TP Neo4J

Gaetan Corin Lou Doussiet Master2 DataEngineer

Chargement de la base de données Movielens:
(installation préalable des fichiers csv dans neo4J)

requêtes utilisés pour charger les NODES:

```
LOAD CSV WITH HEADERS FROM 'file:///users.csv' AS users
CREATE (u:User {
    id: toInteger(users.id),
    sex: users.sex,
    age: toInteger(users.age),
    occupation: users.occupation,
    zip_code: users.zip_code
});

LOAD CSV WITH HEADERS FROM 'file:///movies.csv' AS movies
CREATE (m:Movie {
    id: toInteger(movies.movieId),
    title: movies.title,
    date: toInteger(movies.date)
});

LOAD CSV WITH HEADERS FROM 'file:///genre.csv' AS genres
CREATE (u:Genre {
    id: toInteger(genres.id),
    name: genres.name
});
```

requêtes utilisés pour charger les EDGES:

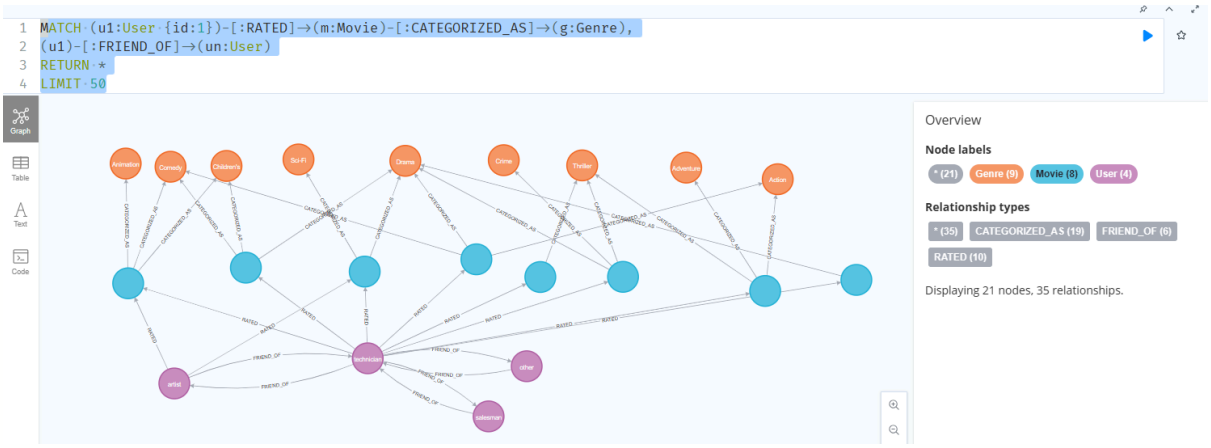
```
LOAD CSV WITH HEADERS FROM 'file:///mov_genre.csv' AS mov_genre
MERGE (m:Movie {id: toInteger(mov_genre.mov_id)})
MERGE (g:Genre {id: toInteger(mov_genre.genre)})
CREATE (m)-[:CATEGORIZED_AS]->(g);

LOAD CSV WITH HEADERS FROM 'file:///ratings.csv' AS ratings
MERGE (u:User {id: toInteger(ratings.user_id)})
MERGE (m:Movie {id: toInteger(ratings.mov_id)})
CREATE (u)-[:RATED {note: toInteger(ratings.rating), timestamp:
toInteger(ratings.timestamp)}]->(m);
```

```
LOAD CSV WITH HEADERS FROM 'file:///friends.csv' AS friends
MERGE (u1:User {id: toInteger(friends.user1_id)})
MERGE (u2:User {id: toInteger(friends.user2_id)})
CREATE (u1)-[:FRIEND_OF]->(u2);
```

RESULTAT:

Exemple de requête fonctionnelle donné dans le TP:

```
MATCH (u1:User {id:1})-[:RATED]->(m:Movie)-[:CATEGORIZED_AS]->(g:Genre),
(u1)-[:FRIEND_OF]->(un:User)
RETURN *
LIMIT 50
```



Questions portant sur les noeuds (nodes):

Question 1. Combien y a-t-il d'utilisateurs dans la base de données ?

```
MATCH (u:User)
RETURN COUNT(u);
```

