_maps_specialty_weight ON tag_maps (specialty_id, weight);
CREATE INDEX idx_tag_maps_tag_id ON tag_maps (tag_id);
```

2-2. DISTINCT COUNT 최소화

과정

1. 식당별 태그 집계를 먼저 만들기 = `restaurant_tag_maps`를 먼저 `restaurant_id` 기준으로 한 번만 집계
2. 집계 결과에 `liked_tags` / `disliked_tags`를 조인해서 점수 계산

결과

- 메인 쿼리에서 `COUNT(DISTINCT ...)`가 3번 이상 반복되는 문제 해결
- 먼저 restaurant_id별 태그 수/매칭 수를 계산한 서브쿼리를 만들고, 그 값을 그대로 쓰면 DISTINCT가 크게 줄어듦

```
WITH user_tags AS (
  SELECT sm.speciality_id
  FROM speciality_mappings sm
  WHERE sm.user_id = :userId
),
liked_tags AS (
  SELECT DISTINCT tm.tag_id
  FROM user_tags us
  JOIN tag_maps tm ON tm.specialty_id = us.speciality_id
  WHERE tm.weight = 1
),
disliked_tags AS (
  SELECT DISTINCT tm.tag_id
  FROM user_tags us
  JOIN tag_maps tm ON tm.specialty_id = us.speciality_id
  WHERE tm.weight = 0
),
restaurant_tag_agg AS (
  SELECT
    rtm.restaurant_id,
    COUNT(*) AS restaurant_tag_count,
    SUM(CASE WHEN lt.tag_id IS NOT NULL THEN 1 ELSE 0 END) AS liked_match_count,
    SUM(CASE WHEN dt.tag_id IS NOT NULL THEN 1 ELSE 0 END) AS disliked_match_count
  FROM restaurant_tag_maps rtm
  LEFT JOIN liked_tags lt ON lt.tag_id = rtm.tag_id
  LEFT JOIN disliked_tags dt ON dt.tag_id = rtm.tag_id
SELECT
  r.restaurant_id,
  COALESCE(1.0 * liked_match_count /
    NULLIF(liked_count + restaurant_tag_count - liked_match_count, 0), 0) AS likeScore,
  COALESCE(1.0 * disliked_match_count / NULLIF(disliked_count, 0), 0) AS dislikePenalty,
  COALESCE(1.0 * liked_match_count /
```

```
    NULLIF(liked_count + restaurant_tag_count - liked_match_count, 0), 0)
    - 0.4 * COALESCE(1.0 * disliked_match_count / NULLIF(disliked_count, 0), 0) AS finalScore
FROM restaurants r
JOIN restaurant_tag_agg a ON a.restaurant_id = r.restaurant_id
CROSS JOIN (SELECT COUNT(*) AS liked_count FROM liked_tags) lc
CROSS JOIN (SELECT COUNT(*) AS disliked_count FROM disliked_tags) dc
WHERE r.status = 'OPEN'
ORDER BY finalScore DESC;
```

→ `restaurant_tag_agg`에서 COUNT와 매칭수를 미리 집계해서 메인 쿼리에서 DISTINCT를 거의 없앨 수 있음

2-3. 조건 필터 위치 수정

과정

1. OPEN인 식당 필터링을 `restaurant_tag_agg`에서 먼저 집계
2. 메인 쿼리에서 `r.status` 조건을 빼서 쿼리문 개선

결과

- `restaurant_tag_agg` 안에 `JOIN restaurants r ... WHERE r.status = 'OPEN'` 추가
- 메인 쿼리 `WHERE r.status = 'OPEN'` 제거

→ 조인 대상 자체가 줄어서 더 빠르게 동작함

```
WITH user_specialities AS (
  SELECT sm.speciality_id
  FROM speciality_mappings sm
  WHERE sm.user_id = :userId
),
liked_tags AS (
  SELECT DISTINCT tm.tag_id
  FROM user_specialities us
  JOIN tag_maps tm ON tm.specialty_id = us.speciality_id
  WHERE tm.weight = 1
),
disliked_tags AS (
  SELECT DISTINCT tm.tag_id
  FROM user_specialities us
  JOIN tag_maps tm ON tm.specialty_id = us.speciality_id
  WHERE tm.weight = 0
),
liked_count AS (
  SELECT COUNT(*) AS cnt FROM liked_tags
),
disliked_count AS (
```

