Универзитет „Св. Кирил и Методиј“ - Скопје
Факултет за информатички науки и компјутерско инженерство

ИЗВЕШТАЈ ЗА ПРОЕКТ

Граф-базирани NoSQL бази на податоци

Студенти:
Андреј Јанчевски - 151003
Александар Трајковски - 151083
Ненад Трајковиќ - 153050
Предмет: Неструктурирани бази на податоци и XML 2017/2018
Ментор: проф. д-р Слободан Калајџиски

Скопје, 26 септември 2018 год.

Содржина

1 Вовед

Современите софтверски архитекти често дискутираат околу миграции, ефикасност на прашалници, нивна оптимизација, нови технологии за работа со податоци и сл. Од тие причини се појавувааат неструктурираните бази на податоци со цел да се решат проблемите на релационите бази околу поставување на базата врз дистрибуирана околина.
Во оваа проектна задача ќе биде разгледан еден тип на NoSQL бази на податоци, граф-базираните бази на податоци, конкректно Neo4j имплементацијата, и ќе биде извршена споредба со традиционалните SQL релациони бази на податоци како MySQL, низ призмата на брзина на извршување на прашалници.
Со цел да се зачува непристрасност на мерењата, ќе бидат користени две различни бази на податоци, секоја од нив моделирана и во двете технологии. Дефинирани се прашалници со различна комплексност кои би вратиле резултат од интерес на корисниците на соодветниот систем.

2 Методологија

Според упатството, процесот на изработката на проектната задача се одвиваше во 4 фази:

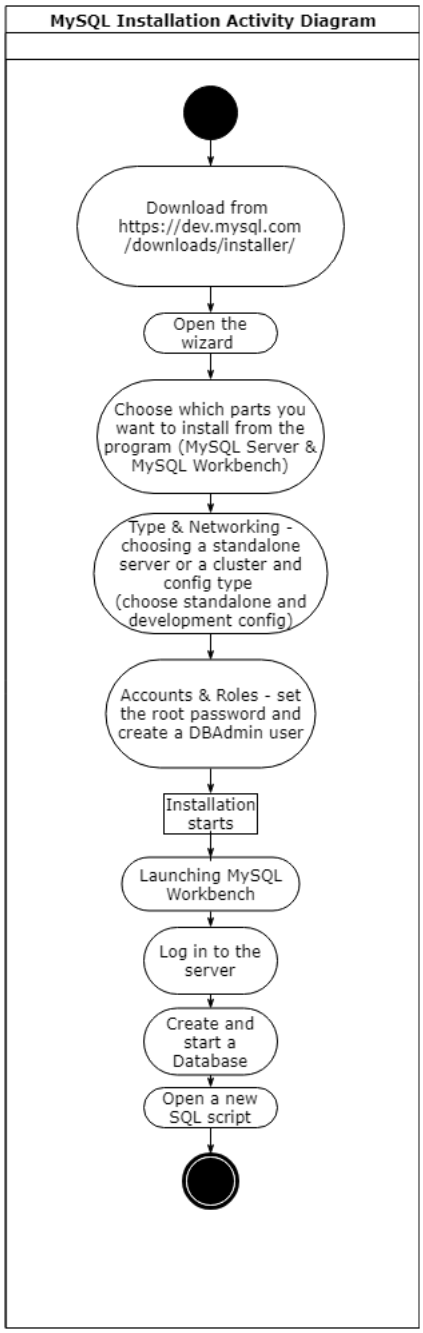
2.1 Инсталација

2.1.1 MySQL v8.0.12

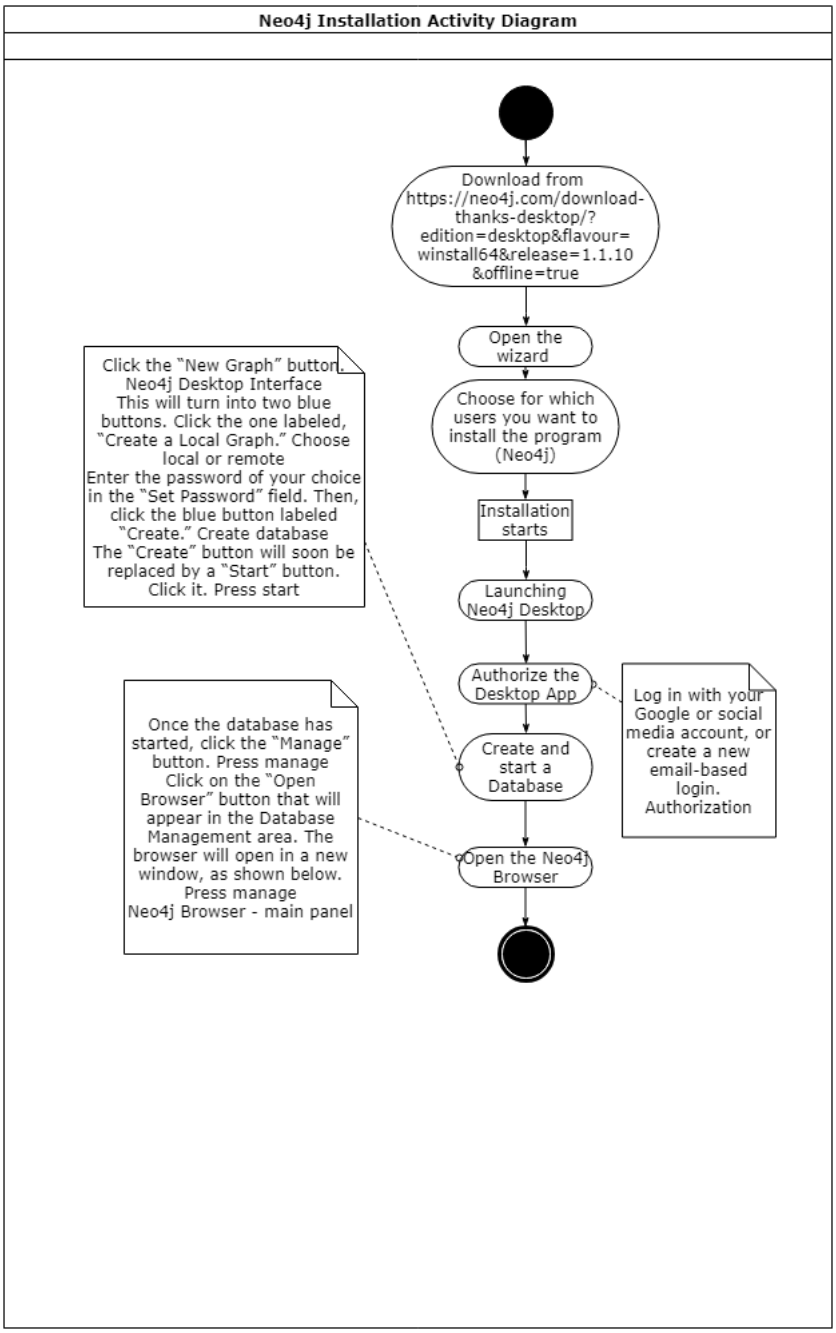
Со цел успешно да се инсталира MySQL системот за управување со бази на податоци потребно е на почеток да се превземе соодветна верзија на MySQL Installer. За нашите потребни беше користена 64-битната верзија за Windows оперативниот систем.
Самиот инсталациски софтвер ги следи официјалните чекори за конфигурирање на серверот и дополнителните апликации, притоа барајќи внес од корисникот каде што е потребно. На слика 1 прикажан е дијаграмот на активности каде што формално е прикажан целиот процес на инсталација и конфигурирање, следејќи ја официјалната документација[1].

