Анализа на перформанси меѓу Граф-База на податоци и Релациона база на податоци

Тијана Атанасовска (196014), Александар Мирчовски (196051),

Агон Османи (191025), Мите Мазгалиев (196090)

1. Вовед

Започната веќе подолги години, дигиталната трансформација се уште носи секојдневни иновации и подобрување на постоечките решенија. Популарноста на применета Вештачка интелигенција побарува големи количества податоци, како и нивен диверзитет, формат на чување и креативноста за оптимизирана манипулација. Имплицитно, развојот на една област, означува итеративно подобрување и на останатите засегнати области.

Потребата на човекот да складира податоци, а воедно и знаење, датира од пред 2700 години со креирање на првата библиотеката. Форматот се менувал низ годините, а минатиот век се појавуваат првите дигитални библиотеки. Првин, се појавуваат хиерархискиот и мрежниот модел како два главни пристапи. Сепак, револуцијата ја носат Релационите бази на податоци, сепарабилно развиени со јазикот за нивно пребарување – SQL. Подоцна, паралелно со појавата на разни алатки за овој тип на податоци, се појавуваат и т.н. NoSQL – бази наменети за пофлексибилно работење со податоците, притоа запазувајќи го нивниот формат на потекло.

Инженерството е популарно за барање на компромиси според проблемите кои се јавуваат, па така и овде не може да се издвои еден тип на бази на податоци како апсолутно решение. Клучен момент е да се знаат предностите и недостатоците на секој пристап и да се прават компаративни анализи меѓу нив. Еден од највлијателните фактори во изборот е потеклото на податоците и нивната крајна апликација. Оттука доаѓа и мотивацијата за оваа проектна задача.

Нашата цел е компаративна анализа на перформансите меѓу Граф-база на податоци (ГБП) и Релациона база на податоци (РБП). Експериментот се изведува врз исто почетно податочно множество и на иста машина со цел поголема валидност на резултатите. Податочното множество е The Movies Dataset, јавно достапно на Kaggle.com. Користените технологиите се избрани според честотата на користење во индустријата, поради нивните докажани перформанси. За ГБП се користи Neo4j, додека за РБП - PostgreSQL.

Прашањата кои се поставуваат до базите се на три нивоа – едноставно условно пребарување, прашања со користење на JOIN, прашања со агрегации. Истите се детално презентирани во следните секции.

2. Податочно множество

Податочното множество е [The Movies Dataset], јавно достапно на Kaggle.com. Се содржи од седум .csv фајлови. Првиот фајл е movies_metadata, каде има основни информации за филмовите: име, жанр, буџет, краток опис, оригинален јазик на кој е изведен, достапност на останати јазици,...

Следниот фајл е credits, во кој се зачувани податоци за актерите во филмот и ролјите кои ги глумат. Постојат податоци и за тимот кој работел на филмот и одделите.

