Université Chouaib Doukkali
Ecole Nationale des Sciences Appliquées d'El Jadida
Département Télécommunications, Réseaux et Informatique

Filière : 2ITE
Niveau : 2ème année

Sujet:

Modélisation Movielens Dataset en utilisant une base de données en graphe et construire un moteur de recommandation des films

Realisé par:
DAHHANE DOHA
LAAZIZ Ahmed
KARTBOUNI Anas

Encadré par:
Professeur: HANINE Mohamed

Année universitaire : 2022/2023

Table des matières

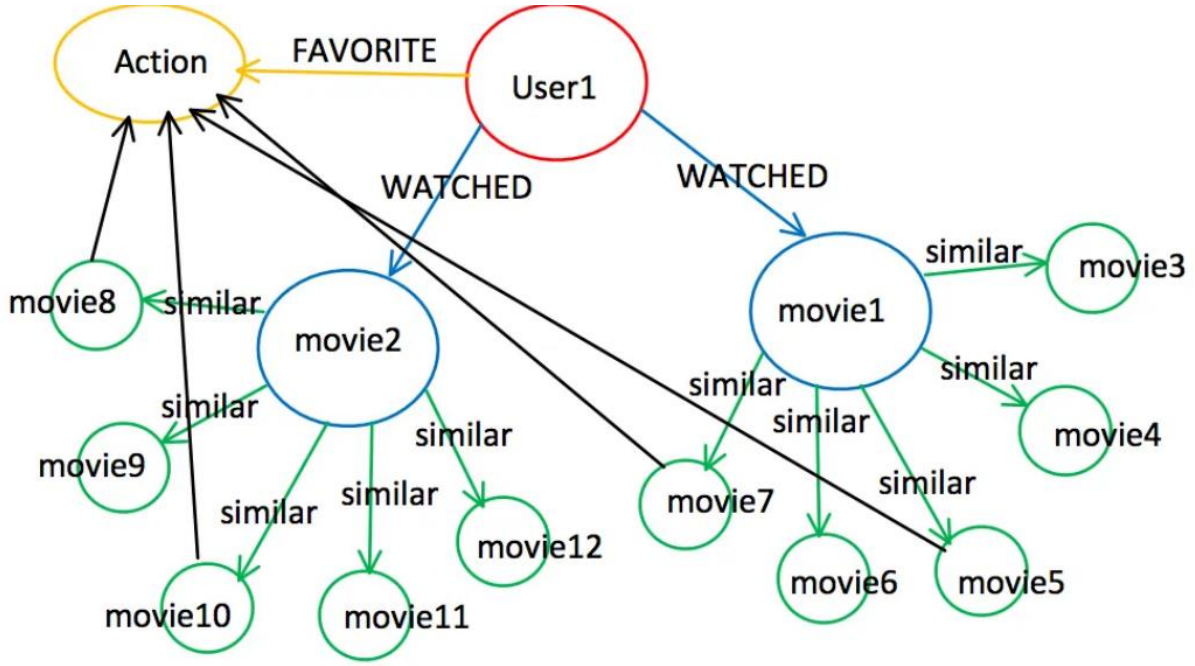
Introduction

Le domaine de la recommandation de films a pris une place prépondérante dans notre société, où l'abondance de contenu disponible rend le choix difficile pour les utilisateurs. Afin de faciliter cette tâche et d'améliorer l'expérience des utilisateurs, les systèmes de recommandation jouent un rôle clé. Dans ce rapport, nous présentons notre projet de création d'un moteur de recommandation de films basé sur la base de données MovieLens et utilisant Neo4j, une base de données orientée graphe.

L'objectif principal de notre projet était de proposer des films qui pourraient plaire à l'utilisateur en se basant sur les films qu'il a déjà appréciés et en tenant compte de son genre favori. Pour atteindre cet objectif, nous avons suivi une approche en plusieurs étapes, commençant par un prétraitement des données MovieLens, suivi de la création d'un graphe Neo4j pour représenter les relations entre les films et les utilisateurs. En utilisant les capacités de requête du langage Cypher, nous avons mis en place un moteur de recommandation performant.

Ce rapport détaillera chaque étape du processus, depuis l'acquisition et le prétraitement des données jusqu'à la conception et la mise en œuvre du moteur de recommandation

1 Traitement des données dans Python

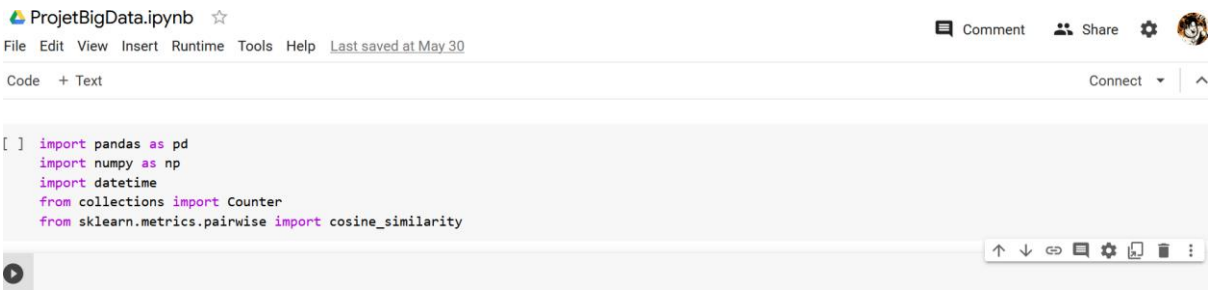


La structure fondamentale de la base de données de recommandation de films ressemble à l'illustration ci-dessous. Elle consiste à trouver les films déjà visionnés par l'utilisateur, puis à identifier les films similaires que l'utilisateur n'a pas encore regardés. Avant de recommander un film, on vérifie également si les genres des films correspondent aux préférences de l'utilisateur. Les films correspondants sont ensuite classés en fonction de leurs évaluations, et les 5 meilleurs résultats sont sélectionnés.

Nous créons 7 ensembles de données pour construire cette structure. Trois d'entre eux sont dédiés aux nœuds (Utilisateurs, Films, Genres) et les autres aux relations.

1.1 Load movielens data

1) On commence par Importer les modules



Nous utilisons 3 ensembles de données : "genome-scores.csv", "movies.csv" et "ratings.csv".

```
[ ]  genome_scores_data = pd.read_csv("/content/genome-scores.csv")
     movies_data = pd.read_csv("/content/movies.csv")
     ratings_data = pd.read_csv("/content/ratings.csv")
```

- **Données d'évaluation**

```
[ ]  genome_scores_data.head()
```

2) On obtient le résultat suivant

	movieId	tagId	relevance
0	1	1	0.02875
1	1	2	0.02375
2	1	3	0.06250
3	1	4	0.07575
4	1	5	0.14075

3) On vérifie le contenu de movies_data :

```
[ ]  movies_data.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

4) On vérifie le contenu de ratings_data

```
[ ]  ratings_data.head()
```

	userId	movieId	rating	timestamp
0	1	296	5.0	1.147880e+09
1	1	306	3.5	1.147869e+09
2	1	307	5.0	1.147869e+09
3	1	665	5.0	1.147879e+09
4	1	899	3.5	1.147869e+09

1.2 Préparation des données pour la matrice de films

1) Nous créons 3 dataframes (mov_tag_df, mov_genres_df, mov_rating_df) et calculons 3 similarités cosinus pour chacun d'entre eux. Ensuite, nous les combinons pour obtenir la matrice de films. Lorsque nous combinons les ensembles de données, nous utilisons une formule (mov_tag_df0.5 + mov_genres_df0.25 + mov_rating_df*0.25). Dans ce cas, les tags sont les données les plus importantes pour calculer la similarité, donc ils devraient avoir plus d'impact sur le calcul de la similarité que les autres. D'accord, commençons à créer les dataframes.