2.1.2 Neo4j v3.4.1

Neo4j во целост е граф-базирана база на податоци. Архитектурата е дизајнирана за брзо менаџирање, складирање и изминување на огромен број на јазли и ребра. Кај Neo4j, ребрата се од најголем приоритет и претставуваат материјализирани врски меѓу ентитети. Перформансите на операцијата JOIN кај релационите бази на податоци деградираат експоненцијално со бројот на врски, а Neo4j ја извршува оваа операција во линеарно време бидејќи е претставена како навигација од еден до друг јазел.
Овој поинаков пристап кон чување и прегледување конекции нуди перформанси до дури 4 милиони скокови во секунда. Имајќи предвид дека повеќето изминувања на граф се локални во соседството на некој јазел, вкупната количина запишани податоци нема ефект врз времето на извршување на операциите.
Neo4j поддржува ACID трансакции, со што се гарантира конзистентност на податоците во случај на системски пад или проблеми со хардверот.
Кластерирање со Neo4j е дизајнирано да подджува бизнис-критични и високоперформансни апликации. Може да се зачуваат стотици трилиони ентитети и притоа сепак запазувајќи ја компактноста на складот. Neo4j може да се имплементира како скалабилен кластер и доволни се околу десетина машини за да се извршува, наместо стотици или илјади, со што се намалуваат трошоците.[2]
На слика 2 прикажан е дијаграмот на активности каде што формално е прикажан целиот процес на инсталација и конфигурирање, следејќи ја официјалната документација[3]. При текот на изработката на проектот, беше користена Desktop верзијата на Neo4j.



Слика 1: Дијаграм на активности за инсталација на MySQL сервер



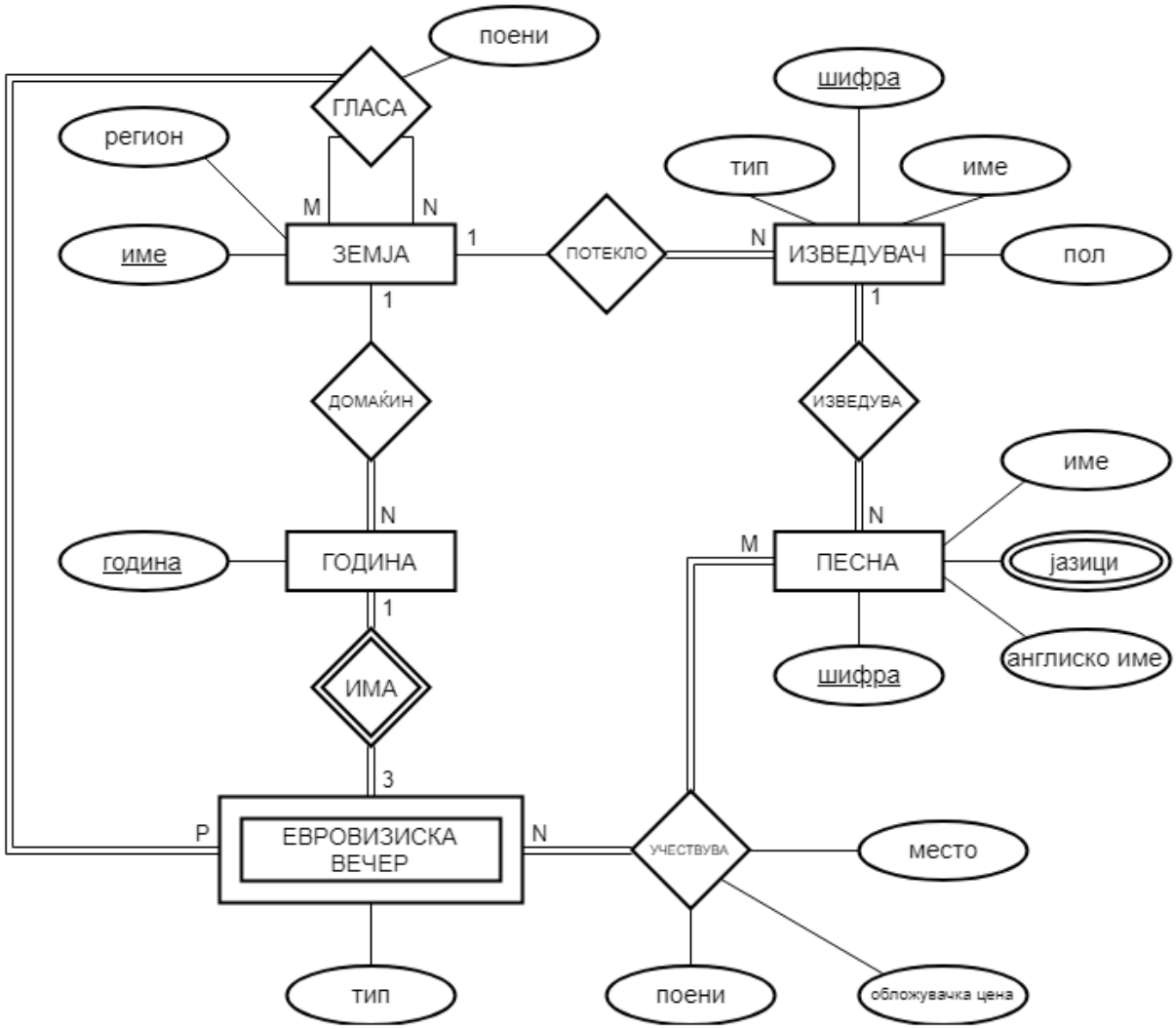
Слика 2: Дијаграм на активности за инсталација на Neo4j

2.2 Импортирање на податоците

Тимот работеше на две бази на податоци, Eurovision и IMDB. Дадените податоци беа анализирани, беа направени модели со цел подобро разбирање на податоците и меѓусебните врски.

2.2.1 Евровизија

По анализирање на структурата на податоците за Евровизија, беше изведен соодветниот ER дијаграм, прикажан на слика 3, од кој потоа беше изведен релационен модел кој ќе се користи за импортирање на податоците во MySQL.



Слика 3: ER дијаграм за Eurovision базата на податоци

Оваа база на податоци има цел да чува информации за одржувања на шоуто Евровизија по години. Секоја година се одржуваат најмногу 3 вечери, две полуфинални вечери и едно финале. На секоја вечер учествуваат неколку земји, претставени преку својот изведувач и неговата песна. По изведувањето на песните секоја земја гласа за 10 други земји, давајќи поени од 1 од 8, 10 или 12 поени.
Пред да се импортираат податоците, врз дадените сирови податоци потребно беше да се изврши претпроцесирање. R скриптата `eurovision_preprocessing.R` ја извршува оваа задача добивајќи ги готовите датотеки за импортирање, секоја од нив претставувајќи по една релација од моделот:

- `Zemja.csv`;
- `Godina.csv`;
- `EvroviziskaVecher.csv`;
- `Izveduvach.csv`;
- `Pesna.csv`;
- `PesnaJazik.csv`;
- `Uchestvo.csv`;
- `Glasanje.csv`;

SQL скриптата `EuroVisionScript.sql` ги соддржи DDL изразите за креирање на шемата и табелите. Со употреба на MySQL Workbench софтверот беше извршена оваа скрипта и потоа со посебна алатка од истиот софтвер беа импортирани податоците во секоја табела.

За Neo4j потребен е посебен модел, кој беше изведен преку трансформација на оригиналниот релационен модел во граф структура. Потребно е да се дефинираат типовите јазли со нивните евентуални атрибути, како и типовите врски меѓу јазлите.
Како типови на јазли беа дефинирани:

- `Zemja`
- `EvroviziskaVecher`
- `Pesna`
- `Godina`
- `Izveduvach`
- `Glasanje`

Покрај јазлите се дефинираат и следните типови врски:

- `:IMA`
- `:POTEKLO`
- `:UCHESTVO`
- `:GLAS_ZA`
- `:DOMAKJIN`
- `:AVTOR`
- `:GLAS_OD`
- `:GLAS_KOGA`

При трансформацијата, секоја двојна врска може да се замени со ребро меѓу соодветните 2 јазли, но тројната врска ГЛАСА потребно е да се замени со јазел за секој глас, поврзан со 3 ребра кон соодветните други јазли.
Neo4j го користи Cypher јазикот за креирање, извлекување и ажурирање на податоци. Cypher е деклатаривен прашален јазик кој е дизајниран така што синтаксата визуелно презентира јазли и врски. Сите потребни скрипти и прашалници беа имплементирани следејќи ја официјалната Cypher документација[4].
Во прилот е даден Cypher кодот за импортирање на евровизиските податоци во Neo4j графот:

```
LOAD CSV WITH HEADERS FROM 'file:///Zemja.csv' AS line
CREATE (:Zemja { ime: line.ime, region: line.region})

CREATE CONSTRAINT ON (z:Zemja) ASSERT z.ime IS UNIQUE

LOAD CSV WITH HEADERS FROM 'file:///Godina.csv' AS line
MATCH (z:Zemja { ime: line.zemjaDomakjin })
CREATE (g:Godina {godina: line.godina })
CREATE (z)-[:DOMAKJIN]->(g)

CREATE CONSTRAINT ON (g:Godina) ASSERT g.godina IS UNIQUE

LOAD CSV WITH HEADERS FROM 'file:///EvroviziskaVecher.csv' AS line
MATCH (g:Godina { godina: line.godina })
CREATE (ev:EvroviziskaVecher {tip: line.tip })
CREATE (g)-[:IMA]->(ev)

USING PERIODIC COMMIT 100
LOAD CSV WITH HEADERS FROM 'file:///Izveduvach.csv' AS line
MATCH (z:Zemja { ime: line.zemjaPoteklo })
CREATE (i:Izveduvach {id: line.id, ime: line.ime,
tip: line.tip, pol: line.pol })
CREATE (z)-[:POTEKLO]->(i)

CREATE CONSTRAINT ON (i:Izveduvach) ASSERT i.id IS UNIQUE

USING PERIODIC COMMIT 100
LOAD CSV WITH HEADERS FROM 'file:///Pesna.csv' AS line
MATCH (i:Izveduvach { id: line.avtor })
CREATE (p:Pesna {id: line.id, ime: line.ime,
```

```
imeAngliski: line.imeAngliski, jazici: [] })
CREATE (i)-[:AVTOR]->(p)

CREATE CONSTRAINT ON (p:Pesna) ASSERT p.id IS UNIQUE

USING PERIODIC COMMIT 100
LOAD CSV WITH HEADERS FROM 'file:///PesnaJazik.csv' AS line
MATCH (p:Pesna { id: line.pesna })
SET p.jazici = p.jazici + line.jazik

USING PERIODIC COMMIT 100
LOAD CSV WITH HEADERS FROM 'file:///Uchestvo.csv' AS line
MATCH (ev:EvroviziskaVecher { tip: line.vecher}) <-[:IMA]-
(g:Godina { godina: line.godina })
MATCH (p:Pesna {id: line.pesna })
CREATE (p)-[:UCHESTVO { mesto: line.mesto, poeni: line.poeni,
cenaOblozhuvanje: line.cenaOblozhuvanje }]->(ev)

USING PERIODIC COMMIT 100
LOAD CSV WITH HEADERS FROM 'file:///Glasanje.csv' AS line
MATCH (ev:EvroviziskaVecher { tip: line.vecher}) <-[:IMA]-
(g:Godina { godina: line.godina })
MATCH (z1:Zemja {ime: line.zemja1 })
MATCH (z2:Zemja {ime: line.zemja2 })
CREATE (glas:Glasanje { poeni: line.poeni })
CREATE (z1)-[:GLAS_OD]->(glas)
CREATE (glas)-[:GLAS_ZA]->(z2)
CREATE (ev)-[:GLAS_KOGA]->(glas)
```

Се користи наредбата `LOAD CSV` за читање на записи од датотека, но притоа применувајќи ја и опцијата `USING PERIODIC COMMIT` со што трансакцијата се извршува по делови, а не одеднаш со цел да се избегне преплавување на серверот со огромен број информации и оддржување на скалабилност.

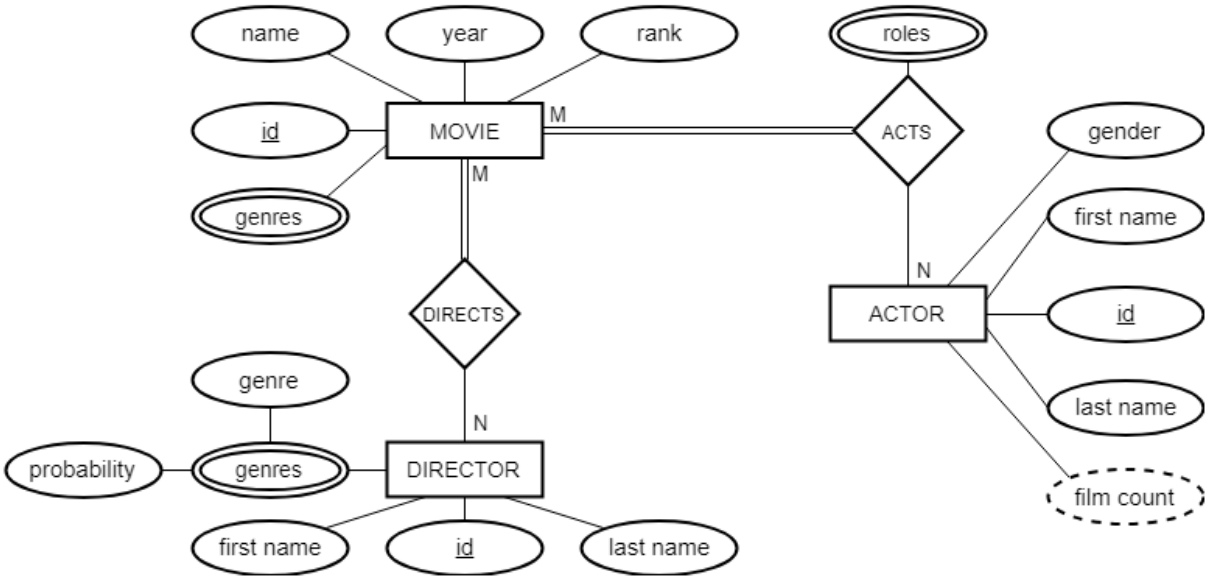
2.2.2 IMDB

Извршувајќи ја дадената скрипта `imdb.sql` во MySQL Workbench, се креира IMDB шемата и се полни со сите податоци, со што веднаш може да се користат податоците. Сепак, бидејќи потребно е и да креира и посебен модел за Neo4j беше анализирана структурата на податоците и беше изведен соодветниот ER дијаграм, прикажан на слика 4. Базата чува податоци за огромен број на филмови, режисери и актери. Се чуваат и сите улоги кои актерот ги имал во секој филм, кон кои жанрови припаѓа секој филм и кои жанрови секој режисер ги преферира. Користејќи ја скриптата `ImdbExportScript.sql` беа извлечени податоците од секоја табела во посебна датотека со цел да може да се импортираат и во Neo4j графот, кој се добива по трансформација на релациониот модел. Како типови на јазли беа дефинирани:

- `Movie`
- `Actor`
- `Director`
- `Genre`

Покрај јазлите се дефинираат и следните типови врски:

- `:DIRECTS`
- `:ACTS`
- `:HAS_GENRE`
- `:DIRECTS_GENRE`



Слика 4: ER дијаграм за IMDB базата на податоци

При трансформацијата потребно е да изведе ентитетот ЖАНР, кој би се поврзал со филмовите и режисерите. Во прилот е даден Cypher кодот за импортирање на податоците од IMDB во Neo4j графот:

```
USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///Actors.csv' AS line
CREATE (:Actor { id: line[0], firstName: line[1], lastName: line[2],
gender: line[3], filmCount: line[4] })

CREATE CONSTRAINT ON (a:Actor) ASSERT a.id IS UNIQUE

USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///Directors.csv' AS line
CREATE (:Director { id: line[0], firstName: line[1], lastName: line[2] })

CREATE CONSTRAINT ON (d:Director) ASSERT d.id IS UNIQUE

USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///Movies.csv' AS line
CREATE (:Movie { id: line[0], name: line[1],
year: line[2], rank: line[3] })

CREATE CONSTRAINT ON (m:Movie) ASSERT m.id IS UNIQUE

USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///MoviesDirectors.csv' AS line
MATCH (d:Director { id: line[0] })
MATCH (m:Movie { id: line[1] })
CREATE (d)-[:DIRECTS]->(m)

USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///Roles.csv' AS line
MATCH (a:Actor { id: line[0] })
```

```
MATCH (m:Movie { id: line[1] })
MERGE (a)-[r:ACTS]->(m)
ON CREATE SET r.roles = [ line[2] ]
ON MATCH SET r.roles = r.roles + line[2]

USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///MoviesGenres.csv' AS line
MATCH (m:Movie { id: line[0] })
MERGE (g:Genre { genre: line[1] })
CREATE (m)-[:HAS_GENRE]->(g)

USING PERIODIC COMMIT 100
LOAD CSV FROM 'file:///DirectorsGenres.csv' AS line
MATCH (d:Director { id: line[0] })
MERGE (g:Genre { genre: line[1] })
CREATE (d)-[:DIRECTS_GENRE { probabillity: line[2] }]->(g)
```

2.3 Користење на податоците

Беа избрани неколку сценарија за пристап до податоците и соодветно подредени според нивната комплексност. За секој од прашалниците ќе бидат приложени решенијата во SQL и Cypher.