Réponse: 200 User

Question 2. Quel est l'âge et le sexe de l'utilisateur qui a pour id la valeur 1?

```
MATCH (u1:User {id:1})
RETURN u1.age, u1.sex;
```

Réponse: age:24, sex:'M
'

Question 3.1. Quelles sont les femmes qui sont artistes (occupation='artist') ?

```
MATCH (u:User {sex:"F", occupation:'artist'})
RETURN u.id
```

Réponse: (en Id) 23, 24, 96, 123, 150

Question 3.2. Quelles sont les femmes qui sont artistes ou auteurs (occupation='writer') ?

```
MATCH (u:User {sex: "F"})
WHERE u.occupation = 'artist' OR u.occupation = 'writer'
RETURN u.id;
```

Réponse: (en Id) 23, 24, 96, 122, 123, 150

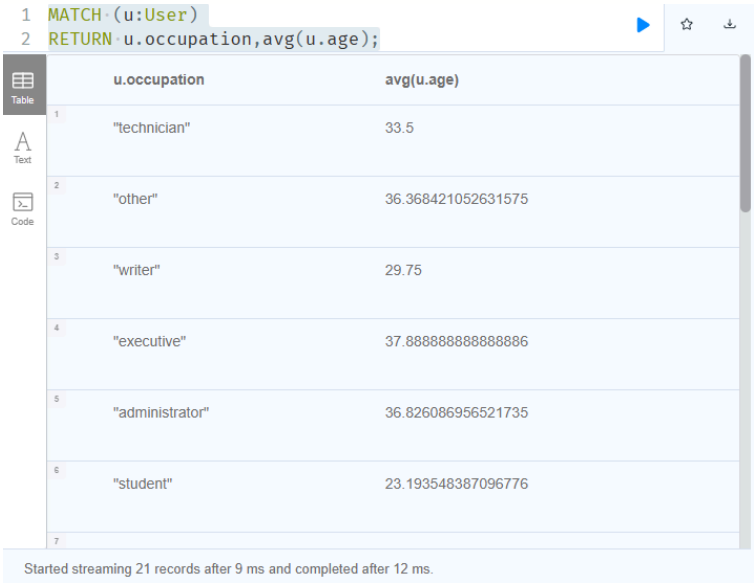
Question 4. Quel est l'âge moyen des étudiants de la base de données ?

```
MATCH (u:User)
RETURN avg(u.age);
```

Réponse : 34,66

Question 5. Quel est l'âge moyen des utilisateurs pour chaque occupation?

```
MATCH (u:User)
RETURN u.occupation,avg(u.age);
```

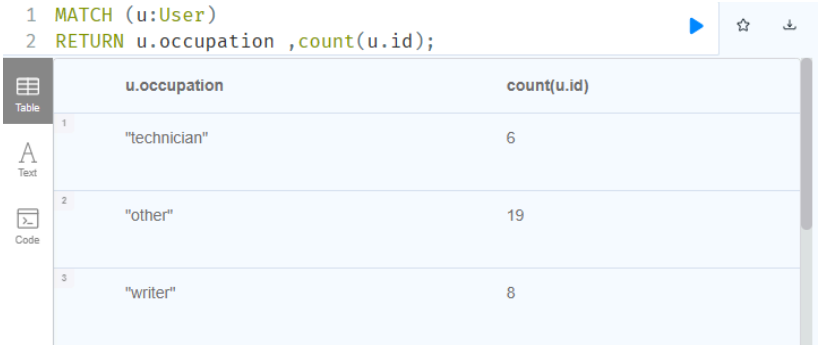


	u.occupation	avg(u.age)
1	"technician"	33.5
2	"other"	36.368421052631575
3	"writer"	29.75
4	"executive"	37.888888888888886
5	"administrator"	36.826086956521735
6	"student"	23.193548387096776
7		

Started streaming 21 records after 9 ms and completed after 12 ms.

Question 6.1. Quel est le nombre d'utilisateurs pour chaque occupation ?