```
[ ]  scores_pivot = genome_scores_data.pivot_table(index = ["movieId"],columns = ["tagId"],values = "relevance").reset_index()
     scores_pivot.head()
```

tagId	movieId	1	2	3	4	5	6	7	8	9	...	1119	1120	1121	1122	1123	1124	
0	1	0.02875	0.02375	0.06250	0.07575	0.14075	0.14675	0.06350	0.20375	0.2020	...	0.04050	0.01425	0.03050	0.03500	0.14125	0.05775	(
1	2	0.04125	0.04050	0.06275	0.08275	0.09100	0.06125	0.06925	0.09600	0.0765	...	0.05250	0.01575	0.01250	0.02000	0.12225	0.03275	(
2	3	0.04675	0.05550	0.02925	0.08700	0.04750	0.04775	0.04600	0.14275	0.0285	...	0.06275	0.01950	0.02225	0.02300	0.12200	0.03475	(
3	4	0.03425	0.03800	0.04050	0.03100	0.06500	0.03575	0.02900	0.08650	0.0320	...	0.05325	0.02800	0.01675	0.03875	0.18200	0.07050	(
4	5	0.04300	0.05325	0.03800	0.04100	0.05400	0.06725	0.02775	0.07650	0.0215	...	0.05350	0.02050	0.01425	0.02550	0.19225	0.02675	(

5 rows × 1129 columns

2) Nous devons joindre le dataframe "scores_pivot" avec le dataframe "movies_data" pour obtenir tous les identifiants de films. Ensuite, nous remplissons les valeurs nulles et supprimons les colonnes qui ne sont pas utilisées

```
[ ] scores_pivot.head()
```

tagId	movieId	1	2	3	4	5	6	7	8	9	...	1119	1120	1121	1122	1123	1124	1
0	1	0.02875	0.02375	0.06250	0.07575	0.14075	0.14675	0.06350	0.20375	0.2020	...	0.04050	0.01425	0.03050	0.03500	0.14125	0.05775	0.03
1	2	0.04125	0.04050	0.06275	0.08275	0.09100	0.06125	0.06925	0.09600	0.0765	...	0.05250	0.01575	0.01250	0.02000	0.12225	0.03275	0.02
2	3	0.04675	0.05550	0.02925	0.08700	0.04750	0.04775	0.04600	0.14275	0.0285	...	0.06275	0.01950	0.02225	0.02300	0.12200	0.03475	0.01
3	4	0.03425	0.03800	0.04050	0.03100	0.06500	0.03575	0.02900	0.08650	0.0320	...	0.05325	0.02800	0.01675	0.03875	0.18200	0.07050	0.01
4	5	0.04300	0.05325	0.03800	0.04100	0.05400	0.06725	0.02775	0.07650	0.0215	...	0.05350	0.02050	0.01425	0.02550	0.19225	0.02675	0.01

5 rows × 1129 columns

mov_genres_df

Le dataframe "mov_genres_df" est créé à partir du fichier "movies.csv". Nous divisons le champ des genres pour chaque film, puis nous créons des colonnes pour chaque genre. Nous définissons une fonction pour diviser la colonne des genres et vérifier si elle existe ou non.

```
mov_tag_df = movies_data.merge(scores_pivot, left_on='movieId', right_on='movieId', how='left')
```

```
mov_tag_df = mov_tag_df.fillna(0)
mov_tag_df = mov_tag_df.drop(['title','genres'], axis = 1)
mov_tag_df.head()
```

movieId	1	2	3	4	5	6	7	8	9	...	1119	1120	1121	1122	1123	1124	1125
0	1	0.02875	0.02375	0.06250	0.07575	0.14075	0.14675	0.06350	0.20375	0.2020	...	0.04050	0.01425	0.03050	0.03500	0.14125	0.05775	0.03900
1	2	0.04125	0.04050	0.06275	0.08275	0.09100	0.06125	0.06925	0.09600	0.0765	...	0.05250	0.01575	0.01250	0.02000	0.12225	0.03275	0.02100
2	3	0.04675	0.05550	0.02925	0.08700	0.04750	0.04775	0.04600	0.14275	0.0285	...	0.06275	0.01950	0.02225	0.02300	0.12200	0.03475	0.01700
3	4	0.03425	0.03800	0.04050	0.03100	0.06500	0.03575	0.02900	0.08650	0.0320	...	0.05325	0.02800	0.01675	0.03875	0.18200	0.07050	0.01625
4	5	0.04300	0.05325	0.03800	0.04100	0.05400	0.06725	0.02775	0.07650	0.0215	...	0.05350	0.02050	0.01425	0.02550	0.19225	0.02675	0.01625

5 rows × 1129 columns

Ce code permet de créer de nouvelles colonnes dans le dataframe mov_genres_df et d'attribuer une valeur à chaque colonne en fonction des genres présents dans la colonne "genres" du dataframe d'origine movies_data.

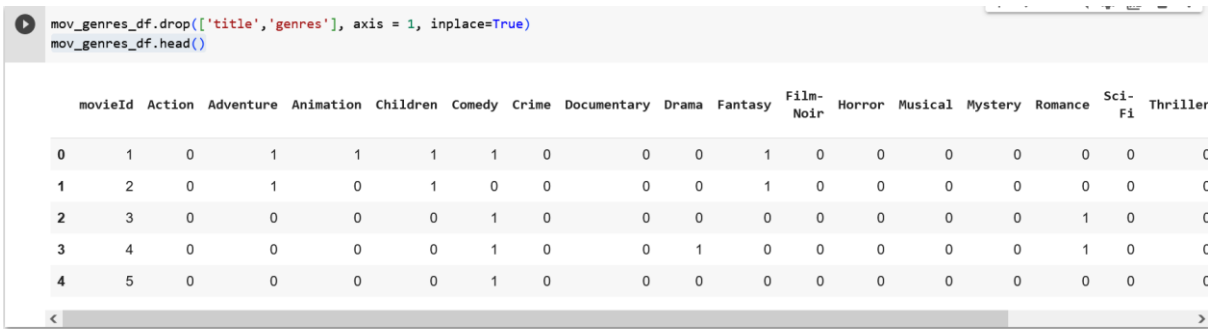
Chaque ligne du dataframe mov_genres_df est traitée individuellement à l'aide de la fonction apply avec une fonction lambda qui utilise la fonction set_genres pour déterminer si le genre spécifique est présent dans la liste des genres de cette ligne. Si le genre est présent, la valeur de la colonne correspondante est définie sur True, sinon elle est définie sur False.

Par exemple, la première ligne de code crée une nouvelle colonne appelée "Action" dans mov_genres_df et utilise la fonction set_genres pour déterminer si le genre "Action" est présent dans la liste des genres de cette ligne. Si le genre "Action" est présent, la valeur de la colonne "Action" pour cette ligne est définie sur True, sinon elle est définie sur False.

3) De cette manière, ce code permet de créer des colonnes binaires pour chaque genre de film possible, indiquant si ce genre est présent ou non pour chaque film dans le

dataframe. Cela facilite l'analyse et la recherche de films en fonction de leurs genres spécifiques

4) Cette commande affichera les autres colonnes qui ont été créées précédemment, telles que 'Action', 'Adventure', 'Animation', etc., qui indiquent si chaque film appartient à ces genres spécifiques



```
mov_genres_df.drop(['title','genres'], axis = 1, inplace=True)
mov_genres_df.head()
```

	movieId	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller
0	1	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0
1	2	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0
2	3	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
3	4	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
4	5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Après le traitement dans Python , nous allons ensuite préparer d'autres ensembles de données pour la base de données de recommandation et en important les données dans la base de données graphique. Enfin, nous rédigerons une requête pour obtenir les meilleurs films recommandés aux utilisateurs.

2 Création du graphe des données dans Neo4j

Maintenant que nos données sont prêts à être importés dans Neo4j. On les exporte d'abord en format csv.