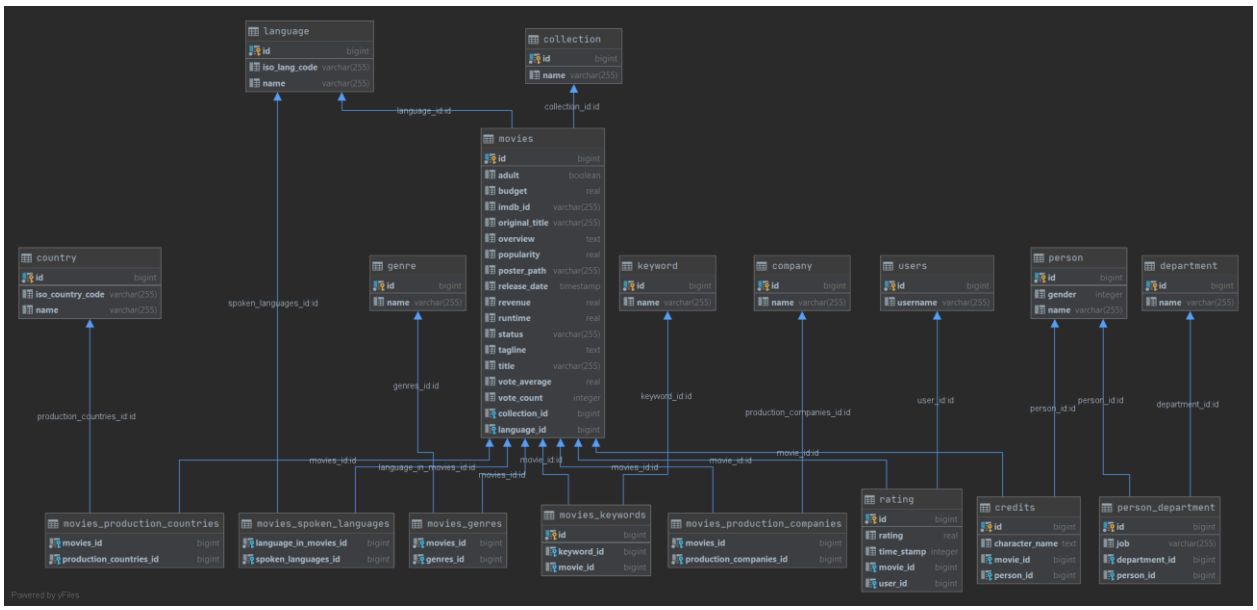
Во фајлот keywords се извадени клучни зборови за секој филм, додека во ratings_small се оценките од корисниците за филмовите.

За дел од фајловите е користена нивната помала верзија (x_small).

3. Модел на податоци

Првиот чекор во креирање база на податоци е дизајнот на логички модел. Односно, визуелно претставување на ентитетите кои ги моделираме и поврзаноста меѓу нив. Процесов го олеснува секој понатамошен чекор, дури и после пуштање во употреба на базата служи за наоѓање на грешки (debug) и лесно претставување на нови членови на тимот кои треба да работат со неа.

Во РБП се користи добро-познатиот ЕР-дијаграм (ентитет релација дијаграм). Овде има цврсто утврдени правила кои треба да се следат за креација. Ентитетите и нивните својства се прикажани во правоаголници, додека меѓусебните релации со