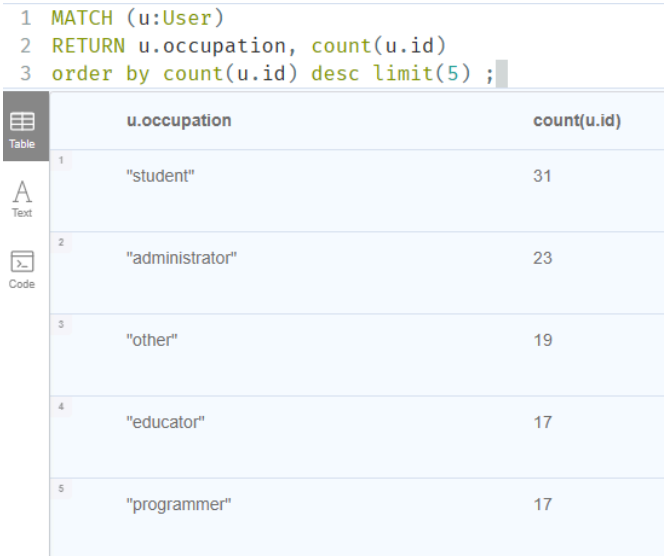
```
MATCH (u:User)
RETURN u.occupation ,count(u.id);
```



	u.occupation	count(u.id)
1	"technician"	6
2	"other"	19
3	"writer"	8

Question 6.2. Quelles sont les 5 occupations les plus populaires ?

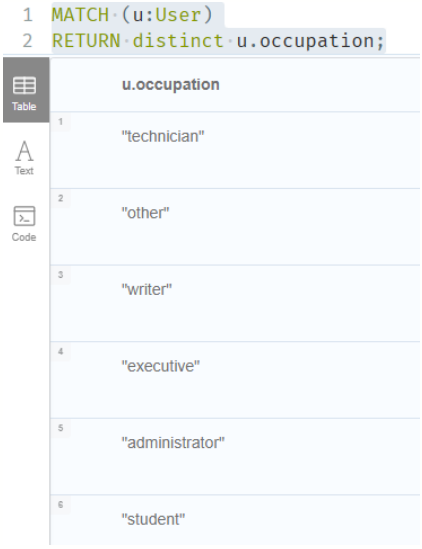
```
MATCH (u:User)
RETURN u.occupation ,count(u.id)
order by count(u.id) desc limit(5) ;
```



	u.occupation	count(u.id)
1	"student"	31
2	"administrator"	23
3	"other"	19
4	"educator"	17
5	"programmer"	17

Question 7. Combien y a-t-il d'occupations différentes dans le jeu de données ?

```
MATCH (u:User)
RETURN distinct u.occupation;
```



	u.occupation
1	"technician"
2	"other"
3	"writer"
4	"executive"
5	"administrator"
6	"student"

Question 8. Combien y a-t-il d'occupations pour lesquelles l'âge moyen des utilisateurs est supérieur à 30 ?

```
MATCH (u:User)
WITH u.occupation AS occupation, avg(u.age) as avgAge
where avgAge >30
RETURN occupation, avgAge;
```

	occupation	avgAge
1	"technician"	33.5
2	"other"	36.368421052631575
3	"executive"	37.888888888888886
4	"administrator"	36.826086956521735

Question 9. Quels sont les films produits en 1995? Notez que la date de production d'un film n'apparaît pas comme propriété des nœuds Movie, mais entre parenthèses à la fin de chaque titre de film ; vous pouvez exploiter cette information avec une expression régulière.

```
MATCH (m:Movie)
where m.title ends with '(1995)'
RETURN m.title
```

	m.title
1	"Toy Story (1995)"
2	"GoldenEye (1995)"
3	"Four Rooms (1995)"
4	"Get Shorty (1995)"

Questions portant sur les relations (edges):

Question 10. Combien de notes contient la base de données ? (Comptez les relation de type RATED)

```
MATCH ()-[r:RATED]->()
RETURN count(r)
```

Réponse: 11310

Question 11. Quelles sont toutes les notes laissées par l'utilisateur qui a pour id la valeur 1 ? Affichez le résultat sous forme de graphe reliant chaque note au film concerné.

```
MATCH (m:Movie)<-[r:RATED]-(u1:User {id:1})
RETURN m.title
```

Cette requête n'a malheureusement pas marché car les noeuds movie ne donnaient pas le titre. Pourtant, les noeuds possédaient bien la colonne title avec les valeurs.