2.3.1 Eurovision

1. Да се најдат сите песни и нивните изведувачи кои во името на песната го имаат зборот 'love'.
   Решението во SQL е дадено во прилог:

   ```
   SELECT Pesna.imeAngliski, Izveduvach.ime
   FROM Pesna JOIN Izveduvach ON Pesna.avtor = Izveduvach.id
   WHERE Pesna.imeAngliski LIKE "\%Love\%"
   OR Pesna.imeAngliski LIKE "\%love\%"
   ```

   Решението во Cypher е дадено во прилог:

   ```
   MATCH (i:Izveduvach)-[:AVTOR]->(p:Pesna)
   WHERE p.imeAngliski CONTAINS 'love'
   OR p.imeAngliski CONTAINS 'Love'
   RETURN p, i
   ```

2. Да се најде земјата, изведувачот и песната која во финалето во 2004 година освоила 4то место.
   Решението во SQL е дадено во прилог:

   ```
   SELECT I.zemjaPoteklo, I.ime, P.ime
   FROM Uchestvo U
    JOIN Pesna P ON P.id=U.pesna
    JOIN Izveduvach I ON I.id=P.avtor
   WHERE U.godina=2004
   AND U.mesto=4
   AND U.vecher="F"
   ```

Решението во Cypher е дадено во прилог:

```
MATCH (z:Zemja)-[:POTEKLO]->(i:Izveduvach)-[:AVTOR]->(p:Pesna)
-[u:UCHESTVO]->(ev:EvroviziskaVecher)<-[:IMA]-(g:Godina)
WHERE ev.tip='F'
AND g.godina='2004'
AND u.mesto='4'
return z, i, p
```

3. Да се најдат изведувачите кои изведувале повеќе од една песна.
Решението во SQL е дадено во прилог:

```
SELECT I.ime, COUNT(*) AS CNT
FROM Pesna P
  JOIN Izveduvach I ON I.id=P.avtor
GROUP BY I.ime
HAVING CNT >1
ORDER BY CNT DESC
```

Решението во Cypher е дадено во прилог:

```
MATCH (p:Pesna)<-[:AVTOR]-(i:Izveduvach)
WITH i.ime AS IME, COUNT(p.ime) AS CNT
WHERE CNT>1
RETURN IME,CNT
ORDER BY CNT DESC
```

4. Да се најдат земјите кои имале повеќе машки изведувачи од женски низ годините.
Решението во SQL е дадено во прилог:

```
SELECT Z.ime,
    COUNT(CASE WHEN I.pol='Male' THEN 1 END ) AS CNT_MALE,
    COUNT(CASE WHEN I.pol='Female' THEN 1 END ) AS CNT_FEMALE
FROM Zemja Z
  JOIN Izveduvach I ON I.zemjaPoteklo = Z.ime
GROUP BY Z.ime
HAVING CNT_MALE>CNT_FEMALE
```

Решението во Cypher е дадено во прилог:

```
MATCH (z:Zemja)-[:POTEKLO]->(i:Izveduvach)
WITH z,i, REDUCE(s={in: 0, out: 0}, x IN
COLLECT({zemja: z.ime, pol: i.pol}) |
    CASE WHEN x.pol="Male"
      THEN {in: s.in + 1, out: s.out}
      WHEN x.pol="Female"
      THEN {in: s.in, out: s.out + 1}
      ELSE {in: s.in, out: s.out}
    END) AS RES
WITH RES, z, i
WITH z.ime AS ZEMJA,
SUM(RES.in) AS BROJ_MASHKI, SUM(RES.out) AS BROJ_ZHENSKI
WHERE BROJ_MASHKI > BROJ_ZHENSKI
RETURN ZEMJA, BROJ_MASHKI, BROJ_ZHENSKI
ORDER BY ZEMJA
```

5. За секоја земја да се пресмета вкупниот број на поени што ги добила од гласови од земјите во истиот регион, наспрема вкупниот број на поени што ги добила од останатите земји. Кои земји добиле повеќе поени од земјите во нивниот регион?
Решението во SQL е дадено во прилог:

```
CREATE OR REPLACE VIEW RAZLICHEN_REGION AS
SELECT G.zemja2, SUM(G.poeni) AS p
FROM Glasanje G
JOIN Zemja Z1 ON Z1.ime=G.zemja1
JOIN Zemja Z2 ON Z2.ime=G.zemja2
WHERE Z1.region!=Z2.region
GROUP BY G.zemja2;

CREATE OR REPLACE VIEW IST_REGION AS
SELECT G.zemja2, SUM(G.poeni) AS p
FROM Glasanje G
JOIN Zemja Z1 ON Z1.ime=G.zemja1
JOIN Zemja Z2 ON Z2.ime=G.zemja2
WHERE Z1.region=Z2.region
GROUP BY G.zemja2;

SELECT G.zemja2, IR.p, RR.p
FROM Glasanje G
JOIN IST_REGION IR ON IR.zemja2=G.zemja2
JOIN RAZLICHEN_REGION RR ON RR.zemja2=G.zemja2
WHERE IR.p>RR.p
GROUP BY G.zemja2;
```

Решението во Cypher е дадено во прилог:

```
MATCH (z1:Zemja)-[:GLAS_OD]->(g:Glasanje)-[:GLAS_ZA]->(z2:Zemja)
WITH z2,
  REDUCE(s={in: 0, out: 0}, x IN
  COLLECT({reg: z1.region, pts: toInteger(g.poeni)}) |
    CASE WHEN x.reg=z2.region
      THEN {in: s.in + x.pts, out: s.out}
      ELSE {in: s.in, out: s.out + x.pts}
    END) AS RES
WHERE RES.in > RES.out
RETURN z2.ime AS NAME, RES.in AS POINTS
```

6. Да се најдат сите изведувачи кои се група составена само од женски членови, кои не се пласирале никогаш во финале и нивната песна се пеела на повеќе од еден јазик.
Решението во SQL е дадено во прилог:

```
CREATE OR REPLACE VIEW JAZICI_BROJ AS (
SELECT PJ.pesna, COUNT(DISTINCT PJ.jazik) AS CNT
FROM PesnaJazik PJ
GROUP BY PJ.pesna
);

SELECT I.ime
FROM Izveduvach I
```

```
JOIN Pesna P ON P.avtor = I.id
JOIN JAZICI_BROJ JB ON JB.pesna = P.id
AND I.tip='Group' AND I.pol='Female' AND JB.CNT>1
AND NOT EXISTS (
SELECT *
FROM Uchestvo U
WHERE U.pesna=P.id
AND U.vecher='F'
);
```

Решението во Cypher е дадено во прилог:

```
MATCH (i:Izveduvach)-[:AVTOR]->(p:Pesna)
-[u:UCHESTVO]->(ev:EvroviziskaVecher)
WHERE i.tip = 'Group' AND i.pol='Female' AND SIZE(p.jazici)>1
WITH i, p, collect(ev.tip) AS VECHERI
WHERE NOT 'F' IN VECHERI
RETURN i
```

7. Да се најде земјата која вкупно низ текот на годините освоила најмногу поени.
   Решението во SQL е дадено во прилог:

```
SELECT G.zemja2, SUM(G.poeni) AS SUMA
FROM Glasanje G
GROUP BY G.zemja2
ORDER BY SUMA DESC
LIMIT 1
```

   Решението во Cypher е дадено во прилог:

```
MATCH (z1:Zemja)-[:GLAS_OD]->(g:Glasanje)-[:GLAS_ZA]->(z2:Zemja)
WITH z1, z2, g
WITH z2, sum(toInteger(g.poeni)) AS POENI
RETURN z2.ime AS ZEMJA, POENI
ORDER BY POENI DESC
LIMIT 1
```

8. Да се сортираат земјите според вкупно освоени поени низ текот на годините.
   Решението на ова прашање е скоро идентично со претходното, но сепак ова прашање беше дефинирано со цел да се утврди дали прикажување на само првата редица од резултатите има влијание врз перформансите кај двете технологии.

