



INF4064 – NoSQL

Projet

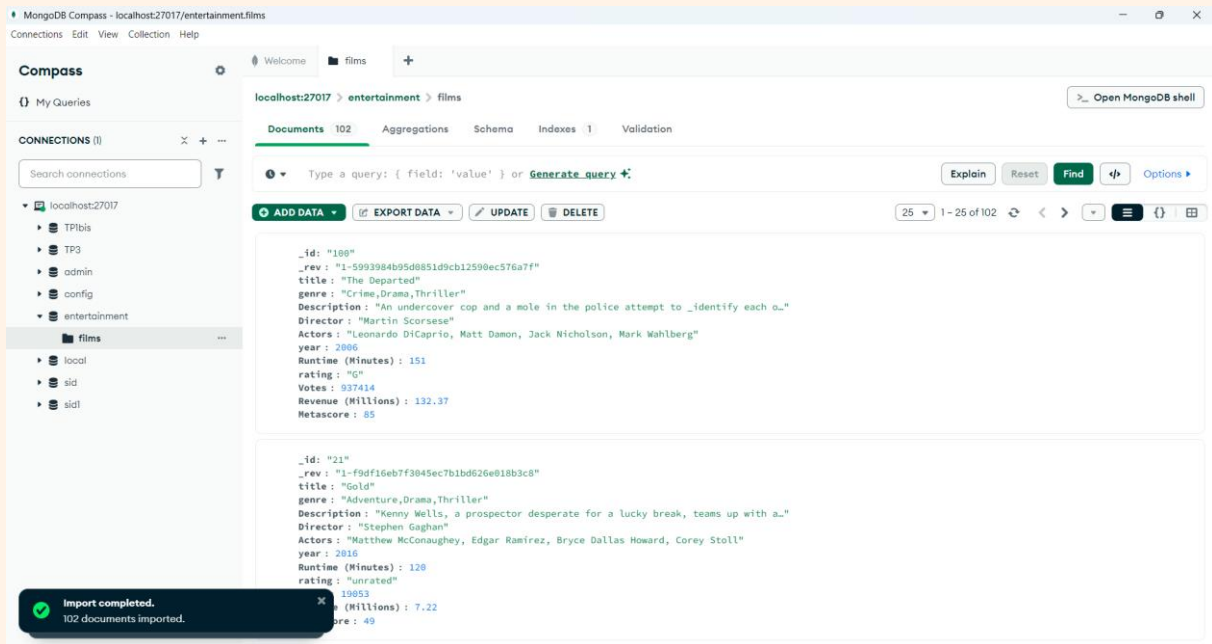


Table des matières

I – Bases de données.....	3
1. Base de données MongoDB	3
2. Base de données Neo4J	11
3. Questions transversales.....	22
II – Application Python.....	26
1. Introduction	26
2. Objectifs du projet.....	26
3. Configuration et mise en place.....	26
3.1 Environnement virtuel	26
3.2 Installation des dépendances.....	26
3.3 Configuration des bases de données	27
4. Structure de l'application.....	27
5. Difficultés rencontrées	27
5.1 Gestion de l'affichage des relations	27
6. Conclusion.....	27
7. Annexes	28
7.1 Code source.....	28
7.2 Références	28

I – Bases de données

1. Base de données MongoDB



1.

```
> db.films.aggregate([{$group: {_id: "$year", total_films: {$sum: 1}}}, {$sort: {total_films: -1}}, {$limit: 1}])
< {
  _id: 2016,
  total_films: 73
}
```

- \$group : permet de regrouper les films par année et compte le nombre de films par année
- \$sort : permet de trier selon le nombre de films, par ordre décroissant.
- \$limit : permet d'afficher seulement le premier résultat (l'année avec le plus de films).

2.

```
> db.films.aggregate([{$match: {"year": {$gt: 1999}}}, {$count: "total_films_after_1999"}])
< {
  total_films_after_1999: 99
}
```

- \$match : permet de poser une condition (ici, l'année doit être supérieure à 1999).
- \$count : permet de compter les documents respectant la condition de \$match.

3.

```
> db.films.aggregate([{$match: {"year": 2007}}, {$group: {_id: null, avg_votes: {$avg: "$Votes"}}}])
< {
  _id: null,
  avg_votes: 192.5
}
```

- \$match : permet de sélectionner les films sortis en 2007.
- \$group : permet de calculer la moyenne des votes.

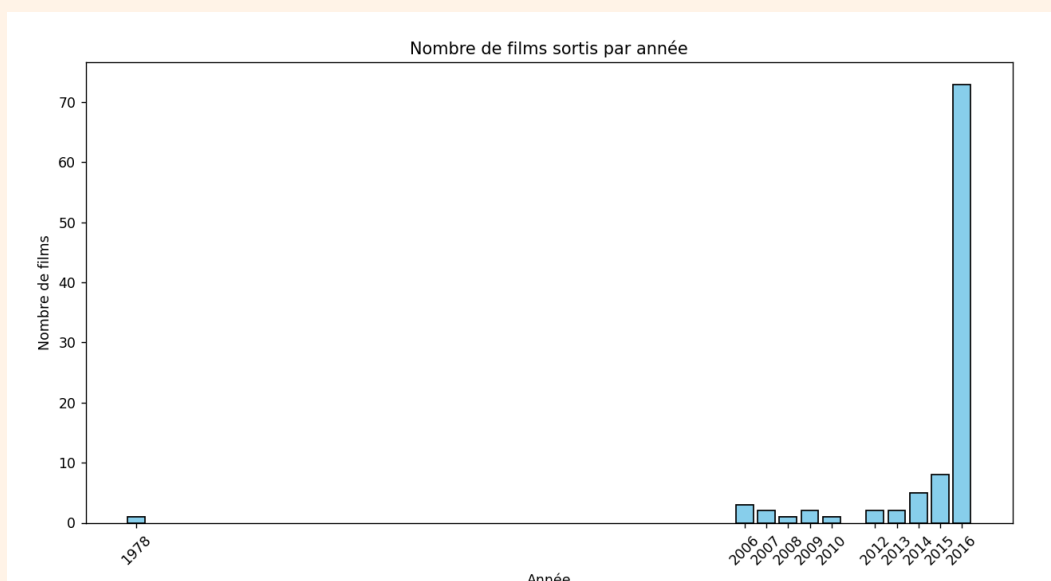
4.

```
> db.films.aggregate([{$group: {_id: "$year", count: {$sum: 1}}}, {$sort: {_id: 1}}])
```

- \$group : permet de compter le nombre de films par année.
- \$sort : permet de trier les résultats selon les années, par ordre croissant.

```
EXPLORER  ...  mongo_quest4.py X
OPEN EDITORS
X mongo_quest4.py
PROJET NOSQL
> .venv
4.png
mongo_quest4.py
requirement.txt

1 import pymongo
2 import matplotlib.pyplot as plt
3
4 # Connexion à la base MongoDB
5 client = pymongo.MongoClient("mongodb://localhost:27017/")
6 db = client["entertainment"]
7 collection = db["films"]
8
9 # Exécuter l'agrégation
10 pipeline = [
11     {"$group": {"_id": "$year", "count": {"$sum": 1}}},
12     {"$sort": {"_id": 1}}
13 ]
14 data = list(collection.aggregate(pipeline))
15
16 # Extraire les données pour le graphique
17 years = [int(entry["_id"]) for entry in data if entry["_id"] is not None] # car une entrée None
18 counts = [entry["count"] for entry in data if entry["_id"] is not None]
19
20 # Afficher l'histogramme
21 plt.figure(figsize=(12,6))
22 plt.bar(years, counts, color='skyblue', edgecolor='black')
23 plt.xlabel("Année")
24 plt.ylabel("Nombre de films")
25 plt.title("Nombre de films sortis par année")
26 plt.xticks(years, rotation=45)
27 plt.show()
```



5.

```
> db.films.aggregate([{"$project": {"genre": {"$split": ["$genre", ","]}}, {"$unwind": "$genre"}, {"$group: {_id: "$genre"}}])
< {
  _id: 'Romance'
}
{
  _id: 'Western'
}
{
  _id: 'Family'
}
{
  _id: 'Animation'
```

- `$project` : permet de diviser la chaîne de caractères pour les genres afin de les séparer et d'en obtenir une liste.
- `$unwind` : permet de décomposer la liste des caractères précédemment obtenus.
- `$group` : permet ici de regrouper les genres pour éviter les répétitions (genres distincts).