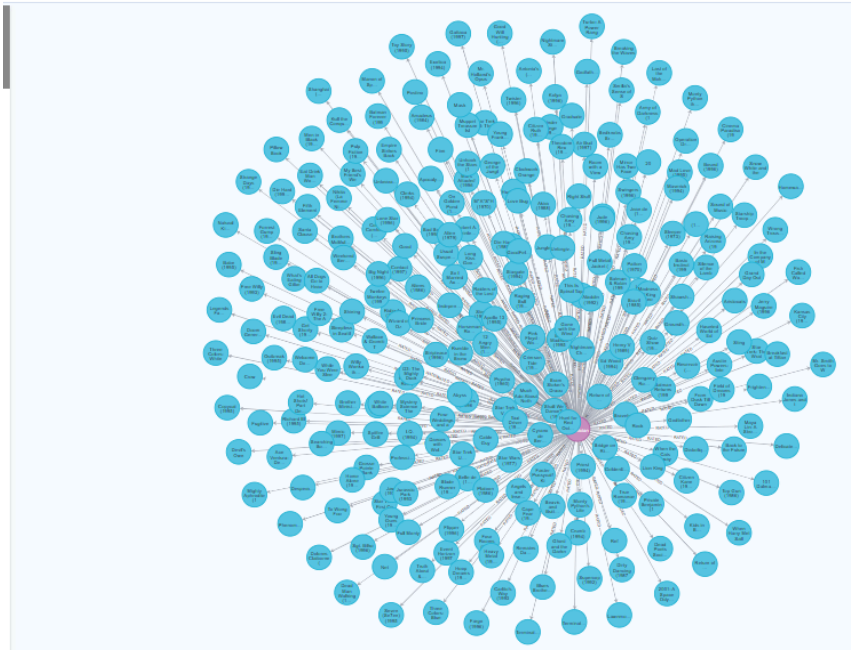
Nous avons donc à nouveau réaliser la base de données, avec une syntaxe différente:

```
LOAD CSV WITH HEADERS FROM 'file:///users.csv' AS users
MERGE (u:User {id : toInteger(users.id)})
SET u.sex = users.sex, u.age = toInteger(users.age), u.occupation =
users.occupation, u.zip_code = users.zip_code;

LOAD CSV WITH HEADERS FROM 'file:///movies.csv' AS movies
MERGE (m:Movie {id : toInteger(movies.id)})
SET m.title = trim(movies.title), m.date = toInteger(movies.date);

LOAD CSV WITH HEADERS FROM 'file:///genre.csv' AS genres
MERGE (g:Genre {id : toInteger(genres.id)})
SET g.name = genres.name;

LOAD CSV WITH HEADERS FROM 'file:///mov_genre.csv' AS mov_genre
MERGE (m:Movie {id : toInteger(mov_genre.mov_id)})
MERGE (g:Genre {id : toInteger(mov_genre.genre)})
CREATE (m)-[:CATEGORIZED_AS]->(g);

LOAD CSV WITH HEADERS FROM 'file:///ratings.csv' AS ratings
MERGE (u:User {id : toInteger(ratings.user_id)})
MERGE (m:Movie {id : toInteger(ratings.mov_id)})
CREATE (u)-[:RATED {note : toInteger(ratings.rating), timestamp :
toInteger(ratings.timestamp)}]->(m);
```

```
LOAD CSV WITH HEADERS FROM 'file:///friends.csv' AS friends
MERGE (u1:User {id : toInteger(friends.user1_id)})
MERGE (u2:User {id : toInteger(friends.user2_id)})
CREATE (u1)-[:FRIEND_OF]->(u2);
```

Nous avons ensuite réalisé à nouveau la requête:

```
MATCH a =(m:Movie)<-[r:RATED]-(u1:User {id:1})
RETURN a
```