```
  SELECT COUNT(*) AS cnt FROM disliked_tags
),
restaurant_tag_agg AS (
  SELECT
    rtm.restaurant_id,
    COUNT(*) AS restaurant_tag_count,
    SUM(CASE WHEN lt.tag_id IS NOT NULL THEN 1 ELSE 0 END) AS liked_match_count,
    SUM(CASE WHEN dt.tag_id IS NOT NULL THEN 1 ELSE 0 END) AS disliked_match_count
  FROM restaurant_tag_maps rtm
  JOIN restaurants r ON r.restaurant_id = rtm.restaurant_id
  LEFT JOIN liked_tags lt ON lt.tag_id = rtm.tag_id
  LEFT JOIN disliked_tags dt ON dt.tag_id = rtm.tag_id
  WHERE r.status = 'OPEN'
  GROUP BY rtm.restaurant_id
)
SELECT
  r.restaurant_id AS restaurantId,
  COALESCE(
    1.0 * rta.liked_match_count /
    NULLIF(lc.cnt + rta.restaurant_tag_count - rta.liked_match_count, 0),
    0
  ) AS likeScore,
  COALESCE(
    1.0 * rta.disliked_match_count / NULLIF(dc.cnt, 0),
    0
  ) AS dislikePenalty,
  COALESCE(
    1.0 * rta.liked_match_count /
    NULLIF(lc.cnt + rta.restaurant_tag_count - rta.liked_match_count, 0),
    0
  ) - 0.4 * COALESCE(
    1.0 * rta.disliked_match_count / NULLIF(dc.cnt, 0),
    0
  ) AS finalScore
FROM restaurants r
JOIN restaurant_tag_agg rta ON rta.restaurant_id = r.restaurant_id
CROSS JOIN liked_count lc
CROSS JOIN disliked_count dc
WHERE (lc.cnt + dc.cnt) > 0
ORDER BY finalScore DESC;
```

2-4. Cross Join 최소화

필요성

`liked_count` / `disliked_count`를 CTE로 계산하면 DB가 내부적으로 임시 테이블을 만들거나, 같은 값을 쿼리 실행 중 반복 참조하게 됨

- 서비스에서 먼저 계산하면, **메인 쿼리는 상수 값만 쓰는 단순 계산이 되어 옵티마이저가 더 단순하게 실행** 가능
- MySQL에서는 CTE가 무조건 최적화되지 않고 **materialize**될 수 있어서 작은 계산이라도 비용이 생길 수 있음

→ **사용자 태그가 많거나, 추천 쿼리가 자주 호출되거나, DB 부하가 높은 경우 Cross Join 최소화**가 효율적임

특히, LunchGo는 실시간 예약 서비스라 동시에 많은 접속자가 생기는 경우가 많기 때문에 더욱 효율적일 것으로 예상

과정

1. `liked_count`, `disliked_count`를 서비스에서 먼저 구해서 파라미터로 넘기도록 수정
2. 해당 카운트 호출 후 `findRestaurantsByUserTagSimilarity(userId, likedCount, dislikedCount)` 실행

```
@Query(
        value = "SELECT COUNT(DISTINCT tm.tag_id) " +
                "FROM speciality_mappings sm " +
                "JOIN tag_maps tm ON tm.specialty_id = sm.speciality_id " +
                "WHERE sm.user_id = :userId AND tm.weight = 1",
        nativeQuery = true
    )
    int countLikedTagsByUserId(@Param("userId") Long userId);

    @Query(
        value = "SELECT COUNT(DISTINCT tm.tag_id) " +
                "FROM speciality_mappings sm " +
                "JOIN tag_maps tm ON tm.specialty_id = sm.speciality_id " +
                "WHERE sm.user_id = :userId AND tm.weight = 0",
        nativeQuery = true
    )
    int countDislikedTagsByUserId(@Param("userId") Long userId);
```

```
WITH user_specialities AS (
    SELECT sm.speciality_id
    FROM speciality_mappings sm
    WHERE sm.user_id = :userId
  ),
  liked_tags AS (
    SELECT DISTINCT tm.tag_id
    FROM user_specialities us
    JOIN tag_maps tm ON tm.specialty_id = us.speciality_id
    WHERE tm.weight = 1
  ),
  disliked_tags AS (
    SELECT DISTINCT tm.tag_id
```

