SQL #6 AI 서비스 리뷰 시스템

1. 문제

각 기획자의 평균 평점과 리뷰 수를 계산하고, 리뷰 수가 2개 이상인 사람 중에서 평점이 높은 순으로 랭킹 정리

2. 쿼리 1,2,3,4 수행

쿼리 1 (CTE + ROW_NUMBER())

```
EXPLAIN ANALYZE
WITH creator_stats AS (
  SELECT
    c.creator_id,
    c.creator_name,
    AVG(r.rating) AS avg_rating,
    COUNT(*) AS review_count
  FROM ai_service_creators c
  JOIN ai_service_reviews r ON c.creator_id = r.creator_id
  GROUP BY c.creator_id, c.creator_name
  HAVING COUNT(*) >= 2
),
ranked AS (
  SELECT *,
        ROW_NUMBER() OVER (ORDER BY avg_rating DESC) AS rank
  FROM creator_stats
)
SELECT * FROM ranked;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	WindowAgg (cost=104.41..111.01 rows=377 width=84) (actual time=0.574..0.581 rows=4 loops=1)
2	-> Sort (cost=104.41..105.35 rows=377 width=76) (actual time=0.562..0.564 rows=4 loops=1)
3	Sort Key: creator_stats.avg_rating DESC
4	Sort Method: quicksort Memory: 25kB
5	-> Subquery Scan on creator_stats (cost=71.33..88.28 rows=377 width=76) (actual time=0.105..0.121 rows=4 loo…
6	-> HashAggregate (cost=71.33..88.28 rows=377 width=76) (actual time=0.104..0.118 rows=4 loops=1)
7	Group Key: c.creator_id
8	Filter: (count(*) >= 2)
9	Batches: 1 Memory Usage: 73kB
10	-> Hash Join (cost=38.58..62.85 rows=1130 width=40) (actual time=0.080..0.087 rows=9 loops=1)
11	Hash Cond: (r.creator_id = c.creator_id)
12	-> Seq Scan on ai_service_reviews r (cost=0.00..21.30 rows=1130 width=8) (actual time=0.026..0.028 r…
13	-> Hash (cost=22.70..22.70 rows=1270 width=36) (actual time=0.031..0.032 rows=4 loops=1)
14	Buckets: 2048 Batches: 1 Memory Usage: 17kB
15	-> Seq Scan on ai_service_creators c (cost=0.00..22.70 rows=1270 width=36) (actual time=0.018..0.…
16	Planning Time: 1.782 ms
17	Execution Time: 1.973 ms

- 설명

- 소요 시간: 3.7 ms
- ROW_NUMBER 순위 부여를 통해 추천 우선순위를 생성
- AI 확장성 o

- AI 응용 예시

구성요소	설명
creator_stats	평균 평점 + 리뷰수로 인기 기획자 후보군 필터링
ROW_NUMBER()	상위 N명의 기획자 순위화하여 추천 순서 정렬
AI 연계	좋은 평가 순으로 상위 N명을 추려 벡터 유사도 필터에 결합해서, 추천 우선순위를 정해주는 전처리용 순위 테이블로 사용

쿼리 2 (서브쿼리 + ORDER BY)

```
EXPLAIN ANALYZE
SELECT *
FROM (
  SELECT
    c.creator_id,
    c.creator_name,
    AVG(r.rating) AS avg_rating,
    COUNT(*) AS review_count
  FROM ai_service_creators c
  JOIN ai_service_reviews r ON c.creator_id = r.creator_id
  GROUP BY c.creator_id, c.creator_name
) AS summary
WHERE review_count >= 2
ORDER BY avg_rating DESC;
```

Data Output Messages Notifications

SQL

	QUERY PLAN text
1	Sort (cost=104.41..105.35 rows=377 width=76) (actual time=0.553..0.554 rows=4 loops=1)
2	Sort Key: (avg(r.rating)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=71.33..88.28 rows=377 width=76) (actual time=0.319..0.329 rows=4 loops=1)
5	Group Key: c.creator_id
6	Filter: (count(*) >= 2)
7	Batches: 1 Memory Usage: 73kB
8	-> Hash Join (cost=38.58..62.85 rows=1130 width=40) (actual time=0.304..0.309 rows=9 loops=1)
9	Hash Cond: (r.creator_id = c.creator_id)
10	-> Seq Scan on ai_service_reviews r (cost=0.00..21.30 rows=1130 width=8) (actual time=0.011..0.012 rows=9 ...
11	-> Hash (cost=22.70..22.70 rows=1270 width=36) (actual time=0.278..0.279 rows=4 loops=1)
12	Buckets: 2048 Batches: 1 Memory Usage: 17kB
13	-> Seq Scan on ai_service_creators c (cost=0.00..22.70 rows=1270 width=36) (actual time=0.012..0.013 ro...
14	Planning Time: 0.660 ms
15	Execution Time: 0.890 ms

- 설명
  - 소요 시간: 1.48 ms
  - 빠르지만 순위 컬럼이 없음

- AI 응용 예시

구성요소	설명
서브쿼리	집계 후 리뷰수 ≥ 2 필터링, 평점순 정렬
ORDER BY	순위 부여 없이 정렬만 수행
AI 연계	유사도 추천 이전에 단순 평점 정렬 필터로 사용 가능

쿼리 3 (RANK())