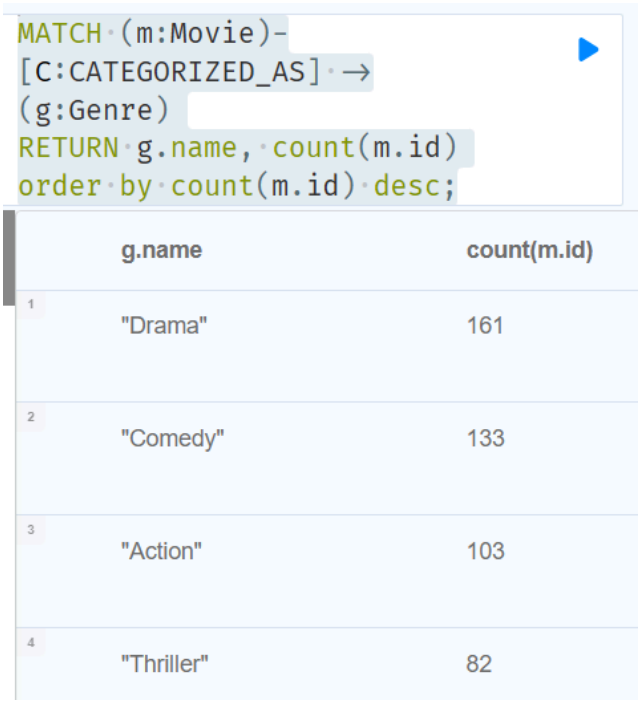
Question 12. Combien d'utilisateurs ont note le film intitulé "Braveheart (1995)" ?

```
MATCH (User)-[r:RATED] ->(m:Movie)
WHERE m.title = 'Braveheart (1995)'
RETURN DISTINCT count(User.id)
```

Réponse: 52

Question 13. Quels sont les cinq genres les plus représentés parmi les films de la base de données ?

```
MATCH (m:Movie)-[C:CATEGORIZED_AS] ->(g:Genre)
RETURN g.name, count(m.id)
order by count(m.id) desc;
```

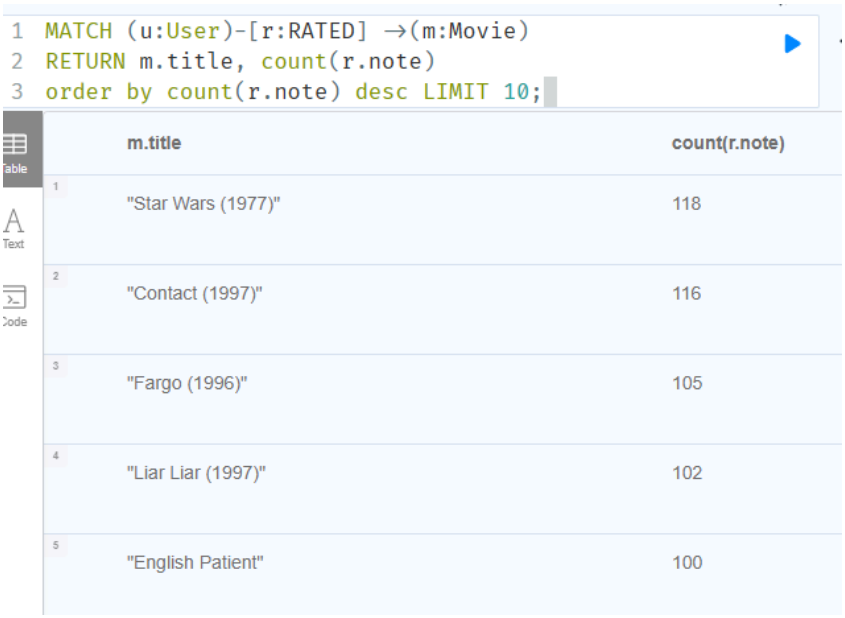


```
MATCH (m:Movie)-
[C:CATEGORIZED_AS] →
(g:Genre)
RETURN g.name, count(m.id)
order by count(m.id) desc;
```

	g.name	count(m.id)
1	"Drama"	161
2	"Comedy"	133
3	"Action"	103
4	"Thriller"	82

Question 14. Quels sont les dix films qui ont reçu le plus de notes ?

```
MATCH (u:User)-[r:RATED] ->(m:Movie)
RETURN m.title, count(r.note)
order by count(r.note) desc LIMIT 10;
```



```
1 MATCH (u:User)-[r:RATED] →(m:Movie)
2 RETURN m.title, count(r.note)
3 order by count(r.note) desc LIMIT 10;
```

	m.title	count(r.note)
1	"Star Wars (1977)"	118
2	"Contact (1997)"	116
3	"Fargo (1996)"	105
4	"Liar Liar (1997)"	102
5	"English Patient"	100

Question 15. Combien de films ont reçu au moins une fois la note 1 ?

```
MATCH (u:User)-[r:RATED] ->(m:Movie)
WHERE r.note = 1
RETURN count(m.title)
```

