

# Этап 10

Для одного из запросов, созданных в пункте 6, провести оптимизацию. В качестве отчета приложить планы выполнения запроса, ваш анализ и показать действия, которые улучшили эффективность запроса.

План запроса.

DBEAVER 22.2.1 - <postgres> Script-2

File Edit Navigate Search SQL Editor Database Window Help

Auto postgres public@kp

<postgres> Script <postgres> Script-1 <postgres> Script-2 <postgres> 09\_Stage\_Queries <postgres> 08\_Stage\_Queries

EXPLAIN ANALYZE  
SELECT  
DISTINCT  
m.id,  
m.title,  
avg(s.number\_of\_stars) OVER (PARTITION BY m.id) AS rate,  
count(s.movie\_id) OVER (PARTITION BY m.id) AS number\_of\_ratings,  
FIRST\_VALUE(up."name" || ' ' || up.surname)  
OVER (PARTITION BY m.id ORDER BY s.rated\_at) AS first\_user  
FROM  
movies m  
JOIN stars s ON  
m.id = s.movie\_id  
JOIN user\_profiles up ON  
up.user\_id = s.user\_id  
ORDER BY rate DESC  
LIMIT 5;

Results 1

EXPLAIN ANALYZE SELECT DISTINCT m.id, m.title, avg(s.number\_of\_stars) OVER (PARTITION BY m.id) AS rate, count(s.movie\_id) OVER (PARTITION BY m.id) AS number\_of\_ratings, FIRST\_VALUE(up."name" || ' ' || up.surname) OVER (PARTITION BY m.id ORDER BY s.rated\_at) AS first\_user FROM movies m JOIN stars s ON m.id = s.movie\_id JOIN user\_profiles up ON up.user\_id = s.user\_id ORDER BY rate DESC LIMIT 5;

Grid

abc QUERY PLAN

1 Limit (cost=91.21..91.23 rows=5 width=96) (actual time=81.604..81.997 rows=5 loops=1)

2 -> Sort (cost=91.21..92.46 rows=497 width=96) (actual time=81.583..81.869 rows=5 loops=1)

3 Sort Key: (avg(s.number\_of\_stars) OVER (?)) DESC

4 Sort Method: top-N heapsort Memory: 26kB

5 -> HashAggregate (cost=77.99..82.96 rows=497 width=96) (actual time=77.326..79.804 rows=141 loops=1)

6 Group Key: avg(s.number\_of\_stars) OVER (?), m.id, m.title, count(s.movie\_id) OVER (?), (first\_value(((up.name)::text || ' ':text) || (up.surname)::text)) OVER (?)

7 Batches: 1 Memory Usage: 81kB

8 -> WindowAgg (cost=50.65..71.78 rows=497 width=96) (actual time=41.973..71.419 rows=497 loops=1)

9 -> WindowAgg (cost=50.65..63.08 rows=497 width=77) (actual time=41.806..59.138 rows=497 loops=1)

10 -> Sort (cost=50.65..51.90 rows=497 width=45) (actual time=41.752..47.182 rows=497 loops=1)

11 Sort Key: m.id, s.rated\_at

12 Sort Method: quicksort Memory: 85kB

13 -> Hash Join (cost=16.75..28.40 rows=497 width=45) (actual time=7.169..35.530 rows=497 loops=1)

14 Hash Cond: (s.user\_id = up.user\_id)

15 -> Hash Join (cost=8.38..18.68 rows=497 width=36) (actual time=3.634..20.247 rows=497 loops=1)

16 Hash Cond: (s.movie\_id = m.id)

17 -> Seq Scan on stars s (cost=0.00..8.97 rows=497 width=20) (actual time=0.033..5.480 rows=497 loops=1)

18 -> Hash (cost=6.50..6.50 rows=150 width=16) (actual time=3.561..3.593 rows=150 loops=1)

19 Buckets: 1024 Batches: 1 Memory Usage: 16kB

20 -> Seq Scan on movies m (cost=0.00..6.50 rows=150 width=16) (actual time=0.242..1.882 rows=150 loops=1)

21 -> Hash (cost=6.50..6.50 rows=150 width=17) (actual time=3.487..3.516 rows=150 loops=1)

22 Buckets: 1024 Batches: 1 Memory Usage: 16kB

23 -> Seq Scan on user\_profiles up (cost=0.00..6.50 rows=150 width=17) (actual time=0.049..1.758 rows=150 loops=1)

24 Planning Time: 0.473 ms

25 Execution Time: 82.178 ms

Record

Save Cancel Script

25 row(s) fetched - 86ms, on 2022-09-26 at 16:04:26

MSK en Writable Smart Insert 10 : 5 : 310 Sel: 0 | 0

Основное время занимает сортировка. Не вижу тут возможностей оптимизации, кроме построения индексов. Кроме таблицы stars, везде используются поля первичного ключа. Соответственно надо построить три индекса на таблицу stars. Поле movie\_id, по которому проходит объединение и отбор окон, поле user\_id, по которому проходит объединение и поле rated\_at, по которому проходит сортировка.

Создаем индексы. Результат не поменялся, и по-прежнему используется Seq Scan.