9. За секоја земја да се пресмета вкупниот број поени што ги добила од гласови од земјите во истиот регион, наспрема вкупниот број на поени што ги добила од останатите земји, во рамки на една година. Кои земји во кои години добивале повеќе поени од земјите од истиот регион?
   Решението во SQL е дадено во прилог:

```
CREATE OR REPLACE VIEW IST_REGION_PO_GODINA AS
SELECT G.godina, G.zemja2 AS ZEMJA, SUM(G.poeni) AS SUMA
FROM Glasanje G
JOIN Zemja Z1 ON Z1.ime = G.zemja1
JOIN Zemja Z2 ON Z2.ime = G.zemja2
```

```
WHERE Z1.region = Z2.region
GROUP BY G.godina, G.zemja2;

CREATE OR REPLACE VIEW RAZ_REGION_PO_GODINA AS
SELECT G.godina, G.zemja2 AS ZEMJA, SUM(G.poeni) AS SUMA
FROM Glasanje G
JOIN Zemja Z1 ON Z1.ime = G.zemja1
JOIN Zemja Z2 ON Z2.ime = G.zemja2
WHERE Z1.region != Z2.region
GROUP BY G.godina, G.zemja2;

SELECT DISTINCT(G.zemja2), IR.SUMA, RR.SUMA
FROM Glasanje G
JOIN IST_REGION_PO_GODINA IR ON IR.ZEMJA=G.zemja2
JOIN RAZ_REGION_PO_GODINA RR ON RR.ZEMJA=G.zemja2
WHERE IR.suma>RR.suma
```

Решението во Cypher е дадено во прилог:

```
MATCH (z1:Zemja)-[:GLAS_OD]->(glas:Glasanje)-[:GLAS_ZA]->(z2:Zemja)
MATCH (glas)<-[:GLAS_KOGA]-(ev:EvroviziskaVecher)<-[:IMA]-(g:Godina)
WITH z2, g,
REDUCE( s={in: 0, out: 0}, x IN
COLLECT({reg: z1.region, pts: toInteger(glas.poeni), god: g.godina}) |
    CASE WHEN x.reg=z2.region
      THEN {in: s.in + x.pts, out: s.out}
      ELSE {in: s.in, out: s.out + x.pts}
    END) AS RES
WHERE RES.in > RES.out
RETURN g.godina AS GODINA, z2.ime AS ZEMJA,
RES.in AS pointsFromRegion, RES.out AS pointFromOutRegion
ORDER BY GODINA DESC
```

10. За секоја година од сите земји кои не се пласирале во финале да се најде земјата која била најблиску до влез во однос на поените.
Решението во SQL е дадено во прилог:

```
SELECT U.godina AS GODINA, U.pesna AS PESNA, I.zemjaPoteklo AS ZEMJA
FROM Uchestvo AS U
JOIN Pesna AS P ON P.id = U.pesna
JOIN Izveduvach AS I ON I.id = P.avtor
WHERE U.vecher = "SF1" OR U.vecher = "SF2"
AND U.mesto = 11
AND U.poeni =
    (SELECT MAX(Z.poeni)
    FROM Uchestvo AS U1
    WHERE U1.vecher = "SF1" OR U1.vecher = "SF2"
    AND U1.mesto = 11
    AND U1.godina =  U.godina)
GROUP BY U.godina
ORDER BY U.godina;
```

Решението во Cypher е дадено во прилог:

```
MATCH (z:Zemja)-[:POTEKLO]->(i:Izveduvach)-[:AVTOR]->(p:Pesna)
-[u:UCHESTVO]->(ev:EvroviziskaVecher)<-[:IMA]-(g:Godina)
WHERE ev.tip IN ["SF1","SF2"]
AND toInteger(u.mesto) = 11
RETURN head(collect(z.ime)) AS ZEMJA,
toInteger(g.godina) AS GODINA, MAX(toInteger(u.poeni)) AS POENI
ORDER BY GODINA DESC
```

2.3.2 IMDB

1. Да се најдат сите филмови кои еден од нивните жанрови е хорор, имаат минимум ранк од 7.0 и се излезени после 1980 година.
   Решението во SQL е дадено во прилог:

```
SELECT M.*
FROM IMDB.movies M, IMDB.movies_genres MG
WHERE M.id = MG.movie_id
AND M.rank >= 7.0
AND M.year >= 1980
AND MG.genre = 'Horror'
ORDER BY M.year DESC, M.rank DESC
```

   Решението во Cypher е дадено во прилог:

```
MATCH (m:Movie)-[:HAS_GENRE]->(g:Genre {genre: "Horror"})
WHERE toInteger(m.year) >= 1980
AND toFloat(m.rank)>=7.0
RETURN m
ORDER BY m.year DESC, m.rank DESC
```

2. Да се најдат актерите чие име е John и кои играле барем еднаш во ист филм повеќе улоги.
   Решението во SQL е дадено во прилог:

```
SELECT CONCAT(A.first_name,' ',A.last_name) AS ACTOR,
M.name AS MOVIE, COUNT(DISTINCT R.role) AS NUM_ROLES
FROM IMDB.actors A, IMDB.roles R, IMDB.movies M
WHERE R.actor_id = A.id
AND M.id = R.movie_id
AND A.first_name = 'John'
GROUP BY ACTOR, MOVIE
HAVING NUM_ROLES > 1
ORDER BY NUM_ROLES DESC
```

   Решението во Cypher е дадено во прилог:

```
MATCH (a:Actor { firstName: 'John' })-[acts:ACTS]->(m:Movie)
WITH a, m, size(acts.roles) AS NUM_ROLES
WHERE NUM_ROLES > 1
RETURN a.firstName+' '+a.lastName AS ACTOR,
m.name AS MOVIE, NUM_ROLES
ORDER BY NUM_ROLES DESC
```

3. Да се најде режисерот кој има режисирано најмногу филмови.
   Решението во SQL е дадено во прилог:

```
SELECT CONCAT(D.first_name,' ',D.last_name) AS DIRECTOR,
DIRECTORS_NUM_MOVIES.NUM_MOVIES AS NUM_MOVIES
FROM IMDB.directors D,
(SELECT MD.director_id AS director,
COUNT(DISTINCT MD.movie_id) AS NUM_MOVIES
FROM IMDB.movies_directors MD
GROUP BY MD.director_id
ORDER BY NUM_MOVIES DESC) AS DIRECTORS_NUM_MOVIES
WHERE D.id = DIRECTORS_NUM_MOVIES.director
AND D.first_name != 'Unknown'
LIMIT 1
```

   Решението во Cypher е дадено во прилог:

```
MATCH (d:Director)-[directs:DIRECTS]->(m:Movie)
WHERE d.firstName <> "Unknown"
WITH d, count(m) AS NUM_MOVIES
ORDER BY NUM_MOVIES DESC
RETURN head(collect(d.firstName + ' ' + d.lastName)) AS DIRECTOR,
head(collect(NUM_MOVIES)) AS NUM_MOVIES
```

4. Да се најдат сите глумци кои настапиле во некој филм во улога на себе сѝ но не во филмови со жанр 'Documentary' и 'Short'.
   Решението во SQL е дадено во прилог:

```
SELECT CONCAT(A.first_name,' ',A.last_name) AS ACTOR
FROM IMDB.actors A
WHERE EXISTS (
SELECT *
FROM IMDB.roles R, IMDB.movies_genres MG
WHERE R.actor_id = A.id
AND MG.movie_id = R.movie_id
AND R.role IN ('Himself','Herself')
AND MG.genre NOT IN ('Documentary','Short')
)
```

   Решението во Cypher е дадено во прилог:

```
MATCH (a:Actor)-[acts:ACTS]->(m:Movie)-[:HAS_GENRE]->(g:Genre)
WHERE NOT g.genre IN ["Documentary", "Short"]
AND ( "Himself" IN acts.roles
OR "Herself" IN acts.roles )
WITH a, count(m) AS NUM_MOVIES
WHERE NUM_MOVIES > 0
RETURN a.firstName + ' ' + a.lastName AS ACTOR
ORDER BY ACTOR
```