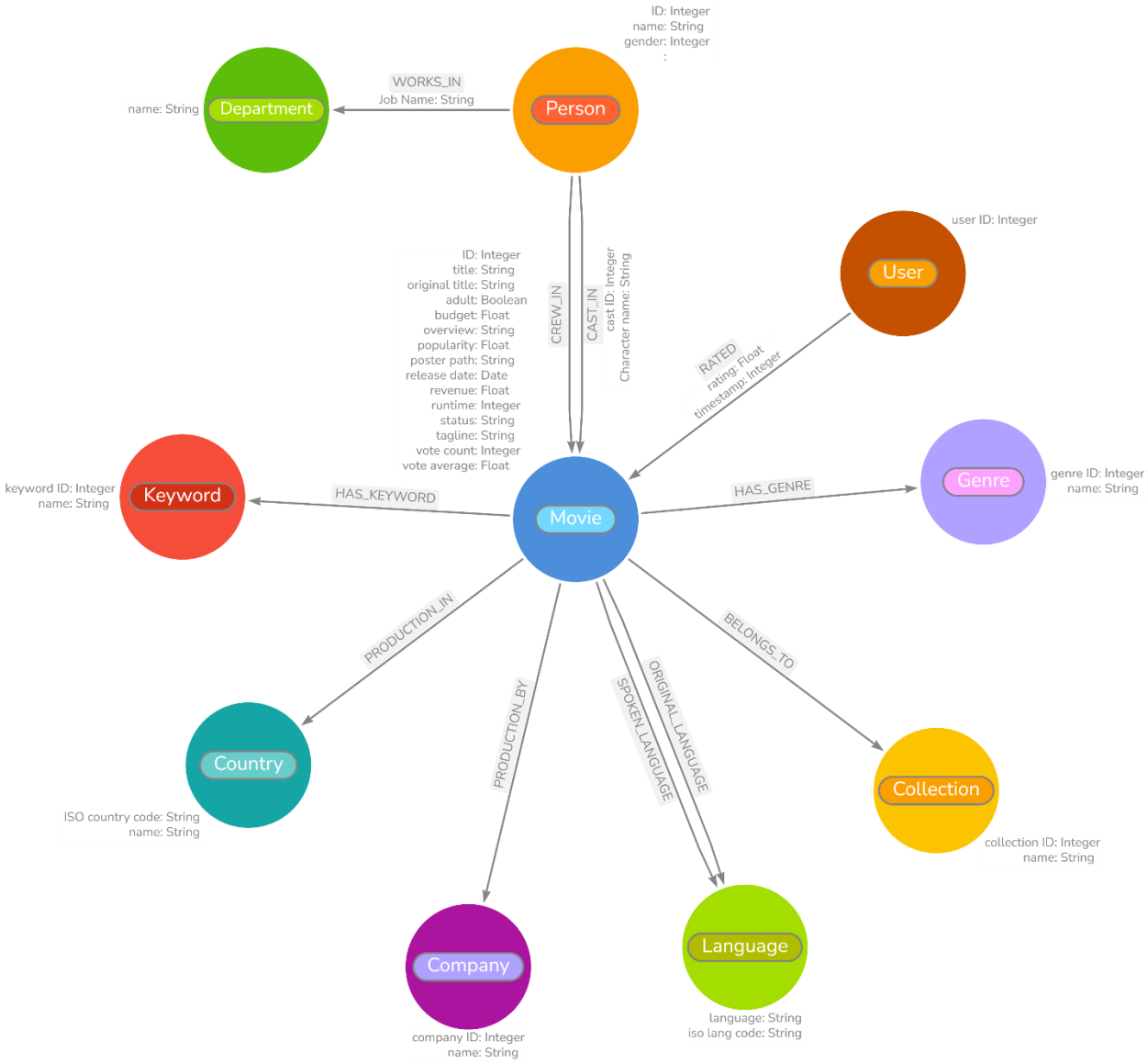


Слика 1

насочена линија каде е прикажана кардиналноста. Дијаграмот е креиран автоматски со помошната алатка Java ORM во софтверот IntelliJ.

Граф базите користат логички модел употребувајќи концепти од теорија на графови.

Типот на ентитетите е претставен како јазел со својства, додека релациите меѓу нив со ребра, кои може да бидат еднонасочени или двонасочни. Врските исто така може да имаат својства. Истите се претставени како составен дел од врската, односно не се изделуваат во посебни јазли, како што во ЕР-моделот креираме нови табели за M-N релации.



Слика 2

Граф дата-моделот е креиран со алатката arrows.app развиена од Neo4j

4. Вчитување на податоците во двете бази

За манипулација на податоците од базите преку апликативен слој, се користат ORM (Object Relational Mapper) и OGM (Object Graph Mapper). Ние ги избравме Hibernate во програмскиот јазик Java и Neomodel во програмскиот јазик Python, за релационата и граф базата соодветно. Генералниот пристап и во двете алатки е креирање на класа за типот на ентитетите и опишување на својствата на ентитетот како својства на класата. Дополнителни ограничување за тип на својство, уникатност, примарен клуч итн., се додаваат како декоратори на класата. Кодот е достапен во двете скрипти: --.py, ___.java

Полнењето на податоци од .csv фајловите во базите може да се изведе преку ОРМ и ОГМ или директно вчитување во базата – метод на големо вчитување (bulk loading). За релационата база се користи првиот пристап, додека во граф-базата вториот пристап со цел покривање на разни методи за остварување на истата цел. Очекувано, методот на големо вчитување е далеку подобар на перформанси, но во понатамошното секојдневно функционирање на апликациите ОРМ и ОГМ нудат многу корисни сервисни операции. За да се овозможи понатамошна надоградба и одржување на базите, креиравме скрипти service.py and service.java во кои се дефинирани функции за постигнување на разни цели во базите.