6.

```
> db.films.aggregate([{"$match": {"Revenue (Millions)": {"$ne": "" } }}, {"$sort": {"Revenue (Millions)": -1}}, {"$limit": 1}])
< {
  _id: '51',
  _rev: '1-efdfbecc4f0bb1b6424fdea48ce766f9',
  title: 'Star Wars: Episode VII - The Force Awakens',
  genre: 'Action,Adventure,Fantasy',
  Description: 'Three decades after the defeat of the Galactic Empire, a new threat arises. The First Order attempts to rule the galaxy',
  Director: 'J.J. Abrams',
  Actors: 'Daisy Ridley, John Boyega, Oscar Isaac, Domhnall Gleeson',
  year: 2015,
  'Runtime (Minutes)': 136,
  rating: 'G',
  Votes: 661608,
  'Revenue (Millions)': 936.63,
  Metascore: 81
}
```

- `$match` : permet de ne pas prendre en compte les films pour lesquels le revenu est inconnu.
- `$sort` : permet de trier les films selon le revenu, par ordre décroissant.
- `$limit` : permet d'afficher seulement le premier résultat (le film avec le meilleur revenu).

7.

```
> db.films.aggregate([{"$group": {_id: "$Director", number_of_films: {"$sum": 1}}}, {"$match": {"number_of_films": {"$gt": 5}}})
<
```

- `$group` : permet de regrouper les films par réalisateur, en les comptant.
- `$match` : permet d'afficher seulement les réalisateurs ayant réalisé plus de 5 films.

Comme aucun réalisateur de la base de données n'a réalisé plus de 5 films, nous avons testé jusqu'à obtenir le résultat ci-dessous :

```
> db.films.aggregate([{$group: {_id: "$Director", number_of_films: {$sum: 1}}}, {$match: {number_of_films: {$gt: 2}}})
< {
  _id: 'Martin Scorsese',
  number_of_films: 3
}
{
  _id: 'Christopher Nolan',
  number_of_films: 4
}
```

C'est donc Christopher Nolan qui a réalisé le plus de films dans notre base de données, au nombre de 4 films.

8.

```
> db.films.aggregate([{$match: {"Revenue (Millions)": {$ne: ""}}},
{$project: {"genre": {$split: ["$genre", ","]}, revenue: {$toDouble: "$Revenue (Millions)"}},
{$unwind: "$genre"},
{$group: {_id: "$genre", avg_revenue: {$avg: "$revenue"}}},
{$sort: {avg_revenue: -1}}, {$limit: 1}])
< {
  _id: 'Fantasy',
  avg_revenue: 265.8335714285714
}
```

- `$match` : permet de ne pas prendre en compte les films pour lesquels le revenu est inconnu.
- `$project` : permet de diviser la chaîne de caractères pour les genres afin de les séparer et d'en obtenir une liste ; permet de convertir le revenu en valeur numérique « double ».
- `$unwind` : permet de décomposer la liste des caractères précédemment obtenus.
- `$group` : permet de regrouper les films par genre, en associant la moyenne des revenus.
- `$sort` : permet de trier les films selon le revenu, par ordre décroissant.
- `$limit` : permet d'afficher seulement le premier résultat (le genre de film avec le meilleur revenu).

9.

```
> db.films.aggregate([{$match: {"rating": {$ne: ""}}},
{$addFields: {decade: {$subtract: [{$toInt: "$year"}, {$mod: [{$toInt: "$year"}, 10]}}}},
{$sort: {"decade": 1, "rating": -1}},
{$group: { _id: "$decade", top_movies: {$push: {title: "$title", rating: "$rating"}}}},
{$project: { _id: 1, top_movies: {$slice: ["$top_movies", 3]}}},
{$sort: { _id: 1}}])
```

- `$match` : permet de ne pas prendre en compte les films pour lesquels le *rating* est inconnu.
- `$addFields` : permet de calculer la décennie de sortie de chaque film (on soustrait le reste de la division euclidienne par 10).

- `$sort` : permet de trier les films selon la décennie, par ordre croissant, et selon le *rating*, par ordre décroissant (les meilleurs films de chaque décennie seront affichés en premier).
- `$group` : permet de regrouper les films par décennie, en créant une liste des titres et *ratings* des films.
- `$project` : permet de ne garder que les 3 meilleurs films de la décennie (pas de `$limit : 3` car `top_movies` est un tableau).
- `$sort` : permet de garantir le tri les décennies, par ordre croissant

Comme le *rating* est soit « G », soit « unrated » (ce qui n'est pas très parlant selon nous), nous avons plutôt décidé d'afficher les 3 films de chaque décennie avec le meilleur *Metascore* :

```
> db.films.aggregate([{$match: {"Metascore": {$ne: ""}}},
  {$addFields: {decade: {$subtract: [{$toInt: "$year"}, {$mod: [{$toInt: "$year"}, 10]}}}},
  {$addFields: {Metascore: {$toInt: "$Metascore"}}},
  {$sort: {"decade": 1, "Metascore": -1}},
  {$group: {_id: "$decade", top_movies: {$push: {title: "$title", Metascore: "$Metascore"}}}},
  {$project: { _id: 1, top_movies: {$slice: ["$top_movies", 3]}}},
  {$sort: { _id: 1}}])
< {
  _id: 1970,
  top_movies: [
    {
      title: 'Top Dog',
      Metascore: 67
    }
  ]
}
{
  _id: 2000,
  top_movies: [
    {
      title: 'The Departed',
      Metascore: 85
    },
    {
      title: 'Avatar',
      Metascore: 83
    },
    {
      title: 'The Dark Knight',
      Metascore: 82
    }
  ]
}
```

```
{
  _id: 2010,
  top_movies: [
    {
      title: 'Moonlight',
      Metascore: 99
    },
    {
      title: 'Manchester by the Sea',
      Metascore: 96
    },
    {
      title: 'La La Land',
      Metascore: 93
    }
  ]
}
```

10.

```
> db.films.aggregate([{$match: {"Runtime (Minutes)": {$ne: ""}}},
  {$project: {"genre": {$split: ["$genre", ","]}, "title": 1, "runtime": {$toInt: "$Runtime (Minutes)"}},
  {$unwind: "$genre"},
  {$sort: {"runtime": -1}},
  {$group: {_id: "$genre", longest_film: {$first: {title: "$title", Runtime: "$runtime"}}}}])
< {
  _id: 'Sci-Fi',
  longest_film: {
    title: 'Interstellar',
    Runtime: 169
  }
}
{
  _id: 'Western',
  longest_film: {
    title: 'The Magnificent Seven',
    Runtime: 132
  }
}
{
  _id: 'Mystery',
  longest_film: {
    title: 'The Hateful Eight',

```

- \$match : permet de ne pas prendre en compte les films pour lesquels la durée en minutes est inconnue.
- \$project : permet de diviser la chaîne de caractères pour les genres afin de les séparer et d'en obtenir une liste ; permet de convertir la durée en valeur numérique « integer ».
- \$unwind : permet de décomposer la liste des caractères précédemment obtenus.
- \$sort : permet de trier les films selon leur durée, par ordre décroissant.
- \$group : permet de regrouper les films par genre, en associant la durée en minutes et en ne retournant que la première valeur par genre.

11.


```

> db.createView("top_movies", "films",
  [{ $match: { "Metascore": { $ne: "", $gt: 80 }, "Revenue (Millions)": { $ne: "", $gt: 50 } } },
   { $project: { "title": 1, "year": 1, Metascore: { $toInt: "$Metascore" }, revenue: { $toDouble: "$Revenue (Millions)" } } } ] )
< { ok: 1 }
> db.top_movies.find()
< {
  _id: '100',
  title: 'The Departed',
  year: 2006,
  Metascore: 85,
  revenue: 132.37
}
{
  _id: '51',
  title: 'Star Wars: Episode VII - The Force Awakens',
  year: 2015,
  Metascore: 81,
  revenue: 936.63
}

```

- `$match` : permet de ne pas prendre en compte les films pour lesquels le *Metascore* est inconnu et de ne garder que ceux dont la valeur est supérieure à 80 ; permet de ne pas prendre en compte les films pour lesquels le revenu est inconnu et de ne garder que ceux dont la valeur est supérieure à 50.
- `$project` : permet de n'afficher que le titre, l'année, le *Metascore* et le revenu.

