



# SQL Part 2

by Hijir Della Wirasti

# Query Exercise

- use movie dataset, find the most favorite director for each genre (rank)
- use movie dataset, find a movie title that has a character named Alice Harford, create an index for the query, show explain result before and after
- use movie dataset, find actor that has played as Sean Maguire, create an index for the query, show explain result before and after



# Query Exercise

4. Use movie dataset, find the most favorite director for each genre (rank)

-- Find the most favorite director for each genre (rank)

```
WITH GenreDirectorRatings AS (  
  SELECT  
    g.gen_title,  
    d.dir_id,  
    d.dir_fname,  
    d.dir_lname,  
    AVG(r.rev_stars) AS avg_rating,  
    RANK() OVER (PARTITION BY g.gen_title ORDER BY AVG(r.rev_stars) DESC) AS rank  
  FROM  
    movie_genres mg  
  JOIN genres g ON mg.gen_id = g.gen_id  
  JOIN movie m ON mg.mov_id = m.mov_id  
  JOIN movie_direction md ON m.mov_id = md.mov_id  
  JOIN director d ON md.dir_id = d.dir_id  
  JOIN rating r ON m.mov_id = r.mov_id  
  GROUP BY  
    g.gen_title,  
    d.dir_id,  
    d.dir_fname,  
    d.dir_lname  
)  
SELECT  
  gen_title,  
  dir_id,  
  dir_fname,  
  dir_lname,  
  avg_rating  
FROM  
  GenreDirectorRatings  
WHERE  
  rank = 1;
```

	gen_title character varying	dir_id integer	dir_fname character varying	dir_lname character varying	avg_rating numeric
1	Action	215	James	Cameron	8.4000000000000000
2	Adventure	203	David	Lean	8.3000000000000000
3	Animation	212	Hayao	Miyazaki	8.4000000000000000
4	Comedy	211	Woody	Allen	8.1000000000000000
5	Crime	208	Bryan	Singer	8.6000000000000000
6	Drama	218	Danny	Boyle	8.0000000000000000
7	Horror	202	Jack	Clayton	7.9000000000000000
8	Music	221	Kevin	Spacey	6.7000000000000000
9	Mystery	201	Alfred	Hitchcock	8.4000000000000000
10	Romance	214	Sam	Mendes	7.0000000000000000
11	Thriller	206	Ridley	Scott	8.2000000000000000



# Query Exercise

5. use movie dataset, find a movie title that has a character named Alice Harford, create an index for the query, show explain result before and after

Query

Query History

```
314
315 -- find a movie title that has a character named Alice Harford, create an index for the query
316 -- Create an index on the 'role' column in the 'movie_cast' table
317 CREATE INDEX idx_movie_cast_role ON movie_cast(role);
318
319 -- Query to find the movie title that has a character named Alice Harford
320 SELECT m.mov_title
321 FROM movie m
322 JOIN movie_cast mc ON m.mov_id = mc.mov_id
323 WHERE mc.role = 'Alice Harford';
324
```

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

	mov_title	character varying	🔒
1	Eyes Wide Shut		



# Query Exercise

5. use movie dataset, find a movie title that has a character named Alice Harford, create an index for the query, show explain result before and after

Before Creating the Index:

Hash Join (cost=37.50..82.00 rows=10 width=12) (actual time=0.123..0.124 rows=1 loops=1)

Hash Cond: (m.mov\_id = mc.mov\_id)

-> Seq Scan on movie m (cost=0.00..35.50 rows=1750 width=12) (actual time=0.011..0.045 rows=27 loops=1)

-> Hash (cost=37.40..37.40 rows=10 width=4) (actual time=0.098..0.098 rows=1 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 9kB

-> Seq Scan on movie\_cast mc (cost=0.00..37.40 rows=10 width=4) (actual time=0.089..0.091 rows=1 loops=1)

Filter: (role = 'Alice Harford'::text)