На слики 3 и 4 е даден дел од кодот за внесување на податоците преку ОРМ и ОГМ. На слика 5 е даден примерот за bulk loading.

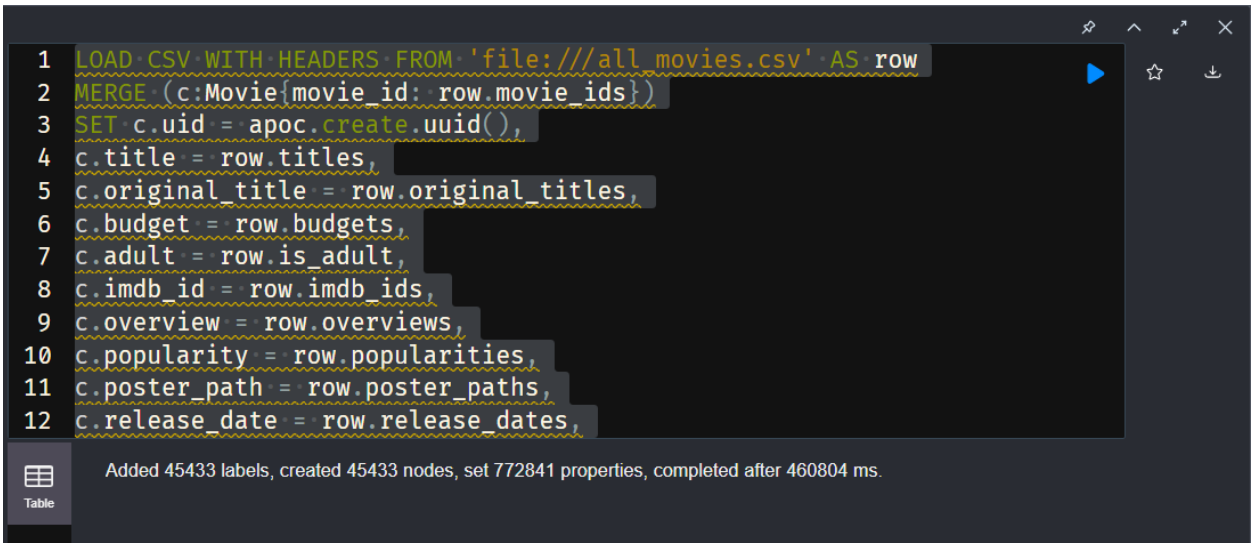
```
140             }
141
142
143             // Create a new YourEntity object and set its attributes based on the CSV values
144             movie.setAdult(parseBoolean(values[0]));
145             movie.setBudget(floatParse(values[2]));
146             movie.setId(Long.parseLong(values[5]));
147             movie.setImdb_ID(values[6]);
148             movie.setOriginal_title(values[8]);
149             movie.setOverview(values[9]);
150             movie.setPopularity(floatParse(values[10]));
151             movie.setPoster_path(values[11]);
152             //1995-10-30
153
154
155             movie.setRelease_date(dateParse(values[14]));
156             movie.setRevenue(floatParse(values[15]));
157             movie.setRuntime(floatParse(values[16]));
158             movie.setStatus(values[18]);
159             movie.setTagline(values[19]);
160             movie.setTitle(values[20]);
161             movie.setVote_average(floatParse(values[22]));
```