12.

```

> db.films.aggregate([ { $match: { "Runtime (Minutes)": { $ne: "" }, "Revenue (Millions)": { $ne: "" } } },
  { $project: { "title": 1, runtime: { $toInt: "$Runtime (Minutes)" }, revenue: { $toDouble: "$Revenue (Millions)" } } } ] )
< {
  _id: '100',
  title: 'The Departed',
  runtime: 151,
  revenue: 132.37
}
{
  _id: '21',
  title: 'Gold',
  runtime: 120,
  revenue: 7.22
}
{
  _id: '31',
  title: 'Why Him?',
  runtime: 111,
  revenue: 60.31
}

```

- `$match` : permet de ne pas prendre en compte les films pour lesquels la durée est inconnue ; permet de ne pas prendre en compte les films pour lesquels le revenu est inconnu.
- `$project` : permet de n'afficher que le titre, la durée et le revenu.

```

EXPLORER
...
mongo_quest12.py X
mongo_quest12.py
1 import pymongo
2 import pandas as pd
3 from scipy.stats import pearsonr
4
5 # Connexion à MongoDB
6 client = pymongo.MongoClient("mongodb://localhost:27017/")
7 db = client["entertainment"]
8 collection = db["films"]
9
10 # Récupération des données
11 pipeline = [
12     { "$match": { "Runtime (Minutes)": { "$ne": "" }, "Revenue (Millions)": { "$ne": "" } }},
13     { "$project": {
14         "title": 1,
15         "runtime": { "$toInt": "$Runtime (Minutes)" },
16         "revenue": { "$toDouble": "$Revenue (Millions)" }
17     }}
18 ]
19 data = list(collection.aggregate(pipeline))
20
21 # Conversion en DataFrame Pandas
22 df = pd.DataFrame(data)
23
24 # Supprimer les valeurs nulles ou aberrantes
25 df = df.dropna(subset=["runtime", "revenue"]) # Supprimer les NaN
26
27 # Vérifier s'il y a des données suffisantes
28 if len(df) > 1:
29     # Calcul de la corrélation de Pearson
30     correlation, p_value = pearsonr(df["runtime"], df["revenue"])
31     print(f"Corrélation de Pearson: {correlation:.3f}")
32     print(f"Valeur p: {p_value:.3f}")
33 else:
34     print("Pas assez de données pour calculer une corrélation.")

```

```

(.venv) PS C:\Users\33622\Bazar\Desktop\Etudes\ESIEA\A4\Semestre 2\Tronc commun\INF4064 - NoSQL\Projet\Projet NoSQL> python .\mongo_quest12.py
Corrélation de Pearson: 0.306
Valeur p: 0.003

```

La corrélation de Pearson fournit un indice reflétant une relation linéaire entre deux variables continues : ici, la valeur positive de 0.306 signifie qu'il existe une corrélation positive entre la durée des films et leurs revenus, mais la corrélation reste assez faible / modérée. La p-valeur, inférieure au seuil 0.05, indique tout de même que le résultat obtenu est significatif.

13.

```

> db.films.aggregate([
  {$match: {"Runtime (Minutes)": {$ne: ""}, "year": {$ne: ""}},
  {$project: {decade: {$subtract: [{$toInt: "$year"}, {$mod: [{$toInt: "$year"}, 10]}]}, runtime: {$toInt: "$Runtime (Minutes)"}},
  {$group: {_id: "$decade", avg_runtime: {$avg: "$runtime"}}},
  {$sort: {_id: 1}}])
< {
  _id: 1970,
  avg_runtime: 116
}
{
  _id: 2000,
  avg_runtime: 147.625
}
{
  _id: 2010,
  avg_runtime: 120.13186813186813
}

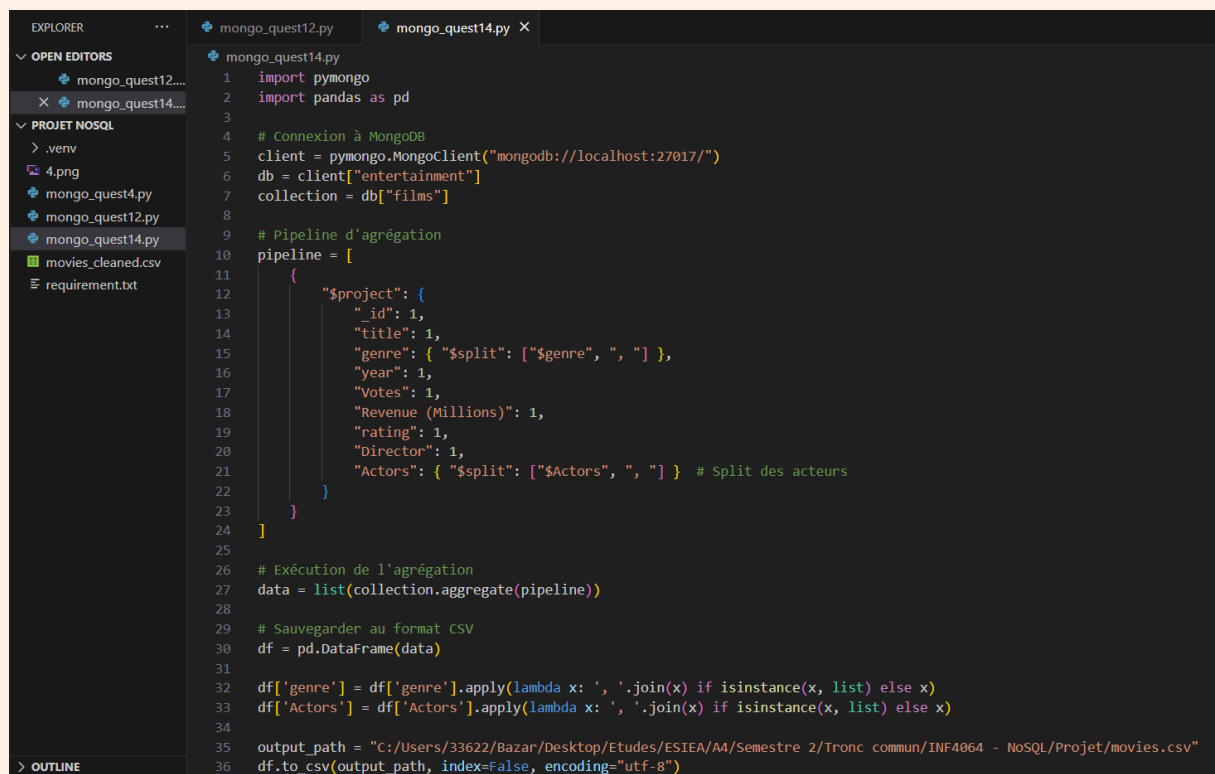
```

- `$match` : permet de ne pas prendre en compte les films pour lesquels la durée est inconnue ; permet de ne pas prendre en compte les films pour lesquels l'année est inconnue.
- `$project` : permet de calculer la décennie ; permet de convertir la durée en valeur numérique « integer ».
- `$group` : permet de regrouper les films par décennie, en associant la moyenne de durée en minutes.
- `$sort` : permet de trier les films selon la décennie, par ordre croissant.

Dans les années 2000, il y a un pic de durée pour les films en moyenne, à savoir 2 h 30 environ, là où les autres décennies sont à 2 h environ.