```
neo4j$ users_df.to_csv ('users.csv', sep='|', header=True, index=False)
```

```
neo4j$ movies_df.to_csv ('movies.csv', sep='|', header=True, index=False)
```

```
neo4j$ genres_df.to_csv ('genres.csv', sep='|', header=True, index=False)
```

```
neo4j$ users_movies_df.to_csv ('users_movies.csv', sep='|', header=True,index=False)
```

```
neo4j$ movies_genres.to_csv('movies_genres.csv', sep='|', header=True,index=False)
```

```
users_genres.to_csv ('users_genres.csv', sep='|', header=True,
index=False)
```

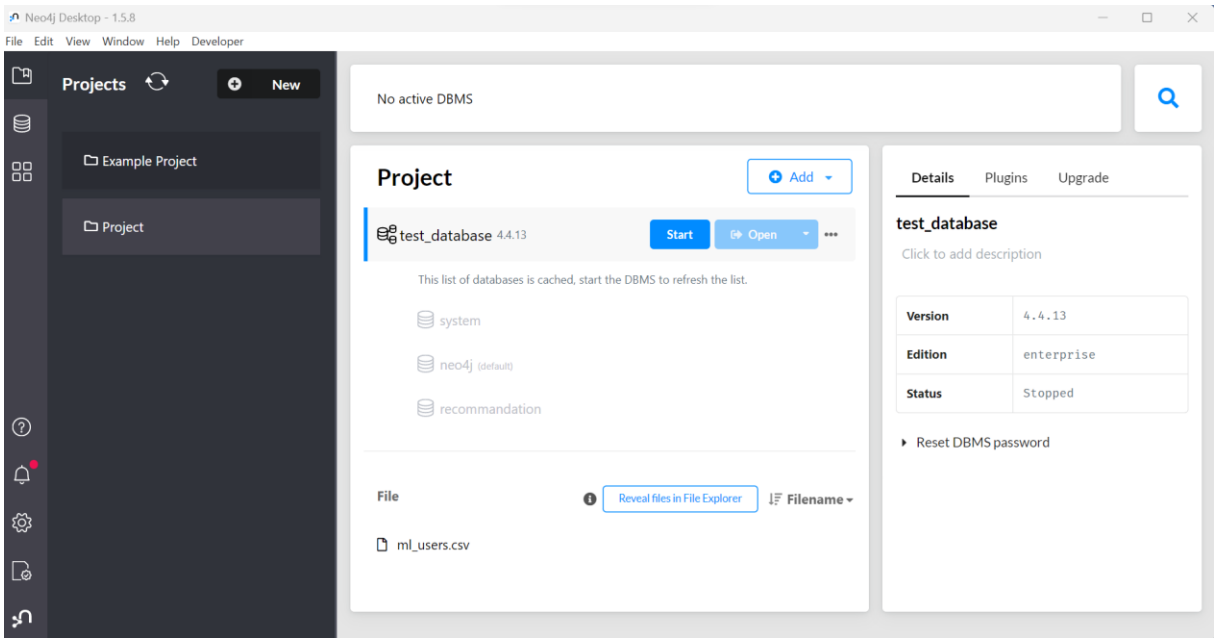
```
movies_similarity.to_csv ('movies_similarity.csv', sep='|',
header=True, index=False)
```

2.1 Configuration de Neo4j

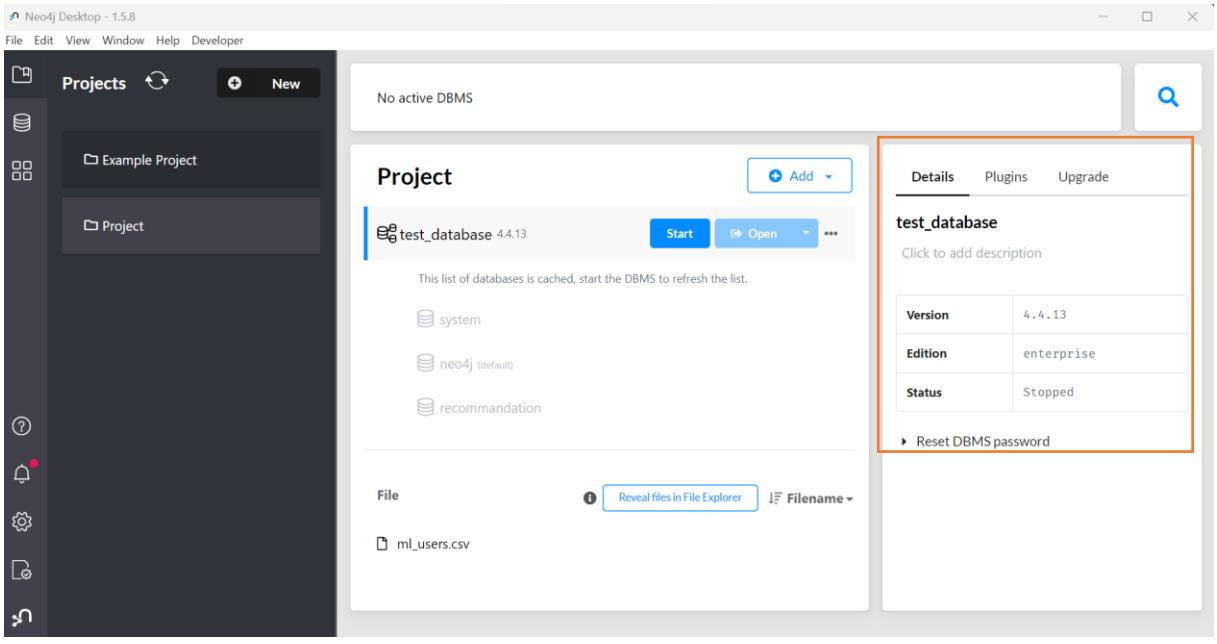
Pour ce projet on a utilisé la version Desktop de Neo4j. Cette version se distingue par les propriétés suivantes :

- **Interface graphique conviviale** : La version desktop de Neo4j est livrée avec une interface graphique conviviale appelée Neo4j Desktop. Cette interface permet de gérer facilement plusieurs bases de données Neo4j, de créer et de gérer des projets, et de configurer les paramètres du serveur.
- **Installation simplifiée** : Neo4j Desktop facilite l'installation de Neo4j sur votre machine. Il vous suffit de télécharger et d'installer Neo4j Desktop, et vous pouvez ensuite créer et gérer plusieurs instances de bases de données Neo4j à partir de l'interface utilisateur.