Слика 3

```python
new_df = pd.DataFrame()
new_df['titles'] = df_data['title']
new_df['original_titles'] = df_data['original_title']
new_df['budgets'] = df_data['budget']
new_df['is_adult'] = df_data['adult']
new_df['movie_ids'] = df_data['id']
new_df['imdb_ids'] = df_data['imdb_id']
new_df['overviews'] = df_data['overview']
new_df['popularities'] = df_data['popularity']
new_df['poster_paths'] = df_data['poster_path']
new_df['release_dates'] = df_data['release_date']
new_df['revenues'] = df_data['revenue']
new_df['runtimes'] = df_data['runtime']
new_df['status'] = df_data['status']
new_df['taglines'] = df_data['tagline']
new_df['vote_counts'] = df_data['vote_count']
new_df['vote_avgs'] = df_data['vote_average']
new_df.to_csv("all_movies.csv")

for i in range(list(check_lens)[0]):
    exists = Movie.nodes.get_or_none(movie_id=movie_ids[i], imdb_id=imdb_ids[i])
    if exists is None:
        m = Movie(title=titles[i], original_title=original_titles[i], budget=budgets[i],
                  adult=is_adult[i], movie_id=movie_ids[i], imdb_id=imdb_ids[i],
                  overview=overviews[i], popularity=popularities[i], poster_path=poster_paths[i],
                  release_date=release_dates[i], revenue=revenues[i], runtime=runtimes[i],
                  status=status[i], tagline=taglines[i], vote_count=vote_counts[i], vote_average=vote_avgs[i])
        m.save()
```

Слика 4



Слика 5

5. Креирање прашања за компарација меѓу базите

Обични филтрирања:

a) Врати ги насловите на сите филмови

SQL:
```
SELECT m.title FROM movies as m;
```

CYPHER:
```
MATCH (m:Movie) RETURN m.title
```

b) Филтрирај ги филмовите кои го содржат зборот ‘Toy’

SQL:
```
SELECT m
FROM movies as m
WHERE m.title LIKE '%Toy%';
```

CYPHER:
```
MATCH (n:Movie)
WHERE n.title contains "Toy"
RETURN n
```

c) Филтрирај ги филмовите чиј број на зборови во полето overview е поголем од 35 и имаат буџет поголем од 300000

SQL:
```
SELECT m
FROM movies as m
WHERE ARRAY_LENGTH(string_to_array(m.overview, ' '), 1)>35 AND
m.budget > 300000.0;
```

CYPHER:
```
MATCH (n:Movie)
WHERE  SIZE(split(n.overview, " "))>35 AND n.budget>300000.0
RETURN n
```

АГРЕГАЦИИ:

d) SQL:

```
SELECT C.iso_country_code, C.name, COUNT(*) as total_movies
FROM country C INNER JOIN movies_production_countries M ON
                                        C.id=M.production_countries_id
GROUP BY C.id
ORDER BY COUNT(*) DESC
```

CYPHER:

```
MATCH (m:Movie)-[r:PRODUCTION_IN]->(c:Country)
RETURN c.iso_country_code, c.name, count(r) as total_movies
ORDER BY total_movies DESC
```

e) За секоја држава да се направи статистичка анализа (мин, макс, перцентили, стандардна девијација) за траењето на филмовите произведени во неа.

SQL:

```
SELECT c.iso_country_code,
MAX(m.runtime), MIN(m.runtime), STDDEV(m.runtime),
    PERCENTILE_CONT(0) within group (order by m.runtime),
    PERCENTILE_CONT(0.25) within group (order by m.runtime),
    PERCENTILE_CONT(0.5) within group (order by m.runtime),
    PERCENTILE_CONT(0.75) within group (order by m.runtime),
    PERCENTILE_CONT(1) within group (order by m.runtime)
FROM movies_production_countries AS mpc JOIN movies as m ON
mpc.movies_id=m.id

JOIN country as c ON mpc.production_countries_id=c.id
GROUP BY c.iso_country_code
```

CYPHER:

```
MATCH (p:Movie)-[:PRODUCTION_IN]->(c:Country)
RETURN c.iso_country_code, max(p.runtime), min(p.runtime), stDev(p.runtime),
        percentileDisc(p.runtime, 0), percentileDisc(p.runtime, 0.25),
        percentileDisc(p.runtime, 0.5), percentileDisc(p.runtime, 0.75),
        percentileDisc(p.runtime, 1)
```

f) Филтрирај ги сите филмови со број на клучни зборови поголем од просекот на клучни зборови по филм