2. Base de données Neo4j

Pour intégrer les données de MongoDB à Neo4j, nous avons d'abord dû les importer : nous avons converti le fichier .json en fichier .csv, puis nous l'avons importé sur Google Drive et nous avons pu charger les données et les relations dans Neo4j grâce au lien de partage.



```

EXPLORER  ...  mongo_quest12.py  mongo_quest14.py X
OPEN EDITORS
  mongo_quest12...
  X mongo_quest14...
PROJET NOSQL
  .venv
  4.png
  mongo_quest4.py
  mongo_quest12.py
  mongo_quest14.py
  movies_cleaned.csv
  requirement.txt

1  import pymongo
2  import pandas as pd
3
4  # Connexion à MongoDB
5  client = pymongo.MongoClient("mongodb://localhost:27017/")
6  db = client["entertainment"]
7  collection = db["films"]
8
9  # Pipeline d'agrégation
10 pipeline = [
11     {
12         "$project": {
13             "_id": 1,
14             "title": 1,
15             "genre": { "$split": ["$genre", ", "] },
16             "year": 1,
17             "Votes": 1,
18             "Revenue (Millions)": 1,
19             "rating": 1,
20             "Director": 1,
21             "Actors": { "$split": ["$Actors", ", "] } # Split des acteurs
22         }
23     }
24 ]
25
26 # Exécution de l'agrégation
27 data = list(collection.aggregate(pipeline))
28
29 # Sauvegarder au format CSV
30 df = pd.DataFrame(data)
31
32 df['genre'] = df['genre'].apply(lambda x: ', '.join(x) if isinstance(x, list) else x)
33 df['Actors'] = df['Actors'].apply(lambda x: ', '.join(x) if isinstance(x, list) else x)
34
35 output_path = "C:/Users/33622/Bazar/Desktop/Etudes/ESIEA/A4/Semestre 2/Tronc commun/INF4064 - NoSQL/Projet/movies.csv"
36 df.to_csv(output_path, index=False, encoding="utf-8")

```

```

1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?
  export=download&id=14iNjLLjCtF1nht3btyW_E692t_A18CIC' AS row
2 CREATE (:Film {
3     id: row._id,
4     title: row.title,
5     year: toInteger(row.year),
6     votes: toInteger(row.Votes),
7     revenue: toFloat(row.`Revenue (Millions)`),
8     rating: row.rating,
9     director: row.Director
10 });

```

```

1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?
  export=download&id=14iNjLLjCtF1nht3btyW_E692t_A18CIC' AS row
2 WITH row, split(row.actors, ',') AS actors
3 UNWIND actors AS actorName
4 WITH distinct trim(actorName) AS actorName, row._id AS filmId
5 WHERE actorName IS NOT NULL AND trim(actorName) <> ""
6 MERGE (a:Actor {name: trim(actorName)})

```

```

1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?
  export=download&id=14iNjLLjCtF1nht3btyW_E692t_A18CIC' AS row
2 WITH row
3 WHERE row.actors IS NOT NULL AND trim(row.actors) <> ""
4 UNWIND split(row.actors, ',') AS actorName
5 WITH DISTINCT trim(actorName) AS actorName, row
6 MERGE (a:Actor {name: actorName})
7 WITH a, row
8 MATCH (f:Film {id: row._id})
9 MERGE (a)-[:A_JOUE]->(f);

```

```

1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?
  export=download&id=14iNjLLjCtF1nht3btyW_E692t_A18CIC' AS row
2 WITH row, split(row.genre, ',') AS genres
3 UNWIND genres AS genre
4 WITH DISTINCT trim(genre) AS genre
5 WHERE genre IS NOT NULL AND genre <> ""
6 MERGE (g:Genre {name: genre});

```

```

1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?
  export=download&id=14iNjLLjCtF1nht3btyW_E692t_A18CIC' AS row
2 WITH row, split(row.genre, ',') AS genres
3 UNWIND genres AS genre
4 WITH DISTINCT trim(genre) AS genre, row
5 WHERE genre IS NOT NULL AND genre <> ""
6 MATCH (f:Film {id: row._id})
7 MERGE (g:Genre {name: genre})
8 MERGE (f)-[:A_POUR_GENRE]->(g);

```

```

1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?
  export=download&id=14iNjLLjCtF1nht3btyW_E692t_A18CIC' AS row
2 WITH row
3 WHERE row.Director IS NOT NULL AND trim(row.Director) <> ""
4 MERGE (d:Director {name: row.Director})

```

```

1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?
  export=download&id=14iNjLLjCtF1nht3btyW_E692t_A18CIC' AS row
2 WITH row
3 WHERE row.Director IS NOT NULL AND trim(row.Director) <> ""
4 MERGE (d:Director {name: row.Director})
5 WITH d, row
6 MATCH (f:Film {id: row._id})
7 MERGE (d)-[:A_REALISE]->(f);

```

Les différentes requêtes ci-dessus permettent d'importer les données stockées dans le fichier .csv que nous avons stocké dans un drive Google (https://drive.google.com/file/d/14iNjLLjCtF1nht3btyW_E692t_A18CIC/view?usp=sharing) et de créer les nœuds entre ces données.

14.

```

1 MATCH (a:Actor)-[:A_JOUE]->(f:Film)
2 WITH a, COUNT(f) AS filmCount
3 ORDER BY filmCount DESC
4 LIMIT 1
5 RETURN a.name AS Actor, filmCount AS NumberOfFilms;

```

Table RAW

Actor	NumberOfFilms
"Matthew McConaughey"	4

Started streaming 1 record after 63 ms and completed after 70 ms.

- MATCH : permet de rechercher un motif dans la base de données, à savoir ici tous les films dans lesquels chaque acteur a joué.
- WITH : permet de passer des résultats intermédiaires à la suite de la requête, en comptant notamment ici le nombre de films par acteur.
- ORDER BY : permet de trier les résultats, ici par ordre décroissant du nombre de films (DESC).
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, le premier).
- RETURN : permet d'afficher le résultat escompté (ici, le nom de l'acteur et le nombre de films dans lesquels il a joué).

```

1 MATCH (a:Actor)-[:A_JOUE]->(f:Film)
2 WITH a, COUNT(f) AS filmCount
3 ORDER BY filmCount DESC
4 RETURN a.name AS Actor, filmCount AS NumberOfFilms;

```

	Actor	NumberOfFilms
1	"Matthew McConaughey"	4
2	"Chris Pratt"	4
3	"Scarlett Johansson"	4
4	"Ben Affleck"	4

Il faut noter que plusieurs autres acteurs ont joué dans 4 films, donc nous avons retiré le LIMIT pour avoir leur nom.

15.

```

1 MATCH (a1:Actor)-[:A_JOUE]->(f:Film)<-[:A_JOUE]-(a2:Actor)
2 WHERE a1.name = 'Anne Hathaway' AND a1 <> a2
3 RETURN DISTINCT a2.name AS Actor, f.title AS Film;

```

	Actor	Film
1	"Matthew McConaughey"	"Interstellar"
2	"Jessica Chastain"	"Interstellar"
3	"Mackenzie Foy"	"Interstellar"
4	"Jason Sudeikis"	"Colossal"
5	"Austin Stowell"	"Colossal"
6	"Tim Blake Nelson"	"Colossal"

Started streaming 6 records after 76 ms and completed after 77 ms.

- MATCH : permet de rechercher les noms des acteurs ayant joué dans les mêmes films.

- WHERE : permet de poser une condition, ici le nom d'un des acteurs devant être Anne Hathaway (sans correspondance avec elle-même) pour trouver les noms des acteurs ayant joué avec elle.
- RETURN DISTINCT : permet d'afficher le résultat escompté sans répétition (ici, le nom des acteurs ayant joué avec Anne Hathaway et les films dans lesquels ils ont joué ensemble).

16.

The screenshot shows a Cypher query interface. The query is as follows:

```

1 MATCH (a:Actor)-[:A_JOUE]->(f:Film)
2 WHERE f.revenue IS NOT NULL
3 WITH a, COLLECT(f.title) AS film_titles, COUNT(f) AS films_count,
  SUM(f.revenue) AS total_revenue
4 ORDER BY total_revenue DESC
5 RETURN a.name AS Actor, films_count AS NumberOfFilms, film_titles AS
  FilmTitles, total_revenue AS TotalRevenue
6 LIMIT 1;

```

Below the query, there are tabs for 'Table' (selected) and 'RAW'. To the right of the tabs are icons for JSON, search, and download. The results are displayed in a table with the following columns: Actor, NumberOfFilms, FilmTitles, and TotalRevenue.

