



UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Exploration des concepts et possibilités des SGBD de type graphes (Neo4j) en comparaison avec les SGBD relationnels

THÈSE DE BACHELOR

CYRIL HUNZIKER

No étudiant : 14-413-116

Fribourg, Juillet 2023

Encadré par :

Prof. Dr. Jacques PASQUIER–ROCHA
Software Engineering Group



UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Département d'Informatique
Université de Fribourg, Suisse

<http://diuf.unifr.ch>



Table des matières

| | |
|--|-----------|
| 1 Introduction | 3 |
| 1.1 Problématique | 3 |
| 1.2 Objectif | 3 |
| 1.3 Structure du rapport | 4 |
| 1.4 Convention..... | 4 |
| 2 Modélisation des données | 5 |
| 2.1 Le modèle entité-association | 5 |
| 2.2 Les bases de données relationnelles | 7 |
| 2.3 Les bases de données graphiques | 9 |
| 2.4 Système de gestion des bases de données de type graphe | 12 |
| 2.4.1 Neo4j | 12 |
| 2.4.2 Autres SGBD..... | 12 |
| 2.5 Langage de requête NoSQL..... | 12 |
| 2.5.1 Le langage Cypher | 13 |
| 2.5.2 Requêtes linéaires Cypher | 14 |
| 3 Implémentation | 17 |
| Etude de cas : Les films Marvel | 17 |
| 3.1 Génération des données | 17 |
| 3.2 Importation des données | 23 |
| 3.3 Résultat et exploration | 24 |
| 3.4 Analyse avec Query Neo4j | 25 |
| 3.5 Analyse avec « Explore » de Neo4j..... | 27 |
| 3.6 Récapitulatif de Neo4j | 29 |
| 4 Conclusion | 30 |
| 4.1 Conclusion | 30 |
| References | 31 |

Liste des Figures

| | |
|---|----|
| Figure 1 : <i>Modèle entité association (modélisé avec Lucidchart)</i> | 6 |
| Figure 2 : <i>Schéma d'une base de données relationnel (modélisé avec Lucidchart)</i> | 7 |
| Figure 3 : <i>Symboles des diagrammes entité-association physiques</i> | 8 |
| Figure 4 : <i>Schéma d'une base de données graphique</i> | 10 |
| Figure 5 : <i>Exemple d'un échantillon de base de donnée graphe créé avec Neo4j</i> | 11 |
| Figure 6 : <i>Exemple 1 de requête Cypher dans Neo4j</i> | 15 |
| Figure 7 : <i>Exemple 2 de requête Cypher dans Neo4j</i> | 16 |
| Figure 8 : <i>Données dans fichier CSV pour le nœud « Movies »</i> | 19 |
| Figure 9 : <i>Interface Neo4j : association du fichier CSV au nœud « Movies »</i> | 20 |
| Figure 10 : <i>Interface Neo4j : association du fichier CSV à la relation « CASTING »</i> | 20 |
| Figure 11 : <i>Données dans le fichier CSV pour la relation « CASTING »</i> | 21 |
| Figure 12 : <i>Création des données avec ChatGPT</i> | 22 |
| Figure 13 : <i>Schéma final créé sur Neo4j avant de confirmer l'importation</i> | 23 |
| Figure 14 : <i>Graphe généré par Neo4j avec la totalité de nos données</i> | 24 |
| Figure 15 : <i>Graphe généré par Neo4j avec la totalité de nos données (zoomé)</i> | 24 |
| Figure 16 : <i>Interface Query de Neo4j</i> | 25 |
| Figure 17 : <i>Interface Query de Neo4j (2)</i> | 26 |
| Figure 18 : <i>Interface Explore de Neo4j, création d'une scène</i> | 27 |
| Figure 19 : <i>Interface Explore de Neo4j, exploration des données</i> | 27 |
| Figure 20 : <i>Interface Explore de Neo4j, exploration des données (2)</i> | 28 |
| Figure 21 : <i>Interface Explore de Neo4j, exploration des données (3)</i> | 28 |
| Figure 22 : <i>Interface Explore de Neo4j, exploration des données (4)</i> | 29 |

1 Introduction

1.1 Problématique

Les Systèmes de Gestion de Bases de Données (SGBD) [1] jouent un rôle central dans la gestion, le stockage et la récupération efficace des données dans le domaine de l'informatique et des technologies de l'information. Au fil des décennies, les SGBD relationnels ont été la norme incontestée en matière de gestion de données structurées, fournissant une méthode robuste pour organiser et interroger des données tabulaires.

Cependant, avec la montée en puissance des données hautement connectées et complexes, de nouveaux types de systèmes de gestion de base de données ont été développés pour répondre aux besoins de stockage et de manipulation de ces données.

Les systèmes de gestion de base de données de type graphe sont l'un de ces nouveaux types de systèmes, qui ont attiré l'attention en raison de leur capacité à stocker des données hautement connectées sous forme de graphes.

1.2 Objectif

Le but de ce travail de bachelor est d'explorer les concepts et les possibilités des systèmes de gestion de base de données de type graphe, en utilisant Neo4j [2] comme exemple de SGBD de type graphe, en comparaison avec les SGBD relationnels. Nous examinerons les différences fondamentales entre les deux types de systèmes, les avantages et les inconvénients de chaque système, et les scénarios dans lesquels l'utilisation d'un SGBD de type graphe peut être plus avantageuse qu'un SGBD relationnel.

Ce travail de bachelor vise à fournir une compréhension approfondie des SGBD de type graphe et de leur potentiel, en mettant l'accent sur la comparaison avec les SGBD relationnels, qui sont les systèmes les plus couramment utilisés. Nous espérons que cette exploration permettra de mieux comprendre les limites des SGBD relationnels et les avantages des SGBD de type graphe dans le contexte des données hautement connectées et complexes.