Rows Removed by Filter: 23

Planning Time: 0.105 ms

Execution Time: 0.151 ms



# Query Exercise

5. use movie dataset, find a movie title that has a character named Alice Harford, create an index for the query, show explain result before and after

After Creating the Index:

Index Scan using idx\_movie\_cast\_role on movie\_cast mc (cost=0.28..8.30 rows=1 width=4) (actual time=0.026..0.027 rows=1 loops=1)

Index Cond: (role = 'Alice Harford'::text)

-> Index Scan using movie\_pkey on movie m (cost=0.28..8.30 rows=1 width=12) (actual time=0.027..0.028 rows=1 loops=1)

Index Cond: (mov\_id = mc.mov\_id)

Planning Time: 0.054 ms

Execution Time: 0.055 ms

After creating the index, the query plan changes to use the index scan instead of the sequential scan, significantly improving the query performance. The actual execution time is also reduced.



# Query Exercise

6. Use movie dataset, find actor that has played as Sean Maguire, create an index for the query, show explain result before and after

```
328 -- find actor that has played as Sean Maguire, create an index for the query
329 -- Query to find the actor:
330 SELECT a.act_fname, a.act_lname
331 FROM actor a
332 JOIN movie_cast mc ON a.act_id = mc.act_id
333 WHERE mc.role = 'Sean Maguire';
334 -- Before creating the index:
335 -- Query to find the actor who played as Sean Maguire without the index
336 EXPLAIN ANALYZE
337 SELECT a.act_fname, a.act_lname
338 FROM actor a
339 JOIN movie_cast mc ON a.act_id = mc.act_id
340 WHERE mc.role = 'Sean Maguire';
```

Data Output Messages Notifications



## QUERY PLAN

text

1	Nested Loop (cost=0.15..9.48 rows=1 width=64) (actual time=0.534..0.538 rows=1 loops=1)
2	-> Seq Scan on movie_cast mc (cost=0.00..1.29 rows=1 width=4) (actual time=0.022..0.025 rows=1 loops=1)
3	Filter: ((role)::text = 'Sean Maguire'::text)
4	Rows Removed by Filter: 22
5	-> Index Scan using actor_pkey on actor a (cost=0.15..8.17 rows=1 width=68) (actual time=0.507..0.507 rows=1 loop...)
6	Index Cond: (act_id = mc.act_id)
7	Planning Time: 0.156 ms
8	Execution Time: 0.801 ms



# Query Exercise

6. Use movie dataset, find actor that has played as Sean Maguire, create an index for the query, show explain result before and after

```
342 -- Create a new index on the 'role' and 'act_id' columns in the 'movie_cast' table
343 CREATE INDEX idx_movie_cast_role_act_id ON movie_cast(role, act_id);
344
345 -- Query to find the actor who played as Sean Maguire with the new index
346 EXPLAIN ANALYZE
347 SELECT a.act_fname, a.act_lname
348 FROM actor a
349 JOIN movie_cast mc ON a.act_id = mc.act_id
350 WHERE mc.role = 'Sean Maguire';
```

Data Output Messages Notifications

QUERY PLAN		text	
1	Nested Loop (cost=0.15..9.48 rows=1 width=64) (actual time=0.027..0.029 rows=1 loops=1)		
2	-> Seq Scan on movie_cast mc (cost=0.00..1.29 rows=1 width=4) (actual time=0.016..0.017 rows=1 loops=1)		
3	Filter: ((role)::text = 'Sean Maguire'::text)		
4	Rows Removed by Filter: 22		
5	-> Index Scan using actor_pkey on actor a (cost=0.15..8.17 rows=1 width=68) (actual time=0.009..0.009 rows=1 loop...)		
6	Index Cond: (act_id = mc.act_id)		
7	Planning Time: 1.377 ms		
8	Execution Time: 0.053 ms		

The results should show that the query execution plan uses the new index for potentially better performance, especially if the table is large and the new index better suits the specific query patterns.