Actor	NumberOfFilms	FilmTitles	TotalRevenue
"Chris Evans"	3	["Captain America: Civil War", "The Avengers", "Avengers: Age of Ultron"]	1490.35

At the bottom of the interface, a status message reads: "Started streaming 1 record after 68 ms and completed after 72 ms."

- MATCH : permet de rechercher tous les films dans lesquels chaque acteur a joué.
- WHERE : permet de ne sélectionner que les films où le revenu n'est pas nul.
- WITH : permet de transmettre les données, en collectant les noms des films, leur nombre et en comptant la somme totale de leur revenu.
- ORDER BY : permet de trier les résultats, ici par ordre décroissant de la somme des revenus des films (DESC).
- RETURN : permet d'afficher le résultat escompté (ici, le nom de l'acteur, le nombre de films dans lesquels il a joué, les titres de ces films et le revenu total de ces films).
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, le premier).

```

1 MATCH (a:Actor)-[:A_JOUER]->(f:Film)
2 WHERE f.revenue IS NOT NULL
3 WITH a, COLLECT(f.title) AS film_titles, COUNT(f) AS films_count,
   SUM(f.revenue) AS total_revenue
4 ORDER BY total_revenue DESC
5 RETURN a.name AS Actor, films_count AS NumberOfFilms, film_titles AS
   FilmTitles, total_revenue AS TotalRevenue;

```

Actor	Number	FilmTitles	TotalRevenue
1 "Chris Evans"	3	["Captain America: Civil War", "The Avengers", "Avengers: Age of Ultron"]	1490.35
2 "Robert Downey Jr."	3	["Captain America: Civil War", "The Avengers", "Avengers: Age of Ultron"]	1490.35
3 "Scarlett Johansson"	4	["Captain America: Civil War", "Sing", "The Avengers", "The P restige"]	1354.759999 9999998

Il faut noter que Robert Downey JR. a joué dans les mêmes films que Chris Evans, donc le revenu total généré est le même pour les deux acteurs.

17.

```

1 MATCH (f:Film)
2 WHERE f.votes IS NOT NULL
3 RETURN AVG(f.votes) AS AverageVotes;

```

AverageVotes
1 259963.74000000002

Started streaming 1 record after 67 ms and completed after 73 ms.

- MATCH : permet de rechercher tous les films.
- WHERE : permet de ne sélectionner que les films où les votes ne sont pas nuls.
- RETURN : permet d'afficher le résultat escompté (la moyenne des votes ici).

18.


```

1 MATCH (f:Film)-[:A_POUR_GENRE]->(g:Genre)
2 RETURN g.name AS Genre, COUNT(f) AS NumberOfFilms
3 ORDER BY NumberOfFilms DESC
4 LIMIT 1;

```

Table RAW

Genre	NumberOfFilms
"Drama"	49

Started streaming 1 record after 92 ms and completed after 97 ms.

- MATCH : permet de rechercher tous les genres des films.
- RETURN : permet d'afficher le résultat escompté (ici, le nom du genre et le nombre de films auxquels il est relié).
- ORDER BY : permet de trier les résultats, ici par ordre décroissant du nombre de films dans auxquels les genres sont liés (DESC).
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, le premier).

19.

```

1 MERGE (a:Actor {name: "Emeline Pellan"})
2 MERGE (f:Film {title: "Interstellar"})
3 MERGE (a)-[:A_JOUE]->(f);

```

Created 1 node, created 1 relationship, set 1 property, added 1 label Completed after 57 ms

Pour répondre à cette question, nous avons d'abord créé un nœud Emeline Pellan en tant qu'Acteur, ainsi que la liaison « A_JOUE » vers le film *Interstellar*.

```

1 MATCH (a1:Actor {name: "Emeline Pellan"})-[:A_JOUE]->(f1:Film)<-[:A_JOUE]->(a2:Actor),
2 (a2)-[:A_JOUE]->(f2:Film)
3 WHERE f2 <> f1
4 RETURN DISTINCT a2.name AS CoActor, f2.title AS FilmTitle;

```

Table RAW

CoActor	FilmTitle
"Matthew McConaughey"	"Gold"
"Matthew McConaughey"	"Sing"
"Matthew McConaughey"	"The Wolf of Wall Street"
"Anne Hathaway"	"Colossal"
"Jessica Chastain"	"Miss Sloane"

Started streaming 5 records after 118 ms and completed after 120 ms.

- **MATCH** : permet de rechercher les noms des acteurs ayant joué dans les mêmes films, avec la condition incluse que le nom du premier Acteur est Emeline PELLAN.
- **WHERE** : permet d'éviter la correspondance.
- **RETURN DISTINCT** : permet d'afficher le résultat escompté sans répétition (le nom des acteurs ayant joué avec Emeline Pellan dans *Interstellar* et le nom des autres films dans lesquels ces mêmes acteurs ont joué).

20.

```

1 MATCH (d:Director)-[:A_REALISE]->(f:Film)<-[:A_JOUE]-(a:Actor)
2 WITH d, COUNT(DISTINCT a) AS num_actors
3 ORDER BY num_actors DESC
4 LIMIT 1
5 RETURN d.name AS Director, num_actors AS NumberOfActors;

```

Table RAW

Director	NumberOfActors
"Christopher Nolan"	15

Started streaming 1 record after 116 ms and completed after 130 ms.

- **MATCH** : permet de rechercher les films en leur associant leur réalisateur et leurs acteurs.
- **WITH** : permet de transmettre les données, en comptant le nombre d'acteurs associés à un réalisateur via un film.
- **ORDER BY** : permet de trier les résultats, ici par ordre décroissant du nombre d'acteurs (DESC).
- **LIMIT** : permet de n'afficher qu'un nombre limité de résultats (ici, le premier).
- **RETURN** : permet d'afficher le résultat escompté (ici, le nom du réalisateur et le nombre d'acteurs avec qui il a collaboré).

21.

```

1 MATCH (f1:Film)<-[:A_JOUE]-(a:Actor)-[:A_JOUE]->(f2:Film)
2 WHERE f1 <> f2 AND f1.title < f2.title
3 WITH f1, f2, COLLECT(DISTINCT a.name) AS shared_actors_names, COUNT(DISTINCT a) AS common_actors
4 ORDER BY common_actors DESC
5 LIMIT 5
6 RETURN f1.title AS Film1, f2.title AS Film2, common_actors AS NumberOfSharedActors, shared_actors_names AS SharedActorsNames;

```

Table RAW

Film1	Film2	Numbr	SharedActorsNames
"Pirates of the Caribbean: At World's End"	"Pirates of the Caribbean: Dead Man's Chest"	3	["Johnny Depp", "Orlando Bloom", "Keira Knightley"]
"Captain America: Civil War"	"The Avengers"	3	["Chris Evans", "Robert Downey Jr.", "Scarlett Johansson"]
"Avengers: Age of Ultron"	"Captain America: Civil War"	2	["Chris Evans", "Robert Downey Jr."]
"Pirates of the Caribbean: At World's End"	"Pirates of the Caribbean: On Stranger Tides"	2	["Johnny Depp", "Geoffrey Rush"]
"Avengers: Age of Ultron"	"The Avengers"	2	["Chris Evans", "Robert Downey Jr."]

Started streaming 5 records after 223 ms and completed after 384 ms.

- **MATCH** : permet de rechercher les connexions entre films et acteurs.
- **WHERE** : permet d'éviter la correspondance et les doublons.

- WITH : permet de transmettre les données, en collectant les noms des acteurs et en les comptant.
- ORDER BY : permet de trier les résultats, ici par ordre décroissant du nombre d'acteurs communs à plusieurs films.
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, les 5 premiers).
- RETURN : permet d'afficher le résultat escompté (ici, les titres des films, le nombre d'acteurs en commun et leur nom).