DBEAVER 22.2.1 - <postgres> Script-2

File Edit Navigate Search SQL Editor Database Window Help

Auto postgres public@kp

<postgres> Script <postgres> Script-1 <postgres> Script-2 <postgres> 09\_Stage\_Queries <postgres> 08\_Stage\_Queries

```

JOIN stars s ON
  m.id = s.movie_id
JOIN user_profiles up ON
  up.user_id = s.user_id
ORDER BY rate DESC
LIMIT 5;

SELECT indexname FROM pg_indexes WHERE tablename = 'stars';

DROP INDEX IF EXISTS stars_movie_id_idx;
DROP INDEX IF EXISTS stars_user_id_idx;
DROP INDEX IF EXISTS starsRated_at_idx;

CREATE INDEX stars_movie_id_idx ON stars (movie_id);
CREATE INDEX stars_user_id_idx ON stars (user_id);

```

Output

```

index "stars_movie_id" does not exist, skipping
index "stars_user_id" does not exist, skipping
index "starsRated_at_idx" does not exist, skipping
index "stars_movie_id" does not exist, skipping
index "stars_user_id" does not exist, skipping
index "starsRated_at_idx" does not exist, skipping
index "starsRated_at_idx" does not exist, skipping

```

Results 1

EXPLAIN ANALYZE SELECT DISTINCT m.id, m.title

Grid

Text

Record

25 row(s) fetched - 84ms, on 2022-09-26 at 17:04:59

MSK en Writable Smart Insert 15:18:404 Sel: 0|0

Попробуем принудительно использовать индексы

DBEAVER 22.2.1 - <postgres> Script-2

File Edit Navigate Search SQL Editor Database Window Help

Auto postgres public@kp

<postgres> Script <postgres> Script-1 \* <postgres> Script-2 <postgres> 09\_Stage\_Queries <postgres> 08\_Stage\_Queries

```

up.user_id = s.user_id
ORDER BY rate DESC
LIMIT 5;

SELECT indexname FROM pg_indexes WHERE tablename = 'stars';

DROP INDEX IF EXISTS stars_movie_id_idx;
DROP INDEX IF EXISTS stars_user_id_idx;
DROP INDEX IF EXISTS starsRated_at_idx;

CREATE INDEX stars_movie_id_idx ON stars (movie_id);
CREATE INDEX stars_user_id_idx ON stars (user_id);
CREATE INDEX starsRated_at_idx ON stars (rated_at);

SET enable_seqscan TO OFF;

```

Output

```

index "stars_movie_id" does not exist, skipping
index "stars_user_id" does not exist, skipping
index "starsRated_at_idx" does not exist, skipping
index "stars_movie_id" does not exist, skipping
index "stars_user_id" does not exist, skipping
index "starsRated_at_idx" does not exist, skipping
index "starsRated_at_idx" does not exist, skipping

```

Results 1

EXPLAIN ANALYZE SELECT DISTINCT m.id, m.title

Grid

Grid	Text	Record
1	Limit (cost=137.61..137.62 rows=5 width=96) (actual time=79.456..80.047 rows=5 loops=1)	
2	-> Sort (cost=137.61..138.85 rows=497 width=96) (actual time=79.435..79.926 rows=5 loops=1)	
3	Sort Key: (avg(s.number_of_stars) OVER (?)) DESC	
4	Sort Method: top-N heapsort Memory: 26kB	
5	-> HashAggregate (cost=124.39..129.36 rows=497 width=96) (actual time=76.114..77.938 rows=141 loops=1)	
6	Group Key: avg(s.number_of_stars) OVER (?), m.id, m.title, count(s.movie_id) OVER (?), (first_value(((up.name)::text    ' ':text)    (up.surname)::text)) OVER (?)	
7	Batches: 1 Memory Usage: 81kB	
8	-> WindowAgg (cost=97.05..118.17 rows=497 width=96) (actual time=41.026..69.829 rows=497 loops=1)	
9	-> WindowAgg (cost=97.05..109.48 rows=497 width=77) (actual time=40.871..58.010 rows=497 loops=1)	
10	-> Sort (cost=97.05..98.29 rows=497 width=45) (actual time=40.819..46.355 rows=497 loops=1)	
11	Sort Key: m.id, s.rated_at	
12	Sort Method: quicksort Memory: 85kB	
13	-> Hash Join (cost=40.69..74.79 rows=497 width=45) (actual time=7.201..35.656 rows=497 loops=1)	
14	Hash Cond: (s.user_id = up.user_id)	
15	-> Hash Join (cost=20.42..53.19 rows=497 width=36) (actual time=3.736..20.747 rows=497 loops=1)	
16	Hash Cond: (s.movie_id = m.id)	
17	-> Index Scan using stars_movie_id_idx on stars s (cost=0.15..31.58 rows=497 width=20) (actual time=0.017..6.197 rows=497 loops=1)	
18	-> Hash (cost=18.39..18.39 rows=150 width=16) (actual time=3.680..3.708 rows=150 loops=1)	
19	Buckets: 1024 Batches: 1 Memory Usage: 16kB	
20	-> Index Scan using movies_pkey on movies m (cost=0.14..18.39 rows=150 width=16) (actual time=0.015..1.981 rows=150 loops=1)	
21	-> Hash (cost=18.39..18.39 rows=150 width=17) (actual time=3.428..3.697 rows=150 loops=1)	
22	Buckets: 1024 Batches: 1 Memory Usage: 16kB	
23	-> Index Scan using user_profiles_pkey on user_profiles up (cost=0.14..18.39 rows=150 width=17) (actual time=0.017..1.743 rows=150 loops=1)	
24	Planning Time: 0.502 ms	
25	Execution Time: 80.397 ms	

Save Cancel Script

25 row(s) fetched - 85ms (1ms fetch), on 2022-09-26 at 17:08:34

MSK en Writable Smart Insert 15:1:387 Sel: 0|0

Результат не поменялся. Делаем вывод, что на таком количестве тестовых данных (stars - 400, movies и users по 150) оптимизация не требуется.