5. Да се најде парот глумец со име Jack и режисер кој најмногу пати соработувале.
Решението во SQL е дадено во прилог:

```
SELECT CONCAT(A.first_name,' ',A.last_name) AS ACTOR,
CONCAT(D.first_name,' ',D.last_name) AS DIRECTOR,
COUNT(R.movie_id) AS NUM_MOVIES
FROM IMDB.actors A, IMDB.roles R,
IMDB.movies_directors MD, IMDB.directors D
WHERE A.id = R.actor_id
AND R.movie_id = MD.movie_id
AND MD.director_id = D.id
AND A.first_name = 'Jack'
AND D.first_name != 'Unknown'
GROUP BY ACTOR, DIRECTOR
ORDER BY NUM_MOVIES DESC
LIMIT 1
```

Решението во Cypher е дадено во прилог:

```
MATCH (a:Actor { firstName: 'Jack' })-[acts:ACTS]->(m:Movie)
<-[directs:DIRECTS]-(d:Director)
WITH a, d, count(m) AS NUM_MOVIES
WHERE d.firstName <> "Unknown"
WITH a.firstName + " " + a.lastName AS ACTOR,
d.firstName + " " + d.lastName AS DIRECTOR, NUM_MOVIES
WHERE ACTOR <> DIRECTOR
RETURN ACTOR, DIRECTOR, NUM_MOVIES
ORDER BY NUM_MOVIES DESC
LIMIT 1
```

6. Да се најдат сите режисери кои режираат акција со веројатност поголема од 0.9 и соработувале со повеќе од 100 различни глумци.
Решението во SQL е дадено во прилог:

```
SELECT CONCAT(D.first_name,' ', D.last_name) AS DIRECTOR,
COUNT(DISTINCT actor_id) AS NUM_ACTORS
FROM IMDB.directors D, IMDB.movies_directors MD,
IMDB.directors_genres DG, IMDB.roles R
WHERE MD.director_id = D.id
AND DG.director_id = D.id
AND R.movie_id = MD.movie_id
AND DG.genre = 'Action'
AND DG.prob >= 0.9
AND D.first_name != 'Unknown'
GROUP BY DIRECTOR
HAVING NUM_ACTORS > 100
ORDER BY NUM_ACTORS DESC
```

Решението во Cypher е дадено во прилог:

```
MATCH (a:Actor)-[:ACTS]->(m:Movie)
<-[:DIRECTS]-(d:Director)-[dg:DIRECTS_GENRE]
->(g:Genre {genre: "Action"})
```

```
WHERE toFloat(dg.probabillity) >= 0.9
WITH d, count(DISTINCT a) AS NUM_ACTORS
WHERE NUM_ACTORS > 100
RETURN d.firstName + ' ' + d.lastName AS DIRECTOR, NUM_ACTORS
ORDER BY NUM_ACTORS DESC
```

7. Да се најде парот на режисери кои соработувале најмногу, притоа презимето на едниот режисер да почнува на буквата 'H' а кај другиот на буквата 'B'.
   Решението во SQL е дадено во прилог:

```
SELECT CONCAT(D1.first_name,' ',D1.last_name) AS DIRECTOR1,
CONCAT(D2.first_name,' ',D2.last_name) AS DIRECTOR2,
COUNT(DISTINCT MD1.movie_id) AS NUM_MOVIES
FROM IMDB.movies_directors MD1, IMDB.movies_directors MD2,
IMDB.directors D1, IMDB.directors D2
WHERE MD1.director_id != MD2.director_id
AND MD1.movie_id = MD2.movie_id
AND D1.id = MD1.director_id
AND D2.id = MD2.director_Id
AND D1.last_name LIKE 'H\%'
AND D2.last_name LIKE 'B\%'
GROUP BY DIRECTOR1, DIRECTOR2
ORDER BY NUM_MOVIES DESC
LIMIT 1
```

   Решението во Cypher е дадено во прилог:

```
MATCH (m:Movie)<-[:DIRECTS]-(d:Director)
WITH m, collect(d) AS DIRECTORS
WHERE size(DIRECTORS) = 2
WITH DIRECTORS, count(m) AS NUM_MOVIES
WHERE (left(DIRECTORS[0].lastName,1)='H'
AND left(DIRECTORS[1].lastName,1)='B')
OR (left(DIRECTORS[0].lastName,1)='B'
AND left(DIRECTORS[1].lastName,1)='H')
RETURN DIRECTORS[0].firstName+" "+DIRECTORS[0].lastName AS DIRECTOR_1,
DIRECTORS[1].firstName+" "+DIRECTORS[1].lastName AS DIRECTOR_2, NUM_MOVIES
ORDER BY NUM_MOVIES DESC
LIMIT 1
```

8. Да се најдат сите режисери, чие име почнува на буквата 'M' и кои во 80тите години соработувале во барем еден филм со Винона Рајдер, но никогаш не соработувале со Николас Кејџ.
   Решението во SQL е дадено во прилог:

```
SELECT D.*
FROM IMDB.directors D
WHERE D.first_name LIKE 'M\%'
AND EXISTS (
SELECT *
FROM IMDB.movies_directors MD, IMDB.movies M,
IMDB.roles R, IMDB.actors A
WHERE MD.director_id = D.id
```

```
AND M.id = MD.movie_id
AND R.movie_id = MD.movie_id
AND A.id = R.actor_id
AND M.year BETWEEN 1980 AND 1990
AND CONCAT(A.first_name,' ',A.last_name) = 'Winona Ryder'
)
AND NOT EXISTS (
SELECT *
FROM IMDB.movies_directors MD, IMDB.movies M,
IMDB.roles R, IMDB.actors A
WHERE MD.director_id = D.id
AND M.id = MD.movie_id
AND R.movie_id = MD.movie_id
AND A.id = R.actor_id
AND M.year BETWEEN 1980 AND 1990
AND CONCAT(A.first_name,' ',A.last_name) = 'Nicolas Cage'
)
```

Решението во Cypher е дадено во прилог:

```
MATCH (a:Actor)-[:ACTS]->(m:Movie)<-[:DIRECTS]-(d:Director)
WHERE 1980 <= toInteger(m.year) <= 1990
AND left(d.firstName,1)='M'
WITH d, collect(DISTINCT a.firstName + " " + a.lastName) AS ACTORS
WHERE "Winona Ryder" IN ACTORS
AND NOT "Nicolas Cage" IN ACTORS
RETURN DISTINCT d.firstName + " " + d.lastName AS DIRECTOR
```

9. За секоја актерка што се презива Smith да се пресмета ранк така што ранкот е дефиниран како просек од ранковите од сите филмови во која глумела.
Решението во SQL е дадено во прилог:

```
SELECT CONCAT(A.first_name,' ',A.last_name) AS ACTRESS,
AVG(M.rank) AS AVG_RANK
FROM IMDB.actors A, IMDB.roles R, IMDB.movies M
WHERE A.gender = 'F'
AND A.last_name = 'Smith'
AND R.actor_id = A.id
AND M.id = R.movie_id
AND M.rank != '\N'
GROUP BY ACTRESS
ORDER BY AVG_RANK DESC
```

Решението во Cypher е дадено во прилог:

```
MATCH (a:Actor { gender: "F", lastName: "Smith" })-[:ACTS]->(m:Movie)
WHERE m.rank <> "\\N"
WITH a, m, toFloat(m.rank) AS RANK
RETURN a.firstName + " " + a.lastName, avg(RANK) AS RANK
ORDER BY RANK DESC
```

10. За секој режисер да се пресмета односот меѓу рангот на неговата најдобра драма во однос на рангот на најдобриот филм на Милчо Манчевски.