- **Gestion des bases de données** : Avec la version desktop, vous pouvez facilement créer, démarrer, arrêter et supprimer des bases de données Neo4j. L'interface permet également de gérer les versions de Neo4j et de configurer les paramètres du serveur.

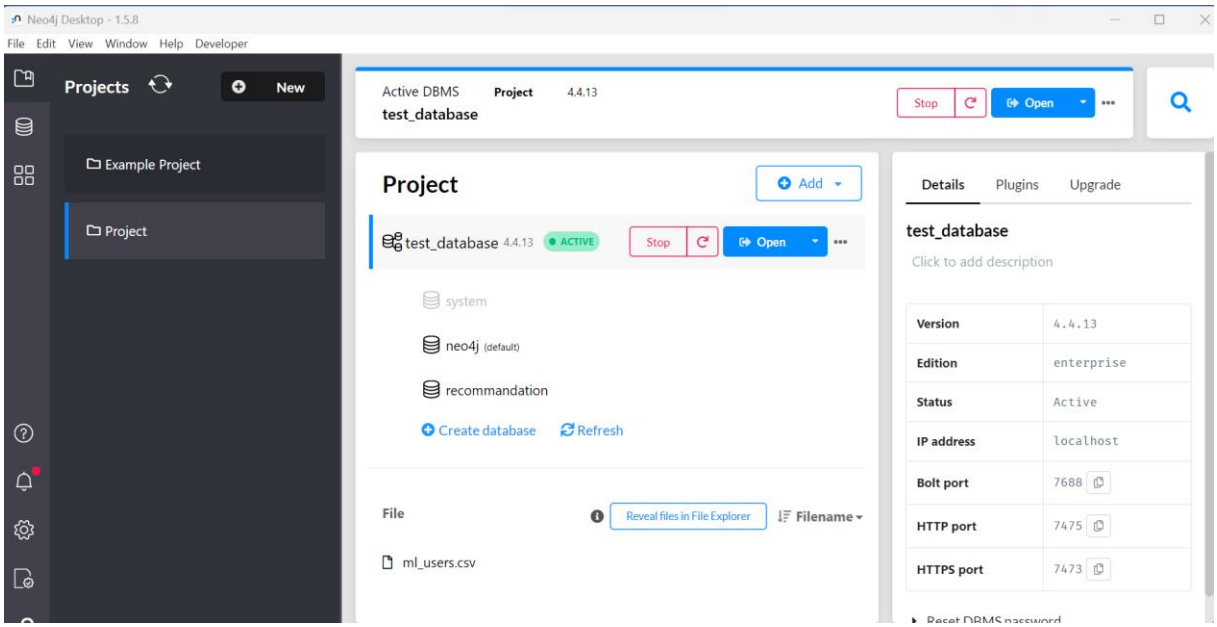
1) Dans le projet qu'on a intitulé « projet », on a créé notre graphe « test_database »



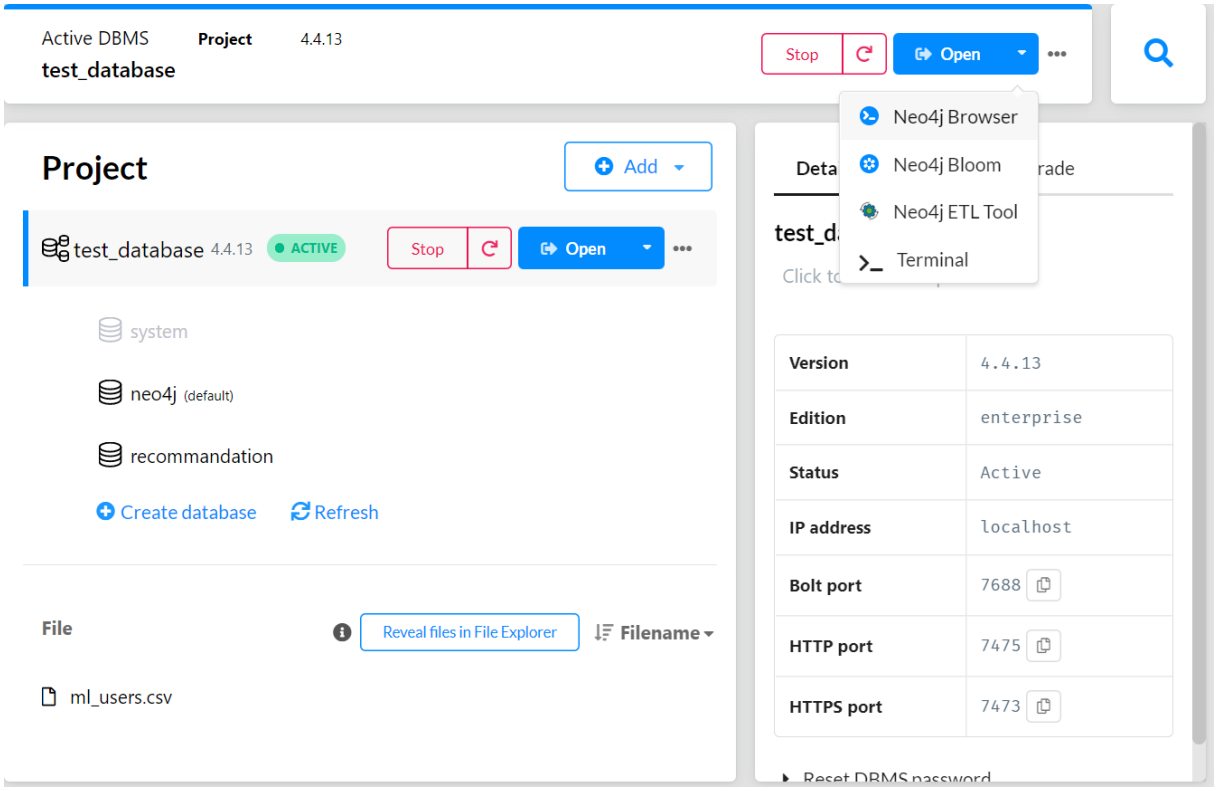
2) Dans les détails de notre base de données à droite on peut apercevoir la version de Neo4j qu'on a utilisé et le status de la base de donnée qui actuellement en arrêt.

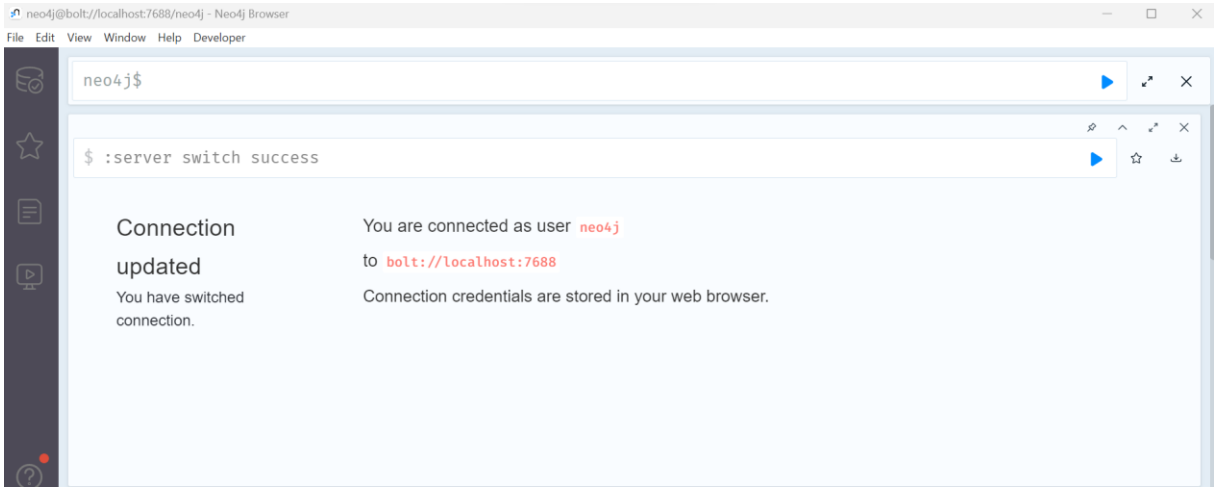


3) On démarre notre base de données et on percoit d'autres détails qui s'affichent comme le port http et le Bolt port utilisés par Neo4j pour les communications entre les clients et le serveur de base de données.



4) On ouvre ensuite notre base de données dans le browser



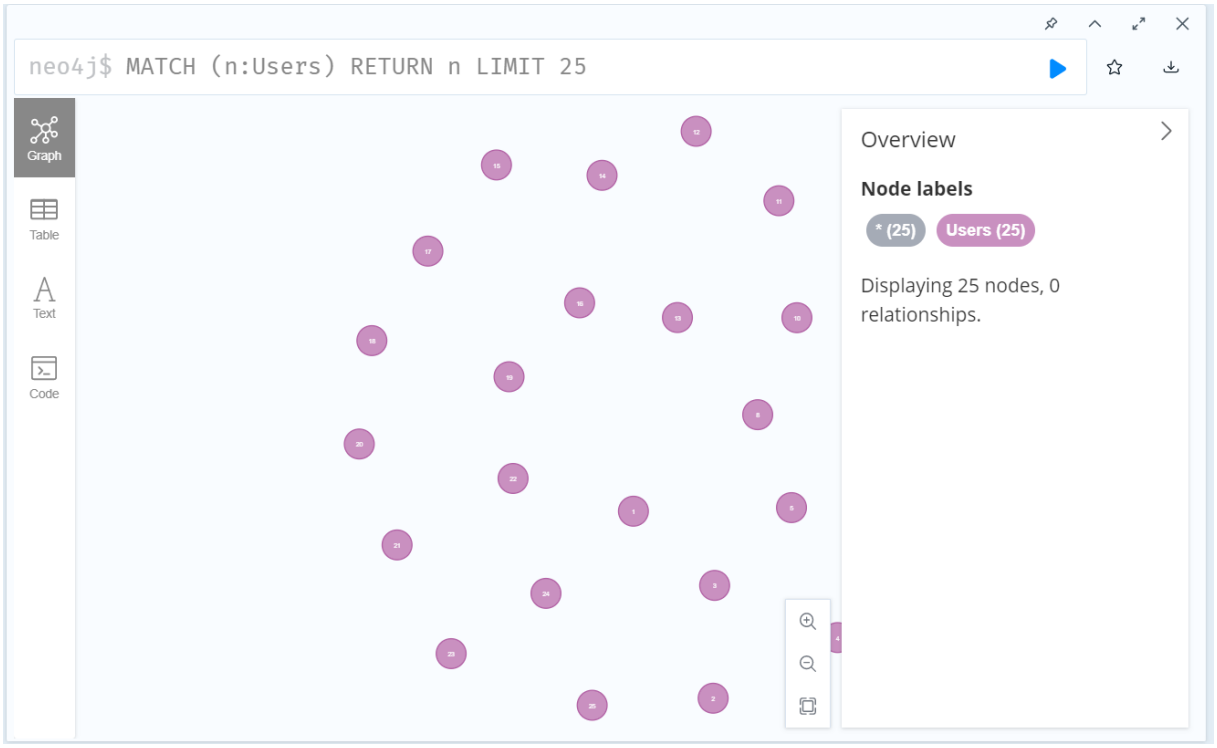


Maintenant que notre environnement est prêt. On enchaine une suite d'opérations qui vont nous permettre de créer notre graphe.

2.2 Chargement des nœuds

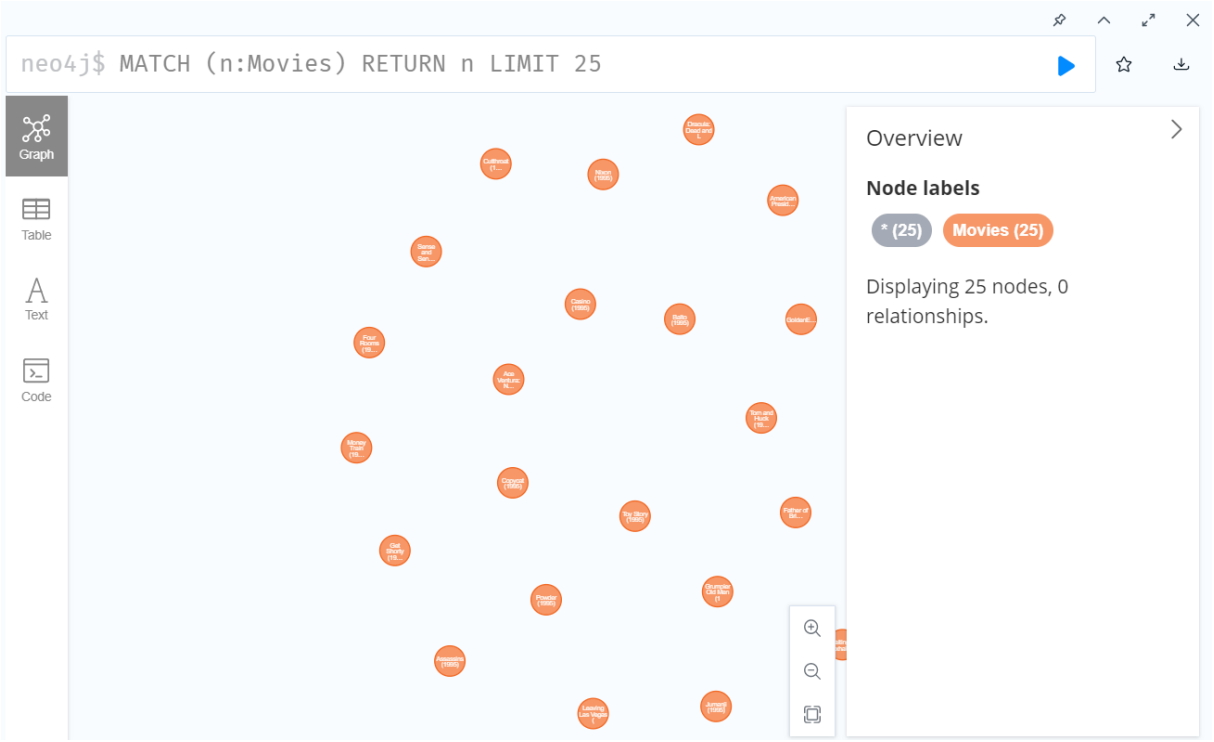
5) Tous d'abord on va charger les nœuds de notre graphe, et on va commencer avec ceux des utilisateurs