22.

```

1 MATCH (a:Actor)-[:A_JOUE]->(f:Film)<-[:A_REALISE]-(d:Director)
2 WITH a, COUNT(DISTINCT d) AS num_directors
3 ORDER BY num_directors DESC
4 LIMIT 5
5 RETURN a.name AS Actor, num_directors AS NumberOfDirectors;

```

Actor	NumberOfDirectors
1 "Scarlett Johansson"	4
2 "Ben Affleck"	4
3 "Chris Pratt"	4
4 "Matthew McConaughey"	4
5 "Bryce Dallas Howard"	3

Started streaming 5 records after 118 ms and completed after 133 ms.

- MATCH : permet de rechercher les films en leur associant leur réalisateur et leurs acteurs.
- WITH : permet de transmettre les données, en comptant le nombre de réalisateurs associés à un acteur.
- ORDER BY : permet de trier les résultats, ici par ordre décroissant du nombre de réalisateurs.
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, les 5 premiers).
- RETURN : permet d'afficher le résultat escompté (ici, les noms des acteurs ayant joué avec le plus de réalisateurs différents).

23.

```

1 MATCH (a:Actor {name: "Anne Hathaway"})-[:A_JOUE]->(f:Film)-[:A_POUR_GENRE]->(g:Genre)
2 MATCH (r:Film)-[:A_POUR_GENRE]->(g)
3 WHERE NOT (a)-[:A_JOUE]->(r)
4 RETURN DISTINCT r.title AS RecommendedFilm, COLLECT(DISTINCT g.name) AS Genres
5 ORDER BY SIZE(COLLECT(DISTINCT g.name)) DESC
6 LIMIT 1;

```

Table RAW

RecommendedFilm	Genres
"Assassin's Creed"	["Drama", "Adventure", "Action"]

Started streaming 1 record after 455 ms and completed after 494 ms.

- MATCH 1 : permet de rechercher les genres des films dans lesquels un acteur a joué (ici, Anne Hathaway).
- MATCH 2 : permet de rechercher les films ayant les mêmes genres que ceux dans lesquels Anne Hathaway a joué pour permettre les recommandations.
- WHERE NOT : permet de ne pas sélectionner les films dans lesquels Anne Hathaway a déjà joué.
- RETURN DISTINCT : permet d'afficher le résultat escompté (ici, le titre du film que l'on recommande, ainsi que ses genres).
- ORDER BY : permet de trier les résultats, ici par ordre décroissant du nombre de genres en commun pour avoir de meilleures correspondances.
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, le premier).

24.

```

1 MATCH (d1:Director)-[:A_REALISE]->(f1:Film)-[:A_POUR_GENRE]->(g1:Genre)
2 WITH d1, COLLECT(DISTINCT g1.name) AS genres_d1
3 MATCH (d2:Director)-[:A_REALISE]->(f2:Film)-[:A_POUR_GENRE]->(g2:Genre)
4 WHERE d1 <> d2
5 WITH d1, d2, genres_d1, COLLECT(DISTINCT g2.name) AS genres_d2
6 WHERE size(apoc.coll.intersection(genres_d1, genres_d2)) > 2
7 MERGE (d1)-[:INFLUENCE_PAR]->(d2)
8 MERGE (d2)-[:INFLUENCE_PAR]->(d1);

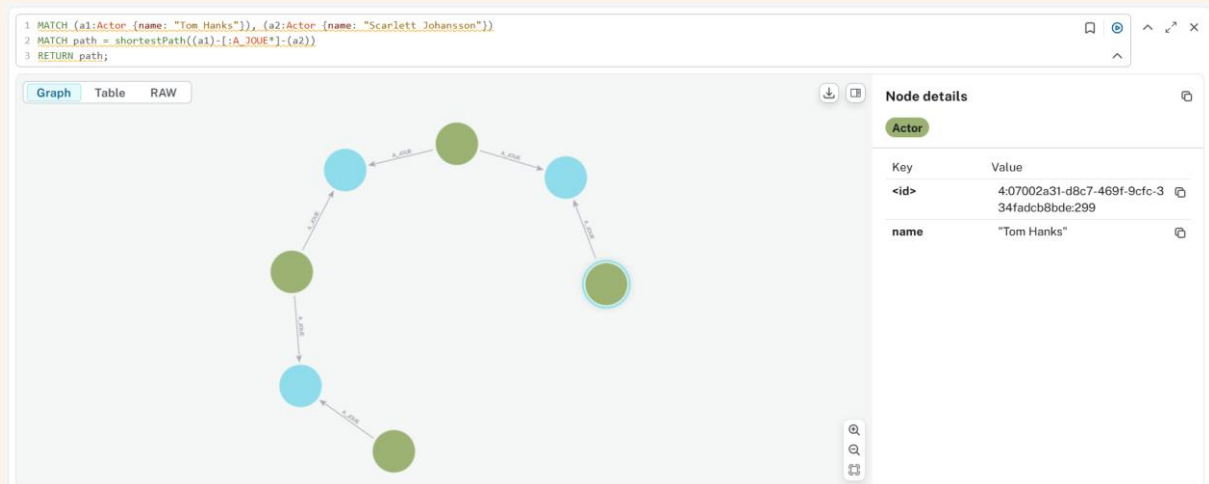
```

Created 258 relationships Completed after 931 ms

- MATCH 1 : permet de rechercher les genres des films en leur associant leur réalisateur.
- WITH 1 : permet de transmettre les données, en collectant les noms des genres des films.
- MATCH 2 : permet de rechercher les genres des films en leur associant leur réalisateur une deuxième fois.
- WHERE 2 : permet d'éviter la correspondance.
- WITH 2 : permettre de transmettre les données, en collectant les noms des genres des films des réalisateurs d1 et d2.
- WHERE : permet de comparer les deux listes de genres et retourne uniquement les genres communs, en gardant seulement les réalisateurs qui ont au moins 3 genres en commun.
- MERGE 1 : permet de créer une relation « INFLUENCE_PAR » du réalisateur d1 au réalisateur d2.

- MERGE 2 : permet de créer une relation « INFLUENCE_PAR » du réalisateur d2 au réalisateur d1.

25.



```
(:Actor {name: "Tom Hanks"})-[:A_JOUE]->(:Film {title: "Sully"})<-[:A_JOUE]-
(:Actor {name: "Aaron Eckhart"})-[:A_JOUE]->(:Film {title: "The Dark Knight"})<-
[:A_JOUE]-(:Actor {name: "Michael Caine"})-[:A_JOUE]->(:Film {title: "The
Prestige"})<-[:A_JOUE]-(:Actor {name: "Scarlett Johansson"})
```

- MATCH 1 : permet de rechercher les nœuds des acteurs Tom Hanks et Scarlett Johansson.
- MATCH 2 : permet de trouver le chemin le plus court entre deux nœuds grâce à la fonction shortestPath() de Neo4j.
- RETURN : permet d'afficher le résultat escompté (ici, le chemin entre Tom Hanks et Scarlett Johansson).

26.

L'algorithme de Louvain n'est pas disponible dans la version gratuite de Neo4j, nous avons donc décidé de nous adapter : le but est de trouver des paires d'acteurs ayant joué ensemble dans plusieurs films.

```

1 MATCH (a1:Actor)-[:A_JOUE]->(f:Film)-[:A_JOUE]-(a2:Actor)
2 WHERE a1 < a2 AND a1.name < a2.name
3 WITH a1, a2, COUNT(f) AS common_movies, COLLECT(f.title) AS common_movies_titles
4 WHERE common_movies > 1
5 RETURN a1.name AS Actor1, a2.name AS Actor2, common_movies AS CommonMovies, common_movies_titles AS CommonMoviesTitle;

```

Actor1	Actor2	CommonMovies	CommonMoviesTitle
"Chris Evans"	"Robert Downey Jr."	3	["Captain America: Civil War", "The Avengers", "Avengers: Age of Ultron"]
"Chris Evans"	"Scarlett Johansson"	2	["Captain America: Civil War", "The Avengers"]
"Robert Downey Jr."	"Scarlett Johansson"	2	["Captain America: Civil War", "The Avengers"]
"Christian Bale"	"Michael Caine"	2	["The Dark Knight", "The Prestige"]
"Geoffrey Rush"	"Johnny Depp"	2	["Pirates of the Caribbean: On Stranger Tides", "Pirates of the Caribbean: At World's End"]
"Johnny Depp"	"Orlando Bloom"	2	["Pirates of the Caribbean: Dead Man's Chest", "Pirates of the Caribbean: At World's End"]
"Keira Knightley"	"Orlando Bloom"	2	["Pirates of the Caribbean: Dead Man's Chest", "Pirates of the Caribbean: At World's End"]
"Johnny Depp"	"Keira Knightley"	2	["Pirates of the Caribbean: Dead Man's Chest", "Pirates of the Caribbean: At World's End"]

Started streaming 8 records after 90 ms and completed after 93 ms.