Решението во SQL е дадено во прилог:

```
SELECT CONCAT(D.first_name,' ',D.last_name) AS DIRECTOR,
MAX(M.rank)/(
SELECT MAX(M.rank)
FROM IMDB.movies M, IMDB.directors D, IMDB.movies_directors MD
WHERE M.id = MD.movie_id
AND M.rank != '\N'
AND MD.director_id = D.id
AND D.last_name = 'Manchevski'
) AS MAX_RANK
FROM IMDB.directors D, IMDB.movies_directors MD,
IMDB.movies_genres MG, IMDB.movies M
WHERE MD.director_id = D.id
AND MG.movie_id = MD.movie_id
AND MG.genre = 'Drama'
AND M.id = MD.movie_id
AND M.rank != '\N'
GROUP BY DIRECTOR
ORDER BY MAX_RANK DESC, DIRECTOR
```

Решението во Cypher е дадено во прилог:

```
MATCH (milcho:Director { firstName: "Milcho",
lastName: "Manchevski" })-[:DIRECTS]->(m:Movie)
MATCH (d:Director)-[:DIRECTS]->(m1:Movie)
-[:HAS_GENRE]->(g:Genre {genre: "Drama"})
WHERE m1.rank <> "\\N"
RETURN d.firstName + " " + d.lastName AS DIRECTOR,
max(toFloat(m1.rank))/max(toFloat(m.rank)) AS RANK
ORDER BY RANK DESC, DIRECTOR
```

2.4 Споредба на перформансите

Сите прашања беа извршени во двете технологии инсталирани на вкупно три машини, за понатаму да може да се анулираат разликите во перформансите на различни компјутери. Со помош на алатките за прикажување на резултатите од секој прашалник, беа запишани соодветните времиња на извршување, во единица секунди. Сите резултати беа организирани на структуриран начин во Excel табела, прикажана на слика 5.
Од овие податоци, со примена на релевантни статистики може да се добие нивна покомпресирана форма, каде што просекот се пресметува со цел да се израмнат разликите во времето на извршување на некое query помеѓу секој од членовите на тимот. На слика 6 се прикажани овие покорисни резултати.

	DETAILS	Andrej											Aleksandar											Nenad										
	Complexity	Simple				Medium				Complex		An	Simple				Medium				Complex		Al	Simple				Medium				Complex		Ne
	Query	1	2	3	4	1	2	3	4	1	2	avg	1	2	3	4	1	2	3	4	1	2	avg	1	2	3	4	1	2	3	4	1	2	avg
Neo4j	IMDB	0.06	0.80	0.85	55.0	0.40	0.11	1.50	0.45	0.60	1.90	6.17	0.07	0.40	0.80	65.0	0.60	0.18	2.10	0.30	0.54	1.45	7.14	0.07	0.40	0.80	65.0	0.60	0.11	1.50	0.45	0.60	1.90	7.14
	EuroVision	0.01	0.00	0.02	0.01	0.10	0.00	0.06	0.06	0.13	0.00	0.04	0.03	0.00	0.01	0.00	0.06	0.00	0.03	0.03	0.09	0.00	0.03	0.02	0.00	0.03	0.03	0.03	0.10	0.05	0.16	0.14	0.00	0.06
MySQL	IMDB	0.11	0.64	2.75	1.00	3.65	1.65	9.15	5.30	0.10	4.75	2.91	1.27	2.30	1.30	0.60	0.38	0.63	0.89	1.70	0.08	1.70	1.08	9.00	1.40	3.78	2.11	0.90	1.84	3.28	5.44	0.17	3.20	3.11
	EuroVision	0.00	0.00	0.00	0.02	0.15	0.03	0.05	0.05	1.30	0.14	0.17	0.08	0.03	0.02	0.02	0.12	0.08	0.02	0.02	1.42	0.09	0.19	0.00	0.00	0.00	0.00	0.33	0.39	0.03	0.32	0.34	0.24	0.16
sum	IMDB	0.17	1.44	3.60	56.0	4.05	1.76	10.6	5.75	0.70	6.65		1.33	2.70	2.10	65.6	0.98	0.81	2.99	2.00	0.62	3.15		9.07	1.80	4.58	67.1	1.50	1.95	4.78	5.89	0.77	5.10	
	EuroVision	0.01	0.00	0.02	0.03	0.25	0.03	0.10	0.10	1.43	0.14		0.11	0.03	0.03	0.02	0.18	0.08	0.04	0.05	1.51	0.10		0.02	0.00	0.03	0.03	0.35	0.49	0.08	0.48	0.48	0.24	

Слика 5: Целосен приказ со сите времиња на извршување

3 Резултати

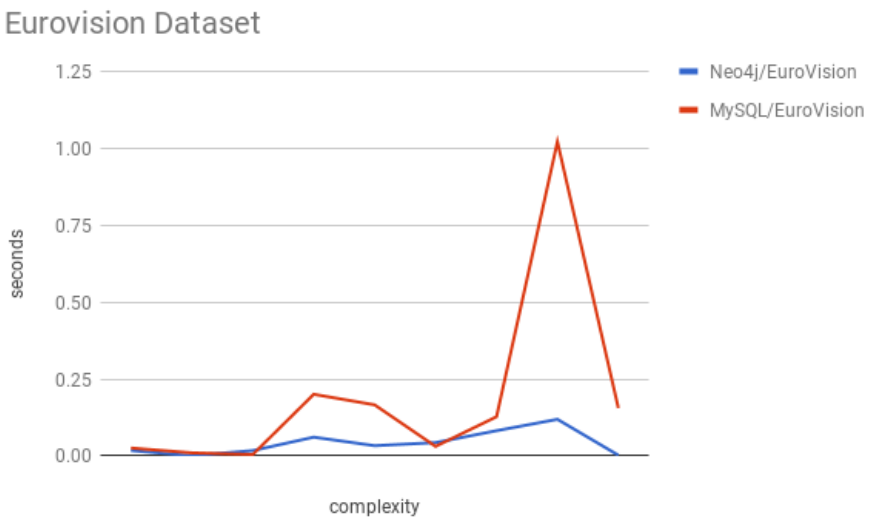
3.1 Статистичка обработка

По извршените мерења, следниот чекор се состоеше од обработка на податоците колектирани од извршувањето на прашањата. Беа изведени проседи по прашање, проседи по база и технологија, како и сума, просек и медијана по категорија на комплексност.
На слика 6 се прикажани овие податоци.

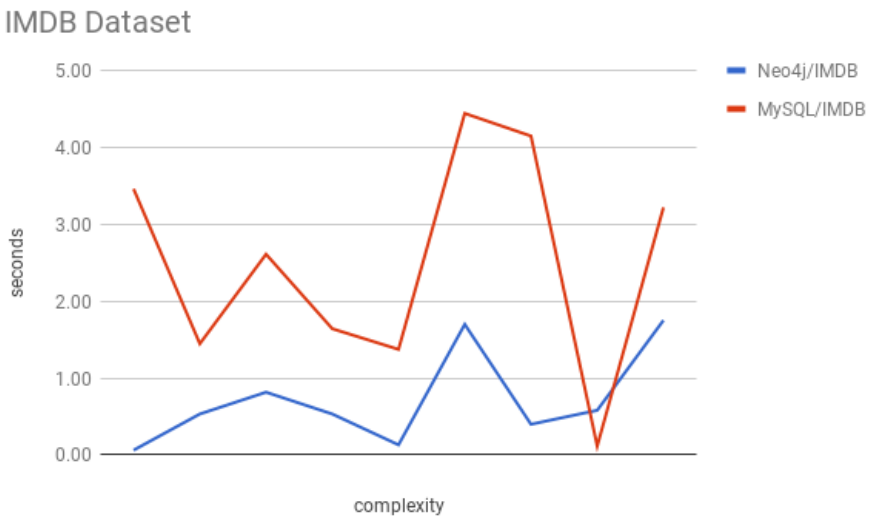
	DETAILS	Total									Average from everyone									
	Complexity	Simple			Medium			Complex			Simple				Medium				Complex	Ne
	Query	sum	avg	median	sum	avg	median	sum	avg	median	1	2	3	4	1	2	3	4	1	2	avg
Neo4j	IMDB	189.24	15.77	0.80	8.29	0.69	0.45	7.00	1.17	1.03	0.06	0.53	0.82	61.6	0.53	0.13	1.70	0.40	0.58	1.75	6.82
	EuroVision	0.16	0.01	0.01	0.66	0.06	0.05	0.37	0.06	0.05	0.02	0.00	0.02	0.02	0.06	0.03	0.04	0.08	0.12	0.00	0.04
MySQL	IMDB	26.26	2.19	1.35	34.80	2.90	1.77	10.00	1.67	0.94	3.46	1.45	2.61	1.24	1.64	1.37	4.44	4.15	0.12	3.22	2.37
	EuroVision	0.15	0.01	0.00	1.58	0.13	0.06	3.53	0.59	0.29	0.03	0.01	0.01	0.01	0.20	0.17	0.03	0.13	1.02	0.16	0.18