```
1 USING PERIODIC COMMIT
2 LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row
3 FIELDTERMINATOR '|'
4 CREATE (:Users{userId:
5 sponible
6 row.userId})
```



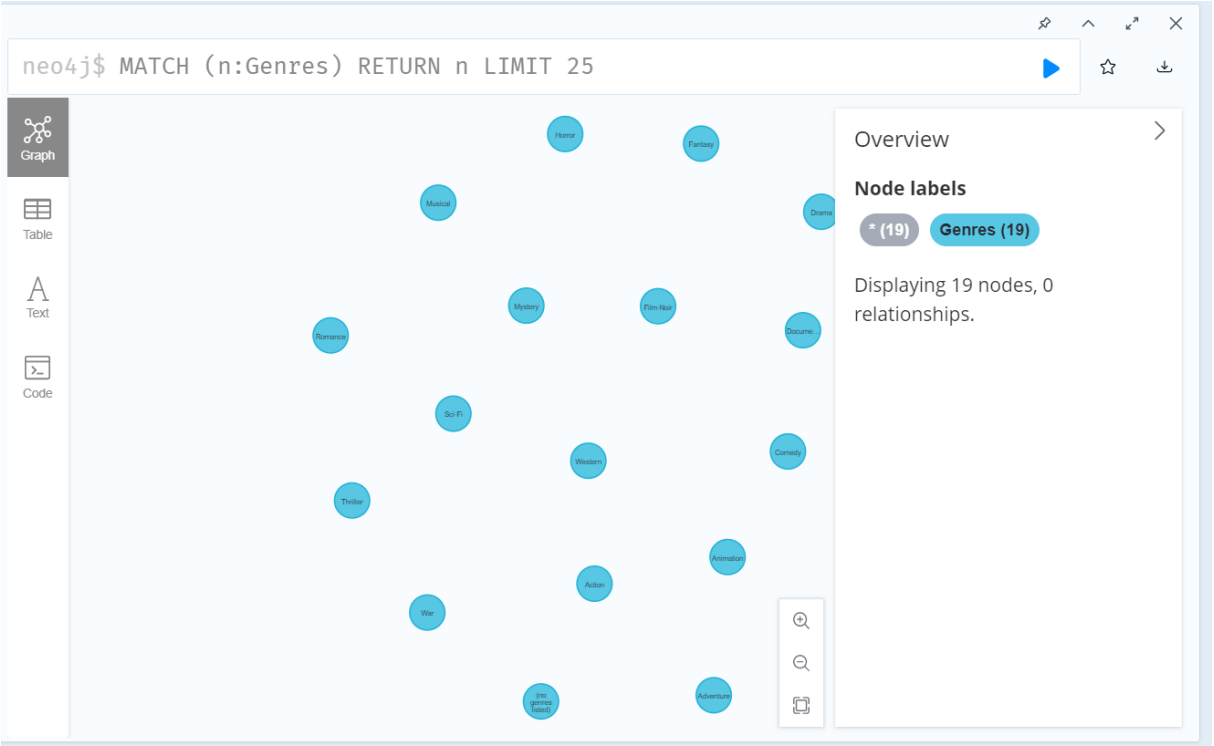
6) On importe ensuite les nœuds correspondants aux films

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///movies.csv" AS row
OFIELDTERMINATOR
'|'
CREATE (:Movies {movieId: row.movieId, title: row.title, rating_mean:
row.rating_mean});
```