```
  FROM user_specialities us
  JOIN tag_maps tm ON tm.specialty_id = us.speciality_id
  WHERE tm.weight = 0
),
restaurant_tag_agg AS (
  SELECT
    rtm.restaurant_id,
    COUNT(*) AS restaurant_tag_count,
    SUM(CASE WHEN lt.tag_id IS NOT NULL THEN 1 ELSE 0 END) AS liked_match_count,
    SUM(CASE WHEN dt.tag_id IS NOT NULL THEN 1 ELSE 0 END) AS disliked_match_count
  FROM restaurant_tag_maps rtm
  JOIN restaurants r ON r.restaurant_id = rtm.restaurant_id
  LEFT JOIN liked_tags lt ON lt.tag_id = rtm.tag_id
  LEFT JOIN disliked_tags dt ON dt.tag_id = rtm.tag_id
  WHERE r.status = 'OPEN'
  GROUP BY rtm.restaurant_id
)
SELECT
  r.restaurant_id AS restaurantId,
  COALESCE(
    1.0 * rta.liked_match_count /
    NULLIF(:likedCount + rta.restaurant_tag_count - rta.liked_match_count, 0),
    0
  ) AS likeScore,
  COALESCE(
    1.0 * rta.disliked_match_count / NULLIF(:dislikedCount, 0),
    0
  ) AS dislikePenalty,
  COALESCE(
    1.0 * rta.liked_match_count /
    NULLIF(:likedCount + rta.restaurant_tag_count - rta.liked_match_count, 0),
    0
  ) - 0.4 * COALESCE(
    1.0 * rta.disliked_match_count / NULLIF(:dislikedCount, 0),
    0
  ) AS finalScore
FROM restaurants r
JOIN restaurant_tag_agg rta ON rta.restaurant_id = r.restaurant_id
WHERE (:likedCount + :dislikedCount) > 0
ORDER BY finalScore DESC
LIMIT 50;
```

2-5. 필터링 후 추천 순위 변화 둔감 문제 해결

기존 스코어링 방식

```
finalScore = likeScore - 0.4 * dislikePenalty
```

현재 스코어링 방식은 **스코어 스케일이 너무 완만**하여 특이사항을 6개 등록한 사람의 응답 결과를 확인했을 때, 0.2 이상의 finalScore가 거의 나오지 않았음

⇒ 순위 변화가 더욱 **민감**하게 바뀌도록 설정 필요

스코어링 방식 개선 방향

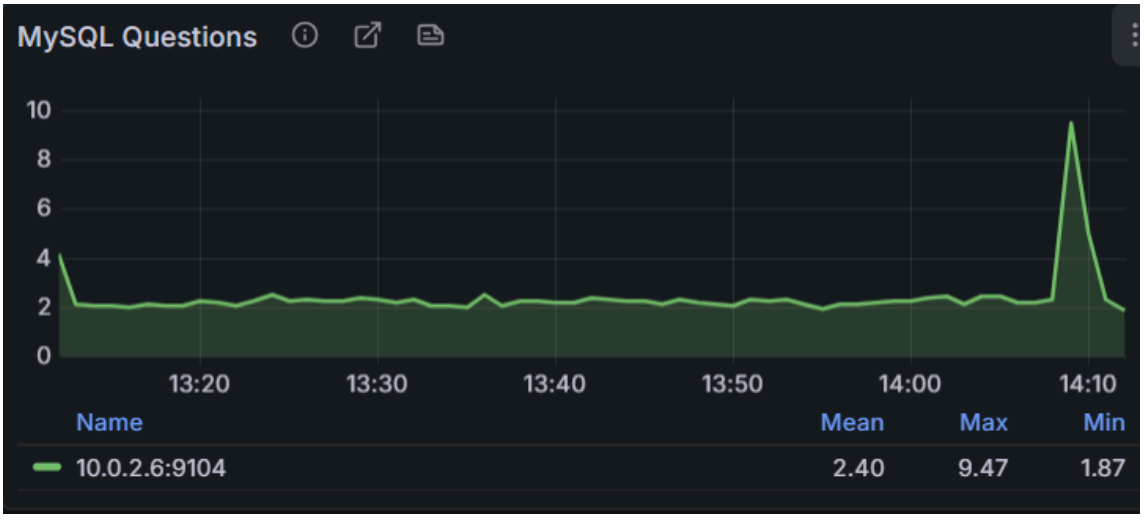
- `좋아요 태그`를 선택하면, **비선형(제곱)** 형태로 크게 키우고, `싫어요 태그`가 있으면 **큰 감점(0.7)**으로 즉시 하향
- likeScore이 소수로 나오는데, 이를 제곱하여 작아질 수 있으며 정수 값으로 확실하게 구분하기 위해 큰 값을 곱함

→ 더욱 민감한 순위변화를 만들 수 있는 방법이라고 생각