- **MATCH** : permet de rechercher les acteurs ayant joué dans un même film.
- **WHERE 1** : permet d'éviter les correspondances et les doublons.
- **WITH** : permet de transmettre les données, en comptant les films en commun et en collectant leur titre.
- **WHERE 2** : permet de ne garder que les acteurs ayant joué ensemble dans au moins deux films.
- **RETURN** : permet d'afficher le résultat escompté (ici, les noms des acteurs, le nombre de films dans lesquels ils ont été coacteurs, ainsi que les titres de ces films).

3. Questions transversales

27.

```

1 MATCH (f1:Film)-[:A_POUR_GENRE]->(g:Genre)-[:A_POUR_GENRE]-(f2:Film)
2 MATCH (f1)-[:A_REALISE]-(d1:Director)-[:A_REALISE]-(d2:Director)
3 WHERE f1 < f2 AND d1 < d2
4 WITH f1, d1, f2, d2, COLLECT(DISTINCT g.name) AS shared_genres
5 WHERE size(shared_genres) >= 2
6 RETURN f1.title AS Film1, d1.name AS Director1,
7       f2.title AS Film2, d2.name AS Director2,
8       shared_genres
9 ORDER BY size(shared_genres) DESC
10 LIMIT 10;

```

Film1	Director1	Film2	Director2	SharedGenres
"Mad Max: Fury Road"	"George Miller"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Batman v Superman: Dawn of Justice"	"Zack Snyder"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Captain America: Civil War"	"Anthony Russo"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Inception"	"Christopher Nolan"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Avengers: Age of Ultron"	"Joss Whedon"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Star Trek Beyond"	"Justin Lin"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Guardians of the Galaxy"	"James Gunn"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"X-Men: Apocalypse"	"Bryan Singer"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Independence Day: Resurgence"	"Roland Emmerich"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]
"Jurassic World"	"Colin Trevorrow"	"Rogue One"	"Gareth Edwards"	["Adventure", "Action", "Sci-Fi"]

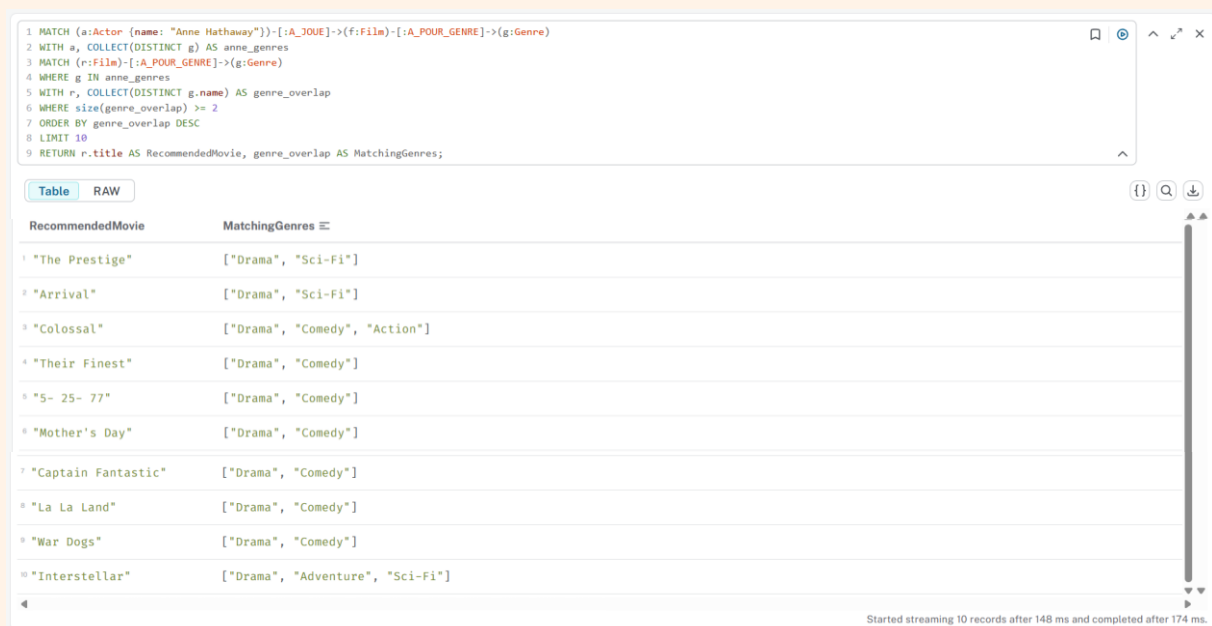
Started streaming 10 records after 586 ms and completed after 771 ms.

> ⓘ This query builds a cartesian product between disconnected patterns.

- **MATCH 1** : permet de rechercher des films qui partagent au moins un même genre en commun.
- **MATCH 2** : permet de rechercher les réalisateurs de ces films.

- WHERE 1 : permet d'éviter les doublons.
- WITH : permet de transmettre les données, en collectant les noms des genres partagés.
- WHERE 2 : permet de ne garder que les films ayant au moins deux genres en commun.
- RETURN : permet d'afficher le résultat escompté (ici, les noms des films ainsi que ceux de leur réalisateur, en précisant les genres en commun).
- ORDER BY : permet de trier les résultats, ici par ordre décroissant du nombre de genres en commun (DESC).
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, les 10 premiers).

28.



```

1 MATCH (a:Actor {name: "Anne Hathaway"})-[:A_JOUÉ]->(f:Film)-[:A_POUR_GENRE]->(g:Genre)
2 WITH a, COLLECT(DISTINCT g) AS anne_genres
3 MATCH (r:Film)-[:A_POUR_GENRE]->(g:Genre)
4 WHERE g IN anne_genres
5 WITH r, COLLECT(DISTINCT g.name) AS genre_overlap
6 WHERE size(genre_overlap) >= 2
7 ORDER BY genre_overlap DESC
8 LIMIT 10
9 RETURN r.title AS RecommendedMovie, genre_overlap AS MatchingGenres;

```

RecommendedMovie	MatchingGenres
1 "The Prestige"	["Drama", "Sci-Fi"]
2 "Arrival"	["Drama", "Sci-Fi"]
3 "Colossal"	["Drama", "Comedy", "Action"]
4 "Their Finest"	["Drama", "Comedy"]
5 "5- 25- 77"	["Drama", "Comedy"]
6 "Mother's Day"	["Drama", "Comedy"]
7 "Captain Fantastic"	["Drama", "Comedy"]
8 "La La Land"	["Drama", "Comedy"]
9 "War Dogs"	["Drama", "Comedy"]
10 "Interstellar"	["Drama", "Adventure", "Sci-Fi"]

Started streaming 10 records after 148 ms and completed after 174 ms.

- MATCH 1 : permet de rechercher les genres des films dans lesquels Anne Hathaway a joué.
- WITH 1 : permet de transmettre les données, en collectant les noms des genres trouvés précédemment.
- MATCH 2 : permet de rechercher les films ayant les mêmes genres que ceux dans lesquels Anne Hathaway a joué pour permettre les recommandations.
- WHERE 2 : permet de ne sélectionner que les genres présents dans la liste des genres liés aux films dans lesquels Anne Hathaway a pu jouer.
- WITH 2 : permet de transmettre les données, en collectant les noms des genres trouvés.
- WHERE 2 : permet de n'afficher que les films ayant au moins deux genres en commun pour améliorer les recommandations.
- ORDER BY : permet de trier les résultats, ici par ordre inverse de l'ordre alphabétique des genres en commun (DESC).
- LIMIT : permet de n'afficher qu'un nombre limité de résultats (ici, les 10 premiers).