7) On importe ensuite les différents genres des films

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///genres.csv" AS row
FIELDTERMINATOR'|'
CREATE (:Genres {genres: row.genres});
```



2.3 Création d'index

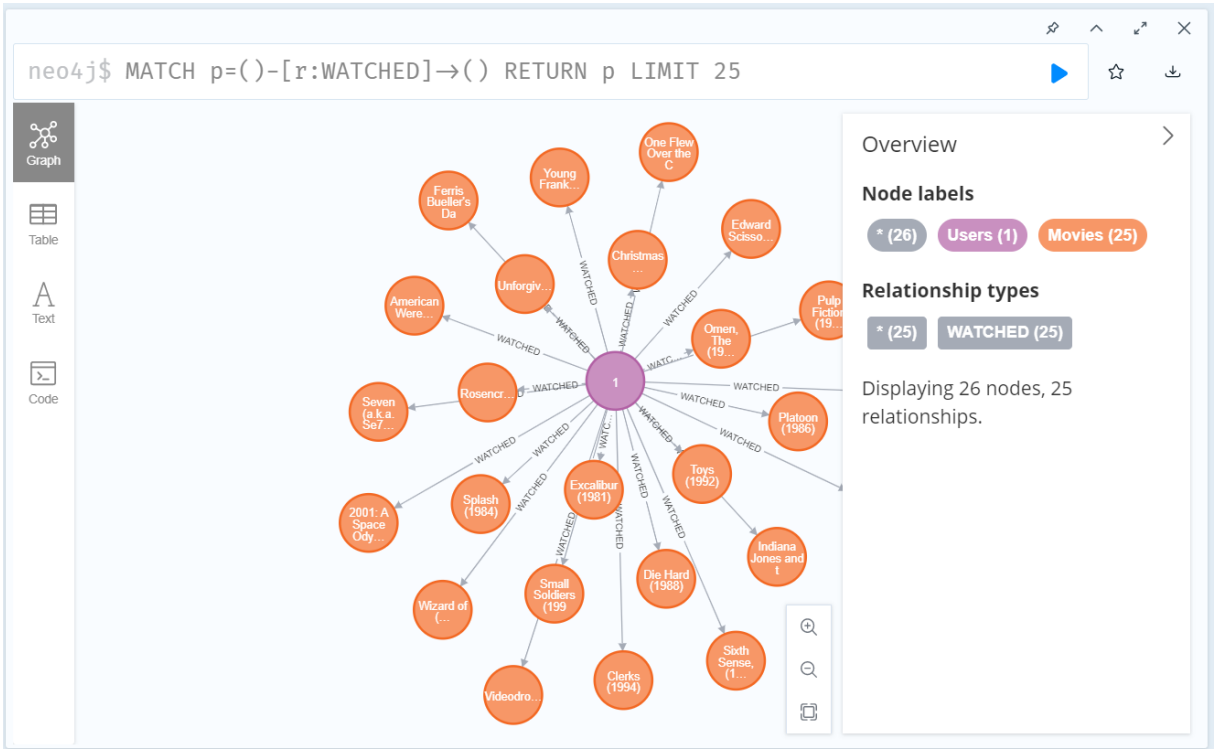
8) Avant de charger les relations entre les nœuds, on va d'abord créer des index sur les propriétés UserId et movieId afin d'améliorer les performances des requêtes de recherche en accélérant la recherche et la récupération des nœuds basée sur ces propriétés

```
1 CREATE INDEX ON :Users (userId);
2 CREATE INDEX ON :Movies (movieId);
```

2.4 Chargement des relations

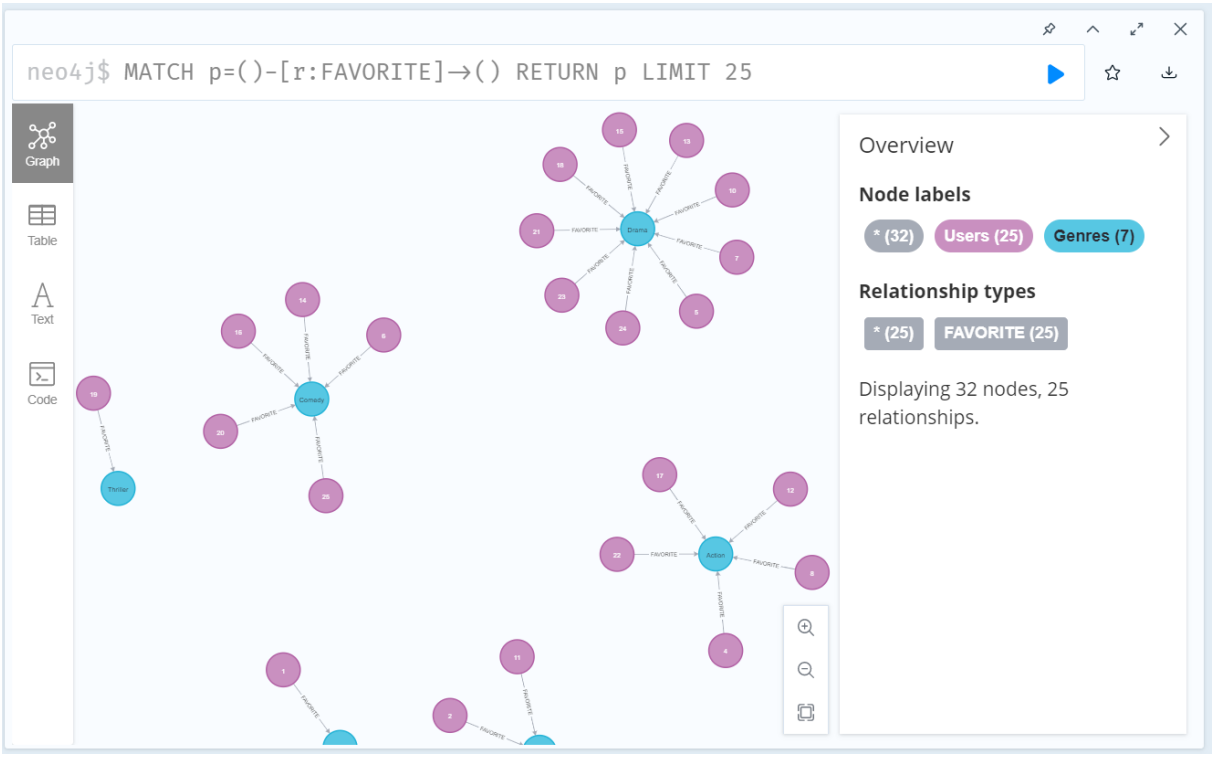
9) La première relation est la relation « watched » qui relie les utilisateurs aux films qu'ils ont vu précédemment

```
1 USING PERIODIC COMMIT
2 LOAD CSV WITH HEADERS FROM
3 FIELDTERMINATOR '|'
4 "file:///users_movies.csv" AS row
5 MATCH (user: Users {userId: row.userId})
6 MATCH (movie: Movies {movieId: row.movieId})
7 MERGE (user)-[:WATCHED {rating: row.rating}]→ (movie);
```



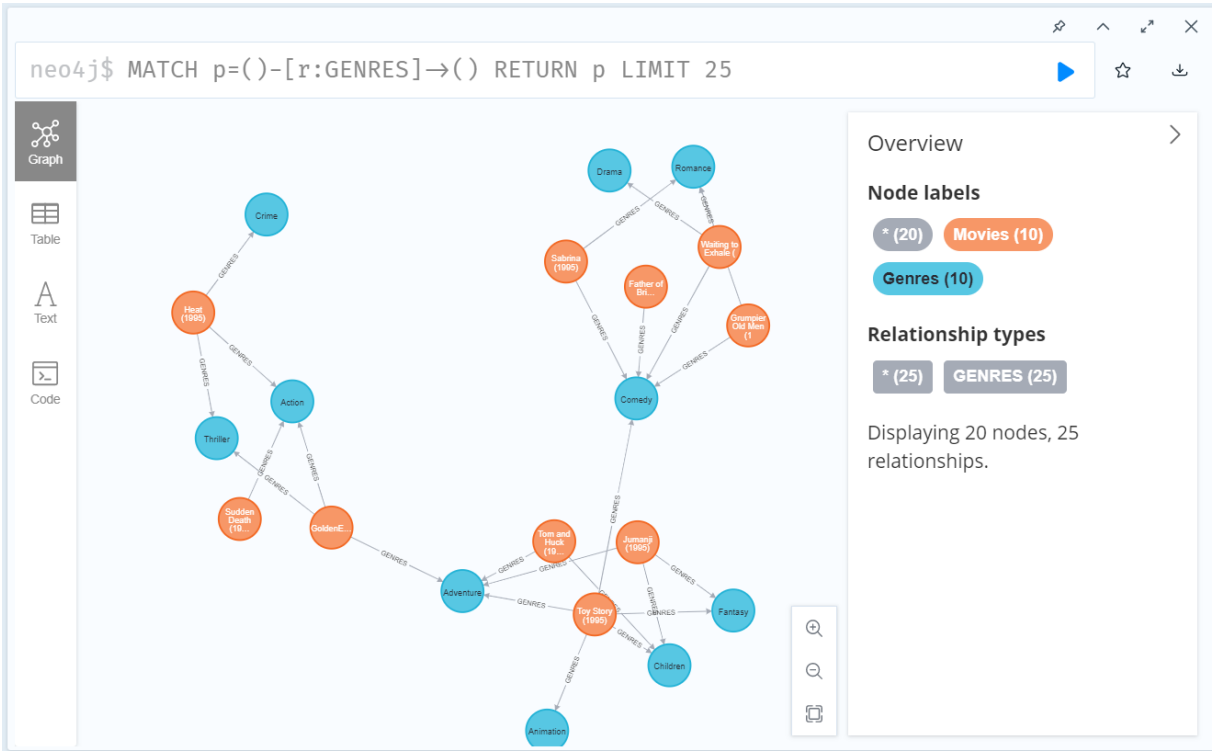
10) La deuxième relation chargée est la relation « favorite » qui relie les utilisateurs aux genres

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
FIELDTERMINATOR '|'
"file:///users_genres.csv" AS row
MATCH (user: Users {userId: row.userId})
MATCH (genres: Genres {genres: row.genre})
MERGE (user)-[:FAVORITE]→ (genres);
```