Слика 6: Изведени статистики за извршувањето на секое прашање

Со цел поубаво да се потенцираат разликите во комплексноста и времето на извршување на сценаријата беа направени две визуелизации, по една за секоја од базите, прикажувајќи ја зависноста меѓу претходноспоменатите величини во форма на line chart:

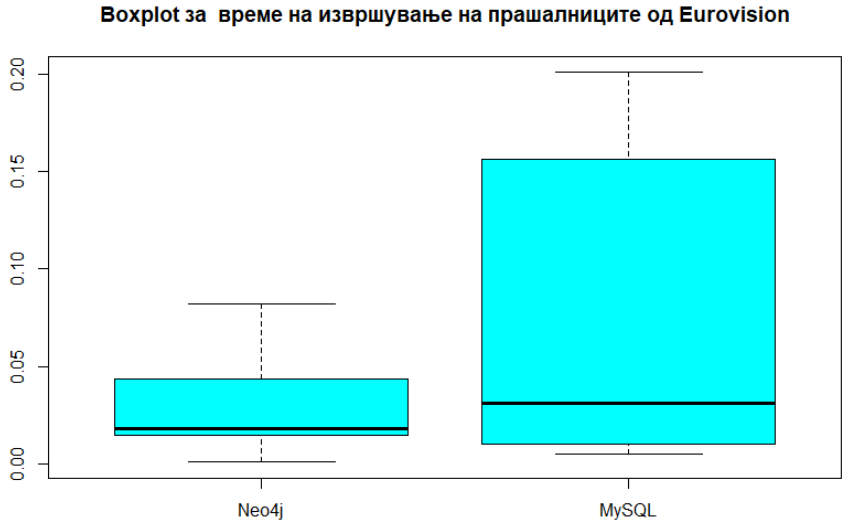


Слика 7: Зависност меѓу комплексноста на прашалниците за Евровизија и нивното време на извршување

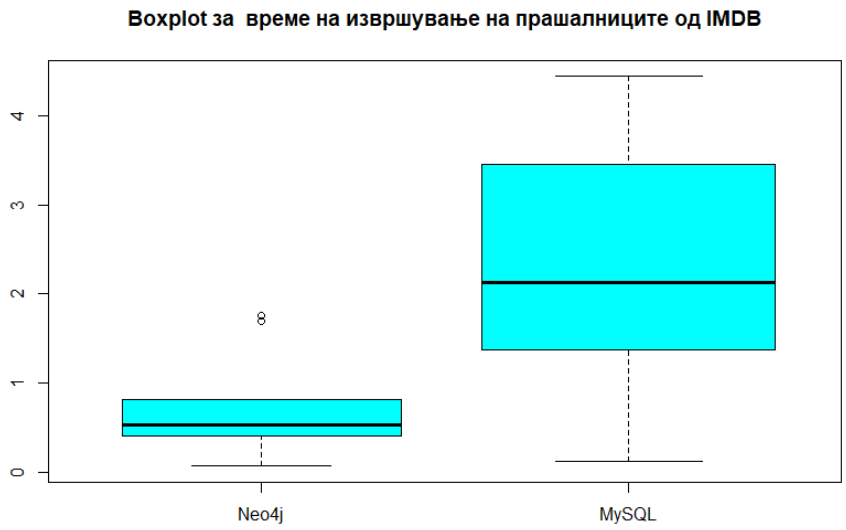


Слика 8: Зависност меѓу комплексноста на прашалниците за IMDB и нивното време на извршување

Со примена на Студентов t-тест за споредба на математички очекувања, беше проверено дали постои статистичка значајност кај разликата во перформансите на Neo4j и MySQL кај двете бази на податоци посебно. Бидејќи кај овој тест зависи дали имаме податок дека се еднакви дисперзиите на величините, претходно направен беше и Фишеровиот f-тест за споредба на дисперзии. Отстранети беа и сите екстремни вредности со цел тестовите да бидат непристрасни, а нивото на доверба беше поставено на 99%. Анализите беа направени користејќи го R пакетот и ги извршува скриптата `measurements-statistics.R`.
Кај податоците од Евровизија не беше откриена значајност и може да се смета дека двете технологии имаат еднакви перформанси. Но, кај IMDB податоците се изразуваат подобрите перформанси на Neo4j и се отфрла хипотезата за еднаквост со p-вредност еднаква на 0.0026. На сликите 9 и 10 преку box plot се споредени перфомансите на двете технологии. Високата скалабилност на NoSQL базите на податоци се одразува и врз резултатите, имајќи ја предвид огромната количина на податоци зачувани во IMDB базата.



Слика 9: Boxplot за споредба на времињата на извршување кај Eurovision



Слика 10: Boxplot за споредба на времињата на извршување кај IMDB

3.2 Коментари околу користењето на податоците

Во прилог подетално ќе разгледаме мал дел од прашањата кои се од посебен интерес бидејќи предизвикуваат да испливаат на површината предностите и слабостите на технологиите со кои беше работено.

- Прашањето со бр. 2 - Евровизија: Релативно едноставно, во SQL при извршување доволно е спојување на 3 табели, а во Neo4j спојување се прави на скоро целиот граф, но и покрај тоа Neo4j покажува еднакви перформанси со SQL.
- Прашањето со бр. 4 - Евровизија: Недостаток во Neo4j е неможност на користење на повеќе агрегатни функции истовремено и потребна е синтаксата со `REDUCE` функци-јата, за разлика од SQL каде тоа лесно се постигнува, но и покрај тоа Neo4j покажува еднакви перформанси со SQL.
- Прашањето со бр. 6 - Евровизија: Имплементацијата во двете технологии е драстично различна. Причината е непостоењето на `NOT EXISTS` механизмот во Neo4j.
- Прашања со бр. 7 и 8 - Евровизија: Двете технологии имаат исти перформанси при користење на сортирање и филтрирање.
- Прашањето со бр. 9 - Евровизија: Додавање на дополнителен атрибут за групирање има значителен негативен ефект врз перформансите на SQL, но кај Neo4j не е толку изразено.
- Прашањето со бр. 3 - IMDB: Во Neo4j се користи функцијата `head` кое го дава првиот елемент од групираните имиња. За постигнување на ова мора да се направи подредувањето пред `RETURN` делот.
- Прашања со бр. 4 и 9 - IMDB: Се согледува слабоста на Neo4j при генерирање на извештаи за кои треба да се изминат скоро сите јазли во графот. За четвртото прашање потребно е најдолго време за добивање на резултати од Neo4j и тука MySQL е значително побрз.
- Прашања со бр. 8 и 10 - IMDB: Имплементацијата во SQL е премногу комплексна и долга во споредба со Neo4j.

4 Заклучок

Според резултатите прикажани во претходната секција, може да се заклучи дека иако граф базираните бази на податоци не се совршени и во некои сценарија не се покажуваат како најефикасни во нашите прашања, за најголем процент од актуелните потреби како BigData Analytics, модерните социјални мрежи, моделирачки софтвери и сл. се значително побрзи од традиционалниот пристап.
Секако, секоја технологија е само алатка и изборот на алатката зависи од проблемот со кој се соочуваме. Понатаму, треба да се нагласи дека секогаш постои опцијата и да се користи хибридна база на податоци, односно мешавина меѓу граф-базирана и релациона структура, задржувајќи го најдоброто од двата света.

Литература

[1] 2018, Oracle Corporation and/or its affiliates, "Chapter 3: Installation Workflow with MySQL Installer"
https://dev.mysql.com/doc/mysql-installer/en/mysql-installer-workflow.html

[2] 2018, Neo4j, Inc., "Chapter 1. Introduction"
https://neo4j.com/docs/developer-manual/current/introduction/#introduction-highlights

[3] 2018, Neo4j, Inc., "2.4. Windows installation"
https://neo4j.com/docs/operations-manual/current/installation/windows/

[4] 2018, Neo4j, Inc., "Chapter 3. Cypher"
https://neo4j.com/docs/developer-manual/current/cypher/