SQL:

```
SELECT mk.movie_id, COUNT(*)
FROM movies_keywords as mk JOIN movies as m ON mk.movie_id=m.id
GROUP BY mk.movie_id
HAVING COUNT(*) > (SELECT AVG(keywords_per_movie.counted)
                   FROM (SELECT COUNT(*) as counted
                         FROM movies_keywords as mk2 JOIN movies
                         as m2 ON mk2.movie_id=m2.id
                   GROUP BY mk2.movie_id ) as keywords_per_movie)
```

CYPHER:

```
PROFILE
CYPHER runtime=pipelined
MATCH (m1:Movie)
WITH AVG(size((m1)-[:HAS_KEYWORD]->())) AS average_relationsm
MATCH (m:Movie)-[r:HAS_KEYWORD]->(k:Keyword)
WITH average_relationsm,m, COUNT(size((m)-[:HAS_KEYWORD]->())) AS
total_keywords
WHERE total_keywords>average_relationsm
return m.movie_id, total_keywords
```

УПОТРЕБА НА JOIN:

g) Најди ги сите сите филмови кои имаат Keyword ‘rabbit’ и врати ги нивните наслови и популарност.

SQL:

```
SELECT movie.title, movie.popularity
FROM movies movie INNER JOIN (
        SELECT movie_keyword.id, movie_keyword.keyword_id,
        movie_keyword.movie_id
        FROM movies_keywords movie_keyword INNER JOIN (
                SELECT * from keyword k
                WHERE k.name like '%rabbit%')
            k ON movie_keyword.keyword_id=k.id) tmp

        ON movie.id=tmp.movie_id;
```

CYPHER:

```
MATCH (movie:Movie)-[:HAS_KEYWORD]->(keyword:Keyword)
WHERE keyword.name contains "rabbit"
RETURN movie.title, movie.popularity
```

h) Просечна оцена на филмовите дадена од корисниците, групирани по жанр и корисник.

SQL:

```
SELECT r.user_id, g.name, AVG(r.rating)
FROM movies as m JOIN rating as r ON m.id=r.movie_id
JOIN movies_genres as mg ON m.id=mg.movies_id
JOIN genre as g ON mg.genres_id = g.id
GROUP BY r.user_id, g.name
```

CYPHER:

```
MATCH (u:User)-[r:RATED]->(m:Movie)-[:HAS_GENRE]->(g:Genre)
with u,g,AVG(r.rating) as avg_rating
RETURN u.user_id, g.name, avg_rating
```

i) Најди ги сите актери кои глумат во филмови каде бројот на актери е поголем од 10 и при тоа државата во која е произведен филмот да има произведено повеќе од 30 филмови.

SQL:

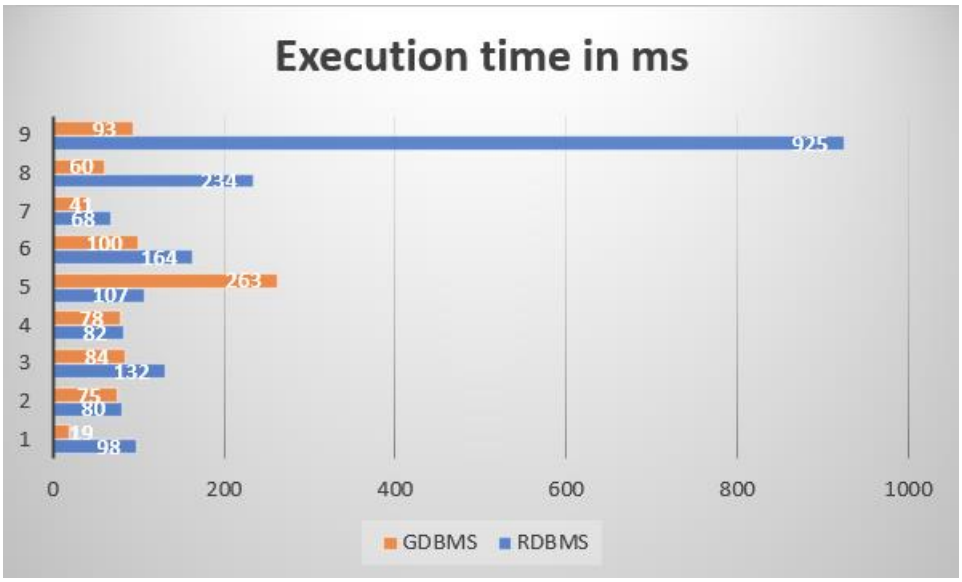
```
SELECT person.name
FROM (SELECT pd.person_id
        FROM person_department as pd JOIN department as d
                                        ON pd.department_id=d.id
        WHERE d.name='Actors' ) as actors
    JOIN credits ON actors.person_id=credits.person_id
    JOIN (SELECT DISTINCT(c.movie_id)
        FROM movies as m JOIN credits as c ON m.id=c.movie_id
        GROUP BY c.movie_id
        HAVING COUNT(c.person_id) > 10) as movies_more_than10_actors
        ON credits.movie_id=movies_more_than10_actors.movie_id
    JOIN movies_production_countries as mpc
        ON mpc.movies_id=movies_more_than10_actors.movie_id
    JOIN (SELECT mpc.production_countries_id
        FROM movies_production_countries as mpc
        GROUP BY mpc.production_countries_id
        HAVING COUNT(mpc.production_countries_id)>30) as
                countries_more_than30_movies
```

```
ON mpc.production_countries_id =
countries_more_than30_movies.production_countries_id
JOIN person ON actors.person_id=person.id

CYPHER:
PROFILE
CYPHER runtime=pipelined
MATCH (c:Country)<-[:PRODUCTION_IN]-(m:Movie)
WITH c, count(m) AS movieNum
WHERE movieNum > 30
MATCH (p:Person)-[:CAST_IN]->(m:Movie)
WITH m, count(p) as total_actors
WHERE total_actors>10
MATCH (m)<-[:CAST_IN]-(p:Person)
RETURN distinct(p.name)
```

6. Резултати и Заклучок

Прашањата се извршени на иста машина, со цел споредување на перформансите да биде точно. На дијаграмот е прикажано времето на извршување во милисекунди. Прашањата се нумерирани од 1 до 9, според редоследот на истите во овој документ.



Дијаграм 1

Во едноставните прашања перформансите се конкуренти, иако скоро секогаш е побрзо извршувањето во Граф базата. Единствен исклучок е прашање 5, за пресметување на статистичка анализа групирано според филмовите. Овде Релационата база постигнува видливо подобри резултати.

Сепак, во покомплексните прашања каде е потребно користење на релации, што означува спојување на табели во Релационата база, перформансите на Граф базата се неколку пати подобри.

Во нашето тестирање, граф базата постигнува видливо подобри резултати. Би било грешка доколку ова се земе како генерален заклучок. Перформансите на базите зависат од системот кој се користи, податоците кои ги имаме, целите кои сакаме да ги остваримe, поставување на прашањата од програмерот, итн... Доколку условите дозволуваат, најдобро е пред започнување со работа да се направи мала анализа на дел од податоците за да се направи споредба.

Користена литература:

https://www.lucidchart.com/pages/er-diagrams#:~:text=Make%20an%20ERD-,What%20is%20an%20ER%20diagram%3F,each%20other%20within%20a%20system.

https://www.thinkautomation.com/histories/the-history-of-databases#:~:text=computers%20were%20invented-,The%20history%20of%20databases%20dates%20back%20long%20before%20computers%20were,the%20beginning%20of%20computerised%20databases.

https://neo4j.com/docs/
https://www.postgresql.org/docs/