- **RETURN** : permet d'afficher le résultat escompté (ici, le nom des films recommandés, ainsi que leur(s) genre(s)).

29.

```
1 MATCH (d1:Director)-[:A_REALISE]->(f1:Film)-[:A_POUR_GENRE]->(g:Genre),
2   (d2:Director)-[:A_REALISE]->(f2:Film)-[:A_POUR_GENRE]->(g:Genre)
3 WHERE d1 <> d2
4 AND f1.year = f2.year
5 WITH d1, d2, f1.year AS year, f1, f2, COUNT(DISTINCT g) AS num_genres
6 WHERE num_genres >= 2
7 MERGE (d1)-[:CONCURRENCE {shared_genres: num_genres, year: year}]->(d2);
```

Created 584 relationships, set 1168 properties Completed after 930 ms

- **MATCH** : permet de rechercher différents films avec différents réalisateurs, mais avec les mêmes genres.
- **WHERE 1** : permet d'éviter les doublons.
- **AND** : permet de comparer les années de sortie des films la même année.
- **WITH** : permet de transmettre les données, en comptant le nombre de genres.
- **WHERE** : permet de ne garder que les films pour lesquels il y a au moins 2 genres en commun.
- **MERGE** : permet de créer la relation « concurrence » entre les réalisateurs, en ajoutant en tant que propriétés le nombre de genres partagés et l'année de sortie.

30.

```
1 LOAD CSV WITH HEADERS FROM 'https://drive.google.com/uc?export=download&id=14iNjLLjCtFihnt3btyW_E692t_A18CIC' AS row
2 MATCH (f:Film {id: row._id})
3 SET f.metascore = toInteger(row.Metascore);
```

Set 94 properties Completed after 2225 ms

```
1 MATCH (d:Director)-[:A_REALISE]->(f:Film)-[:A_JOUE]->(a:Actor)
2 WHERE f.revenue IS NOT NULL AND f.metascore IS NOT NULL
3 WITH d, a, COUNT(f) AS common_films, AVG(f.revenue) AS avg_revenue, AVG(f.metascore) AS avg_metascore
4 WHERE common_films >= 2
5 RETURN d.name AS Director,
6   a.name AS Actor,
7   common_films AS NumberOfFilmsTogether,
8   avg_revenue AS AvgRevenue,
9   avg_metascore AS AvgMetascore
10 ORDER BY common_films DESC, avg_revenue DESC;
```

Table RAW

Director	Actor	NumberOfFilmsTog	AvgRevenue	AvgMetascore
"Joss Whedon"	"Chris Evans"	2	541.135	67.5
"Joss Whedon"	"Robert Downey Jr."	2	541.135	67.5
"Gore Verbinski"	"Johnny Depp"	2	366.215	51.5
"Gore Verbinski"	"Orlando Bloom"	2	366.215	51.5
"Gore Verbinski"	"Keira Knightley"	2	366.215	51.5
"Christopher Nolan"	"Christian Bale"	2	293.20000000000005	74.0
"Christopher Nolan"	"Michael Caine"	2	293.20000000000005	74.0
"Martin Scorsese"	"Leonardo DiCaprio"	2	124.62	80.0
"Peter Berg"	"Mark Wahlberg"	2	46.57	68.5

Started streaming 9 records after 183 ms and completed after 193 ms.

- **MATCH** : permet de rechercher les acteurs et les réalisateurs liés par les films pour lesquels ils ont respectivement joué et réalisé.
- **WHERE 1** : permet de vérifier que les valeurs de revenue et de score ne sont pas nulles.
- **WITH** : permet de transmettre les données, en comptant le nombre de films en commun, la moyenne des revenus et la moyenne des metascors.

- WHERE 2 : permet de ne sélectionner que les réponses pour lesquelles il y a au moins 2 films en commun.
- RETURN : permet d'afficher le résultat escompté (ici, les noms des réalisateurs et acteurs, ainsi que le nombre de films dans lesquels ils ont collaboré, le revenu moyen et le metascote moyen).
- ORDER BY : permet de trier les résultats, ici par ordre décroissant du nombre de films en commun et du revenu généré (DESC).

II – Application Python

1. Introduction

Dans le cadre de ce projet académique, nous avons exploré deux types de bases de données NoSQL : MongoDB, qui est une base de données orientée document, et Neo4j, une base de données orientée graphe. Le but était de créer une application Python avec Streamlit, qui permet de se connecter à ces deux bases de données hébergées dans le cloud, d'exécuter des requêtes pour extraire des données pertinentes, et de visualiser les résultats de manière interactive.

2. Objectifs du projet

Les objectifs principaux étaient les suivants :

- établir une connexion sécurisée avec des instances cloud de MongoDB et Neo4j ;
- effectuer des requêtes sur MongoDB pour récupérer des informations sur les films ;
- utiliser Neo4j pour modéliser les relations entre les films, les acteurs et les réalisateurs ;
- réaliser des analyses statistiques et des visualisations des données obtenues.

3. Configuration et mise en place

3.1 Environnement virtuel

Pour garantir une bonne gestion des dépendances et la portabilité du projet, un environnement virtuel Python a été configuré. Voici les commandes utilisées :

```
python -m venv .venv
source .venv/bin/activate # Sur macOS/Linux
# ou
.venv\Scripts\activate # Sur Windows
```

3.2 Installation des dépendances

Le fichier requirements.txt contient toutes les bibliothèques nécessaires. Après activation de l'environnement virtuel, les dépendances ont été installées avec la commande suivante :

```
pip install -r requirements.txt
```

3.3 Configuration des bases de données

Les connexions aux bases MongoDB et Neo4j ont été établies à l'aide de variables d'environnement, stockées dans le fichier config.py, où sont conservées les informations sensibles comme les URI de connexion et les identifiants.

4. Structure de l'application

L'application se compose de plusieurs fichiers Python, chacun ayant un rôle spécifique dans le projet. Voici un aperçu de la structure du projet :

- **connexion_mongodb.py & connexion_neo4j.py** : ces fichiers contiennent les fonctions pour se connecter aux bases de données MongoDB et Neo4j.
- **queries_mongodb.py** : ce fichier contient les requêtes MongoDB pour extraire des informations sur les films, telles que :
 - trouver l'année avec le plus grand nombre de films sortis ;
 - calculer la moyenne des votes des films sortis en 2007 ;
 - afficher un histogramme du nombre de films par année.
- **queries_neo4j.py** : ce fichier permet de créer des nœuds et des relations dans Neo4j, et de réaliser des requêtes pour obtenir des informations comme :
 - l'acteur ayant joué dans le plus grand nombre de films ;
 - le film le plus populaire selon les revenus.
- **app_streamlit.py** : cette interface Streamlit permet de visualiser les résultats des requêtes MongoDB et Neo4j, ainsi que de générer des graphiques interactifs.

5. Difficultés rencontrées

5.1 Gestion de l'affichage des relations

L'affichage des relations entre films, acteurs et réalisateurs a posé un problème, certaines connexions n'apparaissant pas correctement dans les requêtes Cypher ou les graphes générés. Pour résoudre cela, les requêtes Cypher ont été ajustées pour garantir que les relations étaient correctement créées. De plus, la commande MATCH (n)-[r]->(m) RETURN n,r,m LIMIT 25 a permis d'afficher un échantillon des relations. Pour améliorer la visualisation, nous avons intégré les bibliothèques pyvis et NetworkX, permettant de générer un graphe interactif directement dans Streamlit.

6. Conclusion

Ce projet a permis de combiner MongoDB et Neo4j pour analyser des données relatives aux films. En répondant aux différentes questions du projet, nous avons exploré des méthodes d'agrégation et d'interrogation dans MongoDB, tout en tirant parti de la puissance de Neo4j pour modéliser des graphes complexes et effectuer des analyses avancées. L'application développée avec Streamlit offre une interface intuitive et interactive pour interagir avec les bases de données et visualiser les résultats.

7. Annexes

7.1 Code source

Le code source complet est disponible sur le dépôt GitHub :
<https://github.com/CarotteT/NoSQL.git>.

7.2 Références

- Documentation MongoDB
- Documentation Neo4j
- Documentation Streamlit