Réponse: 624

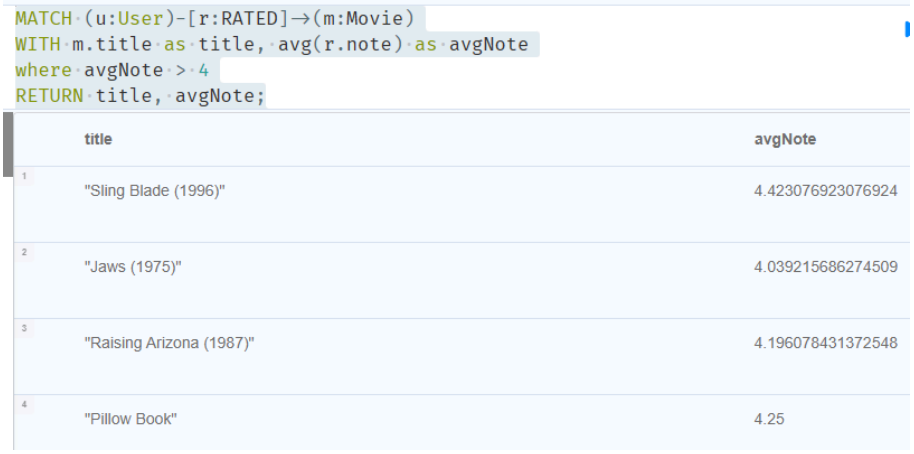
Question 16. Quels sont les films qui ont reçu au moins une fois la note 1, combien de fois ont-ils obtenu cette note, et qui la leur a attribué ? Cette requête doit retourner une ligne par film (donc autant de lignes que la réponse à la question précédente) avec trois colonnes : le titre du film, le nombre de fois où il a reçu la note 1, et la liste des identifiants de chaque utilisateur qui lui a donné cette note.

```
MATCH (u:User)-[r:RATED] ->(m:Movie)
WHERE r.note = 1
RETURN m.title, count(r.note), COLLECT(u.id) AS liste_utilisateurs
```

	m.title	count(r.note)	liste_utilisateurs
1	"Batman Returns (1992)"	3	[1, 110, 151]
2	"Jungle2Jungle (1997)"	10	[1, 5, 15, 17, 59, 92, 142, 177, 181, 199]
3	"Turbo: A Power Rangers Movie (1997)"	1	[1]

Question 17. Quels sont les films dont la note moyenne dépasse 4.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WITH m.title as title, avg(r.note) as avgNote
where avgNote > 4
RETURN title, avgNote;
```



	title	avgNote
1	"Sling Blade (1996)"	4.423076923076924
2	"Jaws (1975)"	4.039215686274509
3	"Raising Arizona (1987)"	4.196078431372548
4	"Pillow Book"	4.25

Question 18. Quels sont les films qui ont la note moyenne maximale (note = 5) ?

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WITH m.title as title, avg(r.note) as avgNote
where avgNote > 4
RETURN title, avgNote;
```

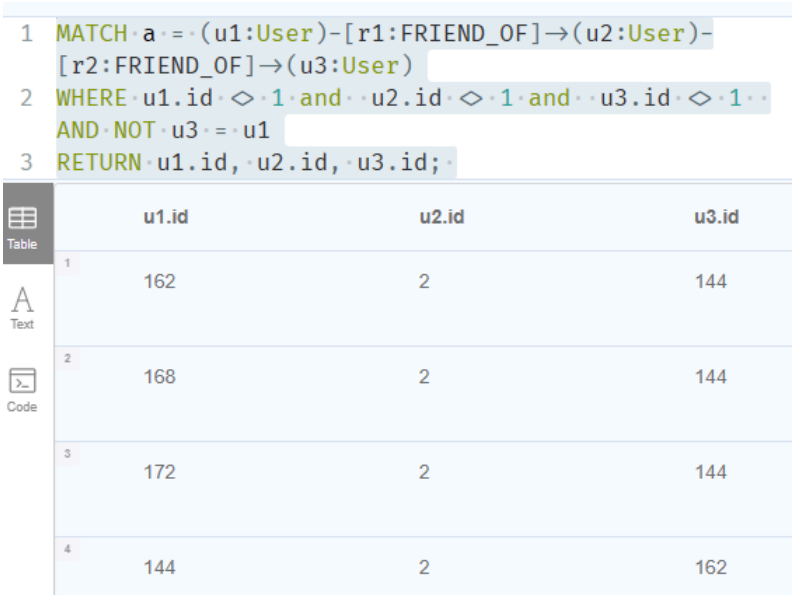
```
MATCH (u:User)-[r:RATED]→(m:Movie)
WITH m.title as title, avg(r.note) as avgNote
where avgNote = 5
RETURN title, avgNote;
```