```
finalScore =
  100 * POW(likeScore, 2)
  - CASE WHEN rta.disliked_match_count > 0 THEN 30 ELSE 0 END
  - 80 * dislikePenalty
```

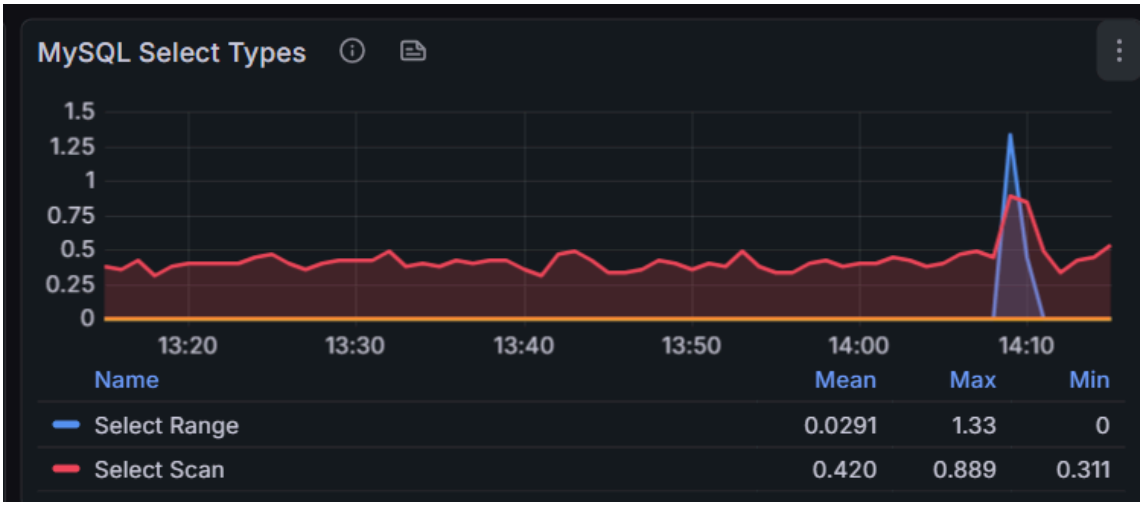
3. 최적화 테스트

3-1. 기존 쿼리문을 이용한 테스트 결과



초당 요청된 SQL 쿼리수가 평소엔 2-4이다가, 테스트 시간인 2시 9분 근처에서 거의 10까지 도달

→ 기존의 api 요청은 **평균적으로 6-7번의 쿼리**를 처리하고 있음을 알 수 있음



api 요청 시간인 2시 9분 근처에서, select Range가 커지지만 select Scan 역시 커짐

→ **index 수정**이 필요함을 알 수 있음

3-2. 수정 후 쿼리문을 이용한 테스트 결과


```
[
    {
        "restaurantId": 88,
        "likeScore": 0.1,
        "dislikePenalty": 0.0,
        "finalScore": 1.0000000000000002
    },
    {
        "restaurantId": 79,
        "likeScore": 0.08333,
        "dislikePenalty": 0.0,
        "finalScore": 0.6944444388888888
    },
    {
        "restaurantId": 80,
        "likeScore": 0.07692,
        "dislikePenalty": 0.0,
        "finalScore": 0.5917159621301777
    },
    {
```


⇒ 응답 결과는 1에 가까운 정수 값으로 잘 보이는 것을 확인할 수 있었음

테스트 시간인 3시 5분을 기준으로, 이전의 쿼리수가 **7**까지 낮아짐

→ 평균 쿼리수가 **4-5번**으로 낮아짐을 확인할 수 있음

select 범위는 index를 tag_maps라는 테이블에만 수정했으므로 크게 변화는 없지만, 더욱 row 수가 많아지면 큰 변화가 생길 것으로 보임