```
EXPLAIN ANALYZE
WITH creator_stats AS (
  SELECT
    c.creator_id,
    c.creator_name,
    AVG(r.rating) AS avg_rating,
    COUNT(*) AS review_count
  FROM ai_service_creators c
  JOIN ai_service_reviews r ON c.creator_id = r.creator_id
  GROUP BY c.creator_id, c.creator_name
  HAVING COUNT(*) >= 2
)
SELECT *,
       RANK() OVER (ORDER BY avg_rating DESC) AS rank
FROM creator_stats;
```

Data Output Messages Notifications

	QUERY PLAN text
1	WindowAgg (cost=104.41..111.01 rows=377 width=84) (actual time=0.433..0.448 rows=4 loops=1)
2	-> Sort (cost=104.41..105.35 rows=377 width=76) (actual time=0.423..0.425 rows=4 loops=1)
3	Sort Key: creator_stats.avg_rating DESC
4	Sort Method: quicksort Memory: 25kB
5	-> Subquery Scan on creator_stats (cost=71.33..88.28 rows=377 width=76) (actual time=0.103..0.116 rows=4 loo…
6	-> HashAggregate (cost=71.33..88.28 rows=377 width=76) (actual time=0.102..0.114 rows=4 loops=1)
7	Group Key: c.creator_id
8	Filter: (count(*) >= 2)
9	Batches: 1 Memory Usage: 73kB
10	-> Hash Join (cost=38.58..62.85 rows=1130 width=40) (actual time=0.083..0.089 rows=9 loops=1)
11	Hash Cond: (r.creator_id = c.creator_id)
12	-> Seq Scan on ai_service_reviews r (cost=0.00..21.30 rows=1130 width=8) (actual time=0.029..0.030 r…
13	-> Hash (cost=22.70..22.70 rows=1270 width=36) (actual time=0.036..0.036 rows=4 loops=1)
14	Buckets: 2048 Batches: 1 Memory Usage: 17kB
15	-> Seq Scan on ai_service_creators c (cost=0.00..22.70 rows=1270 width=36) (actual time=0.013..0.…
16	Planning Time: 3.654 ms
17	Execution Time: 1.109 ms

- 설명
  - 소요 시간: 1.35 ms
  - RANK는 동점 처리 가능
  - AI 확장성 o
- AI 응용 예시

구성요소	설명
creator_stats	리뷰 수 + 평균 평점 기준으로 필터링된 기획자 집계
RANK	평점 기준 동점순위 허용 → 보다 유연한 랭킹구조 제공
AI 연계	동점 순위를 허용해 같은 우선순위의 여러 추천 후보를 제공 가능 → 유사도 추천 결과와 합쳐서 유연하게 순위 적용 가능

쿼리 4 (FILTER())