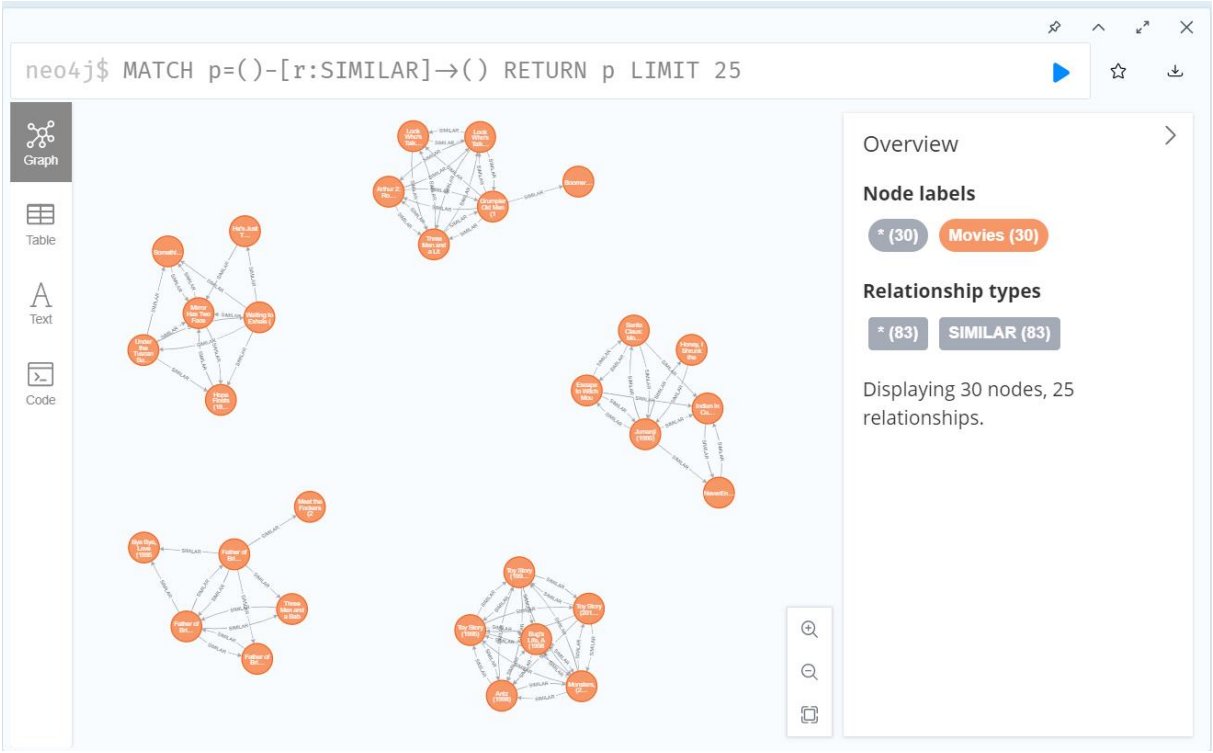
11) La relation chargée ensuite est « genre », c'est celle reliant les genres aux films

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
FIELDTERMINATOR '|'
"file:///movies_genres.csv" AS row
MATCH (movie: Movies {movieId: row.movieId})
MATCH (genres: Genres {genres: row.genres})
MERGE (movie)-[:GENRES]→ (genres);
```



12) On charge ensuite la relation de similarité entre les films

```
1 USING PERIODIC COMMIT
2 LOAD CSV WITH HEADERS FROM "file:///movies_similarity.csv" AS row
3 FIELDTERMINATOR '|'
4 MATCH (moviel: Movies {movieId: row.movieId})
5 MATCH (movie2: Movies {movieId: row.sim_moveId})
6 MERGE (movie1)-[:SIMILAR [relevance: row. relevance}]→ (movie2);
```



Maintenant notre graphe est prêt pour les recommandations

2.5 Recommandation des films

Maintenant que notre base de données est prête, on va interagir avec cette dernière à travers des requetés dans le langage Cypher

- **Extraire les films regardés par un utilisateur spécifique**

```
MATCH path = (u:Users)-[:WATCHED]→(m1:Movies)
WHERE u.userId =~'4'
RETURN u.userId, m1.title, m1.rating_mean
```

```
MATCH path = (u:Users)-[:WATCHED]→(m1:Movies)
WHERE u.userId =~'4'
RETURN u.userId, m1.title, m1.rating_mean
```

Table | Text | Code

	u.userId	m1.title	m1.rating_mean
1	"4"	"Die Hard: With a Vengeance (1995)"	"3.4890247495580438"
2	"4"	"RoboCop 3 (1993)"	"2.19690357439734"
3	"4"	"Forrest Gump (1994)"	"4.029000181345584"
4	"4"	"Terminator 2: Judgment Day (1991)"	"3.9319539085828037"
5	"4"	"Rock, The (1996)"	"3.679536248524862"
6	"4"	"Naked Gun 33 1/3: The Final Insult (1994)"	"2.9555080288899798"

- **Extraire le genre favori d'un utilisateur**

```
MATCH path = (u:Users)-[:FAVORITE]→(g:Genres)
WHERE u.userId =~'4'
RETURN u.userId, g.genres
```

Table | Text | Code

	u.userId	g.genres
1	"4"	"Action"

Started streaming 1 records after 13 ms and completed after 217 ms.

- **Requête pour extraire les films similaires aux films que l'utilisateur à déjà regardé**. pour chaque film, 5 films similaires seront recommandés. Mais vu qu'on n'est intéressés que par les films de genre favori de l'utilisateur, on va filtrer selon cette condition

```
MATCH path = (u:Users)-[:WATCHED]→(m1:Movies)-[s:SIMILAR]→
(m2:Movies),
(m2)-[:GENRES]→(g:Genres),
(u)-[:FAVORITE]→(g)
WHERE u.userId =~'4'
RETURN u.userId, g.genres, m1.title, m2.title, m2.rating_mean
```

Table | Text | Code

	u.userId	g.genres	m1.title	m2.title
2	"4"	"Action"	"Die Hard: With a Vengeance (1995)"	"U.S. Marshals (1998)"
3	"4"	"Action"	"Die Hard: With a Vengeance (1995)"	"Point Break (1991)"
4	"4"	"Action"	"Die Hard: With a Vengeance (1995)"	"Die Hard (1988)"
5	"4"	"Action"	"Die Hard: With a Vengeance (1995)"	"F/X (1986)"
6				

- **Cette recommande précédente nous a renvoyé 67 films, on veut limiter nos recommandations à 5 films seulement.**