	title	avgNote
1	"Horseman on the Roof"	5.0
2	"Maya Lin: A Strong Clear Vision (1994)"	5.0
3	"Incognito (1997)"	5.0

Requêtes avancées :

Question 19. Qui sont les amis et les amis des amis de l'utilisateur qui a pour id la valeur 1 ? La requête ne doit pas retourner l'utilisateur 1 dans la liste de résultats.

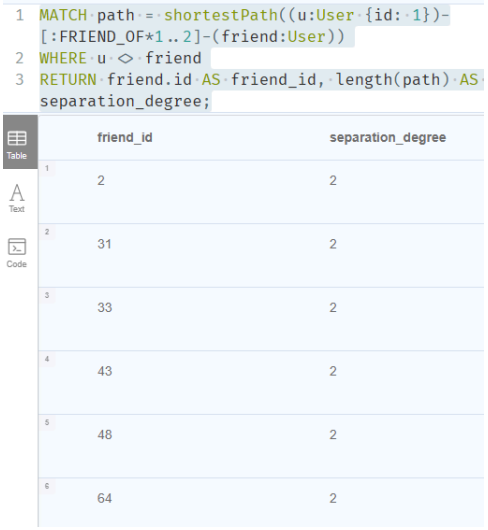
```
MATCH a = (u1:User)-[r1:FRIEND_OF]->(u2:User)-[r2:FRIEND_OF]->(u3:User)
WHERE u1.id <> 1 and  u2.id <> 1 and  u3.id <> 1  AND NOT u3 = u1
RETURN u1.id, u2.id, u3.id;
```



	u1.id	u2.id	u3.id
1	162	2	144
2	168	2	144
3	172	2	144
4	144	2	162

Question 20. Qui sont les amis et les amis des amis de l'utilisateur qui a pour id la valeur 1, et leur degrés de séparation (nombre de relations FRIEND_OF qui les séparent) ?

```
MATCH path = shortestPath((u:User {id: 1})-[:FRIEND_OF*1..2]-(friend:User))
WHERE u <> friend
RETURN friend.id AS friend_id, length(path) AS separation_degree;
```



	friend_id	separation_degree
1	2	2
2	31	2
3	33	2
4	43	2
5	48	2
6	64	2

Question 21. Quel est l'utilisateur qui a le plus d'amis en commun avec l'utilisateur 1 ?

```
MATCH (u1:User {id:1})-[:FRIEND_OF]->(u2:User)<-[:FRIEND_OF]-(u3:User)
WHERE u3 <> u1
RETURN u3.id AS user_id, COUNT(u2) AS amis_communs
ORDER BY amis_communs DESC
```

Réponse: user_id 31

Question 22. Quel est l'utilisateur qui a le plus d'amis en commun avec l'utilisateur 1 mais qui n'est pas déjà son ami ?

```
MATCH (u1:User {id:1})-[:FRIEND_OF]->(u2:User)<-[:FRIEND_OF]-(u3:User)
WHERE u3 <> u1 and u3 <> u2
RETURN u3.id AS user_id, COUNT(u2) AS amis_communs
ORDER BY amis_communs DESC
```

Réponse: user_id 31

Question 23. Quel est l'utilisateur qui a noté le plus grand nombre de films parmi ceux que l'utilisateur 1 a notés, et quel est ce nombre ?

Question 24 : Comment porter le modèle données, utilisé sous neo4J, Dans MongoDB ?

Réponse:
On peut exporter des modèles de données neo4j par normalisation, ou par dénormalisation
La normalisation représente un json par noeuds,
alors que la dénormalisation représente des json avec des noeuds dans des noeuds.

Question 25 : Quelles sont les questions qui ne peuvent pas être exécutées sous MongoDB

Les questions qui ne peuvent pas être exécutées sous mongodb sont: 19,20,21,22