```
EXPLAIN ANALYZE
SELECT
  c.creator_id,
  c.creator_name,
  AVG(r.rating) FILTER (WHERE r.review_id IS NOT NULL) AS avg_rating,
  COUNT(r.review_id) AS review_count
FROM ai_service_creators c
LEFT JOIN ai_service_reviews r ON c.creator_id = r.creator_id
GROUP BY c.creator_id, c.creator_name
HAVING COUNT(r.review_id) >= 2
ORDER BY avg_rating DESC;
```

Data Output Messages Notifications

	QUERY PLAN text
1	Sort (cost=109.88..110.94 rows=423 width=76) (actual time=0.203..0.205 rows=4 loops=1)
2	Sort Key: (avg(r.rating) FILTER (WHERE (r.review_id IS NOT NULL))) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=72.38..91.43 rows=423 width=76) (actual time=0.129..0.148 rows=4 loops=1)
5	Group Key: c.creator_id
6	Filter: (count(r.review_id) >= 2)
7	Batches: 1 Memory Usage: 73kB
8	-> Hash Right Join (cost=38.58..62.85 rows=1270 width=44) (actual time=0.105..0.116 rows=9 loops=1)
9	Hash Cond: (r.creator_id = c.creator_id)
10	-> Seq Scan on ai_service_reviews r (cost=0.00..21.30 rows=1130 width=12) (actual time=0.029..0.030 rows=...
11	-> Hash (cost=22.70..22.70 rows=1270 width=36) (actual time=0.048..0.048 rows=4 loops=1)
12	Buckets: 2048 Batches: 1 Memory Usage: 17kB
13	-> Seq Scan on ai_service_creators c (cost=0.00..22.70 rows=1270 width=36) (actual time=0.026..0.027 ro...
14	Planning Time: 1.125 ms
15	Execution Time: 0.648 ms

- 설명
  - 소요 시간: 1.05 ms
  - 가장 빠른 쿼리, 리뷰가 없는 기획자도 분석 가능
  - AI 확장성 o
- AI 응용 예시

구성요소	설명
FILTER()	조건부 집계를 통해 빠르게 평점 평균 계산
LEFT JOIN	리뷰가 없는 기획자까지 포함하여 전체 후보군 생성 가능
AI 연계	실시간 추천이나 전체 기획자 간 유사도 비교를 빠르게 할 수 있다. 또한 리뷰가 없더라도 모든 기획자 정보를 포함해서 추천 후보에 넣을 수 있다.

3. 성능 비교

1. 쿼리1 vs 쿼리2

   쿼리1은 랭킹 컬럼을 제공하므로 상위 N명을 추출하거나 사용자가 현재 몇 위에 있는지를 알려주는 추천 시스템에서 유리하다. 하지만 성능 측면에서는 다소 비용이 든다. 쿼리2는 순위를 부여하는 컬럼이 없기 때문에 추천 알고리즘에서 특정 위치를 식별하거나 상위 몇 명을 구분하는 데는 추가 처리 또는 래퍼 함수가 필요하지만, 성능은 빠르다.

2. 쿼리1 vs 쿼리4

   쿼리 1은 조인된 리뷰 데이터를 기준으로 필터링과 정렬, 순위까지 모두 수행하며 리뷰가 없는 기획자는 전혀 포함되지 않는다. 이에 비해 쿼리 4는 LEFT JOIN을 통해 리뷰가 존재하지 않는 기획자까지 포함하고, FILTER() 구문으로 조건부 집계를 수행한다. 이로 인해 전체 기획자에 대한 벡터 기반 유사도 분석에 활용하기 유리하다. 또한 Postgres 전용 함수를 사용해서 실행 시간과 Planning 시간이 빠른 편이다.

3. 쿼리1 vs 쿼리3

   쿼리1의 ROW_NUMBER()은 단순히 정렬된 순서대로 1, 2, 3... 순위를 부여하는 반면, 쿼리3의 RANK()는 동점 처리 시 동일한 순위를 부여하고 그 다음 순위를 건너뛴다. 예를 들어, 동일한 평점이 2개 있다면 ROW_NUMBER()는 각각 1, 2로 부여하고, RANK()는 둘 다 1로 부여한 뒤 다음은 3이 된다.

   성능 측면에서 RANK()는 ROW_NUMBER()보다 처리량이 적다. ROW_NUMBER()는 모든 행을 고유하게 구분해 정렬해야 하지만, RANK()는 동점 처리를 허용하기 때문에 정렬 이후 중복값을 묶는 처리를 덜 수행하고 실제 성능도 쿼리3이 더 빠르다.

   또한 기능 측면에서 기능적으로도 동일 평점을 받은 기획자를 "동일 순위"로 처리하는 구조이기 때문에 RANK()는 사용자에게 더 유연한 결과를 제공할 수 있다.