```
MATCH (u1:Users)-[:WATCHED]→(m3:Movies)
WHERE u1.userId =~'4'
WITH [i in m3.movieId | i] as movies
MATCH path = (u:Users)-[:WATCHED]→(m1:Movies)-[s:SIMILAR]→
(m2:Movies),
(m2)-[:GENRES]→(g:Genres),
(u)-[:FAVORITE]→(g)
WHERE u.userId =~'4' and not m2.movieId in movies
RETURN distinct u.userId as userId, g.genres as genres,
m2.title as title, m2.rating_mean as rating
ORDER BY m2.rating_mean descending
LIMIT 5
```

Table | Text | Code

	userId	genres	title	rating
1	"4"	"Action"	"French Connection, The (1971)"	"3.9614397386535996"
2	"4"	"Action"	"Die Hard (1988)"	"3.9338465081088194"
3	"4"	"Action"	"Terminator, The (1984)"	"3.8963923539441803"
4	"4"	"Action"	"Edge of Tomorrow (2014)"	"3.8436374549819927"

Started streaming 5 records after 2 ms and completed after 4261 ms.

Notre moteur de recommandation fonctionne bien maintenant.

3 Interface graphique

3.1 Technologies utilisées

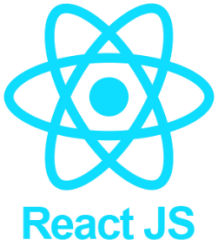
Pour développer notre interface graphique, on a utilisé les technologies suivantes :

Flask

Flask est un framework web léger et flexible écrit en Python. Il permet de construire rapidement des applications web en utilisant Python pour gérer les requêtes HTTP, la logique métier et l'interaction avec la base de données Neo4j. Flask offre également des fonctionnalités pour le routage, la gestion des sessions et le rendu de templates, entre autres.

ReactJs

React, quant à lui, est une bibliothèque JavaScript populaire pour la construction d'interfaces utilisateur interactives. Il offre une approche déclarative pour créer des composants réutilisables et gérer efficacement l'état de l'application. Avec React, nous pouvons créer une interface utilisateur réactive et dynamique pour interagir avec le moteur de recherche et afficher les résultats de manière conviviale.

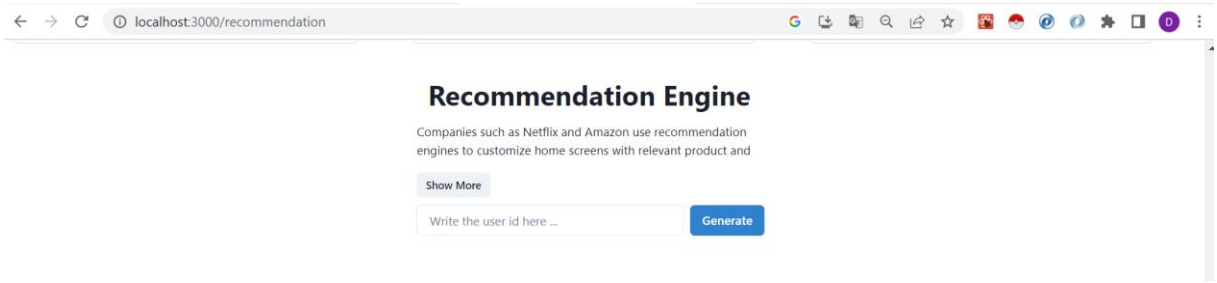


3.2 La combinaison des deux outils Flask et ReactJs

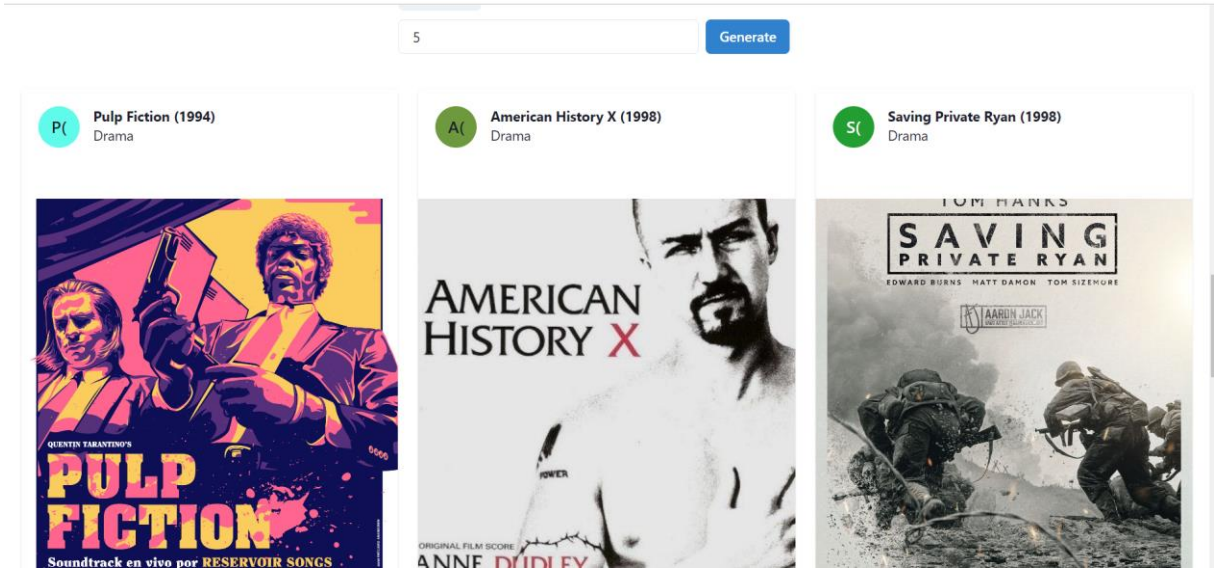
- En utilisant Flask en tant que backend, nous puvons exposer des API pour interagir avec notre base de données Neo4j et traiter les requêtes de recherche. Flask gère les requêtes HTTP, la logique de l'application et peut intégrer des bibliothèques Python spécifiques à Neo4j pour exécuter les requêtes Cypher et récupérer les résultats.
- De l'autre côté, React se chargera de la partie frontend de notre application, offrant une interface utilisateur réactive qui communique avec le backend via des requêtes HTTP. nous pouvez utiliser des bibliothèques telles que Axios pour effectuer les appels API vers notre backend Flask et afficher les résultats de recherche de manière interactive dans votre interface React.

3.3 Interface de recommandation

Voici l'interface de recommandation des films qui sera affiché à l'utilisateur souhaitant retrouver des recommandations.



- Recherche pour un utilisateur spécifique



Conclusion

La conception d'un système de recommandation de films utilisant une base de données graphes offre de nombreux avantages. En utilisant un modèle de graphe, nous pouvons représenter les films, les utilisateurs et les relations entre eux de manière intuitive et flexible. Ce système permet de capturer les relations complexes entre les films, tels que les genres, les acteurs, les réalisateurs, les évaluations des utilisateurs, etc. En modélisant ces informations sous forme de nœuds et de relations dans le graphe, nous pouvons effectuer des requêtes avancées pour recommander des films pertinents aux utilisateurs.

De plus, les bases de données graphes offrent des performances élevées pour les requêtes complexes impliquant des relations entre les données. Les graphes sont particulièrement efficaces pour les opérations de recherche et de navigation, ce qui est essentiel dans un système de recommandation où nous voulons trouver rapidement des films pertinents pour les utilisateurs.

Finalement La conception d'un système de recommandation de films basé sur une base de données graphes offre une approche puissante et flexible pour offrir des recommandations personnalisées aux utilisateurs. Ce système améliore l'expérience utilisateur en proposant des films adaptés à leurs préférences individuelles et en favorisant la découverte de nouveaux films intéressants. Grâce à l'utilisation de graphes, les relations complexes entre les films, les utilisateurs et les caractéristiques des films peuvent être représentées de manière efficace. Cela permet d'effectuer des requêtes avancées pour identifier les films similaires en fonction des préférences des utilisateurs et des caractéristiques des films. De plus, les bases de données graphes offrent des performances élevées pour les requêtes complexes, ce qui garantit une réponse rapide et efficace lors de la recherche de recommandations. En résumé, un système de recommandation de films basé sur une base de données graphes permet d'offrir des recommandations personnalisées, d'améliorer l'expérience utilisateur et de faciliter la découverte de nouveaux films, tout en garantissant des performances élevées pour les requêtes complexes