1.3 Structure du rapport

Afin de réaliser au mieux cet objectif, le rapport est structuré comme suit :

Chapitre 1 : Introduction, présentation de la problématique et de l'objectif

Chapitre 2 : Modélisation des données, présentation des différents moyens de modéliser les données et introduction aux systèmes de gestion de base de données de type graphe

Chapitre 3 : Étude de cas avec implémentation, présentation d'une étude de cas sur les films Marvel [3] avec Neo4j

Chapitre 4 : Conclusion, récapitulatif et potentielle suite

1.4 Convention

Afin de réaliser ce rapport, nous avons utilisé des figures allant de 1 à 22, le titre de chaque figure est rédigé en dessous de l'image sous cette forme : « **Figure X** : (Titre) ».

Les références sont notées comme ceci : « Exemple [numéro de la référence] », la liste de toutes les références est écrite à la fin du rapport.

2 Modélisation des données

2.1 Le modèle entité-association

« Un diagramme entité-association [4] est un type d'organigramme illustrant la façon dont des « entités » telles que des personnes, objets ou concepts sont liés les unes aux autres au sein d'un système.

Les diagrammes entité-association sont généralement utilisés pour concevoir ou déboguer des bases de données relationnelles dans le domaine de l'ingénierie informatique, des systèmes d'information d'entreprise, de l'éducation et de la recherche. Appelés ERD en anglais, ils utilisent une série de symboles prédéfinis tels que des rectangles, losanges et ovales reliés par des lignes pour décrire les interconnexions entre les entités, leurs relations et leurs attributs.

Ils imitent une structure grammaticale, où les entités sont des noms et les relations des verbes. » [5]

Ce modèle utilise une notation particulière pour expliquer son fonctionnement :

- « 1 » Cela signifie qu'une entité est associée à exactement une instance d'une autre entité dans la relation.
- « 0,1 » Cela signifie qu'une entité peut être associée à zéro ou une instance d'une autre entité dans la relation.
- « m » Cela signifie qu'une entité peut être associée à une ou plusieurs instances d'une autre entité dans la relation.
- « mc » Cela signifie qu'une entité peut être associée à zéro, une ou plusieurs instances d'une autre entité dans la relation.

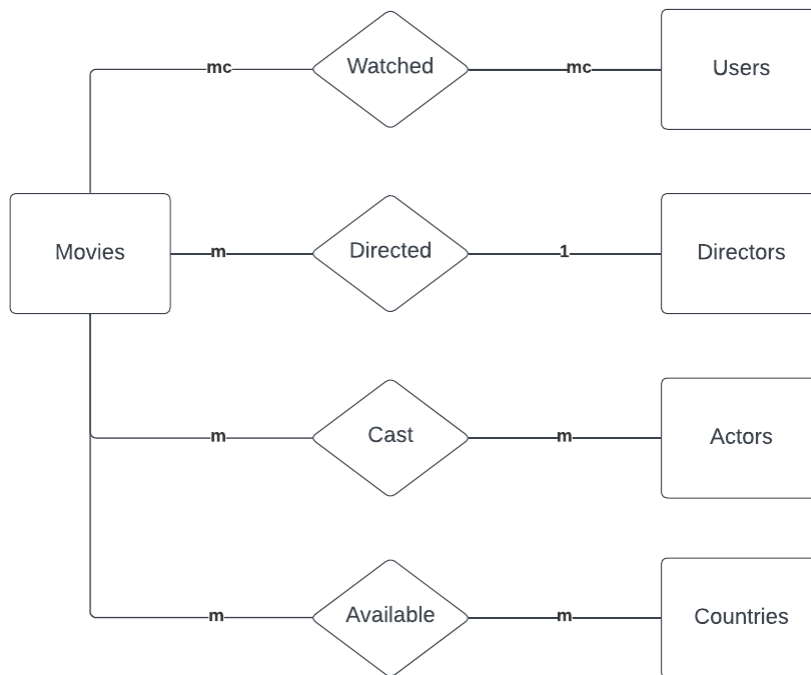


Figure 1 : *Modèle entité association (modélisé avec Lucidchart [6])*

Dans ce modèle (**Figure 1**), on a 5 entités : Movies, Users, Directors, Actors et Countries, ainsi que 4 relations : Watched, Directed, Cast et Available. Avec ce modèle, on comprend que :

- Chaque film peut être vu par zéro, un ou plusieurs utilisateurs
- Chaque utilisateur peut avoir vu zéro, un ou plusieurs films
- Chaque film a été dirigé par un directeur
- Chaque directeur a dirigé un ou plusieurs films
- Chaque film met en scène un ou plusieurs acteurs
- Chaque acteur est mis en scène par un ou plusieurs films
- Chaque film est disponible dans un ou plusieurs pays
- Chaque pays met à disposition un ou plusieurs films

Ce modèle nous permet une compréhension visuelle de la base de données que nous souhaitons créer. Maintenant, il nous est beaucoup plus facile d'implémenter cette base de données, qu'elle soit relationnelle ou graphique.

2.2 Les bases de données relationnelles

Une base de données relationnelle est un système de gestion de bases de données qui repose sur une structure bien définie et rigoureuse. Elle utilise des tables pour stocker les données, où chaque table contient des lignes représentant des enregistrements individuels et des colonnes représentant les attributs de ces enregistrements. Chaque table est liée aux autres par des clés primaires et des clés étrangères, établissant ainsi des relations entre les différentes entités.

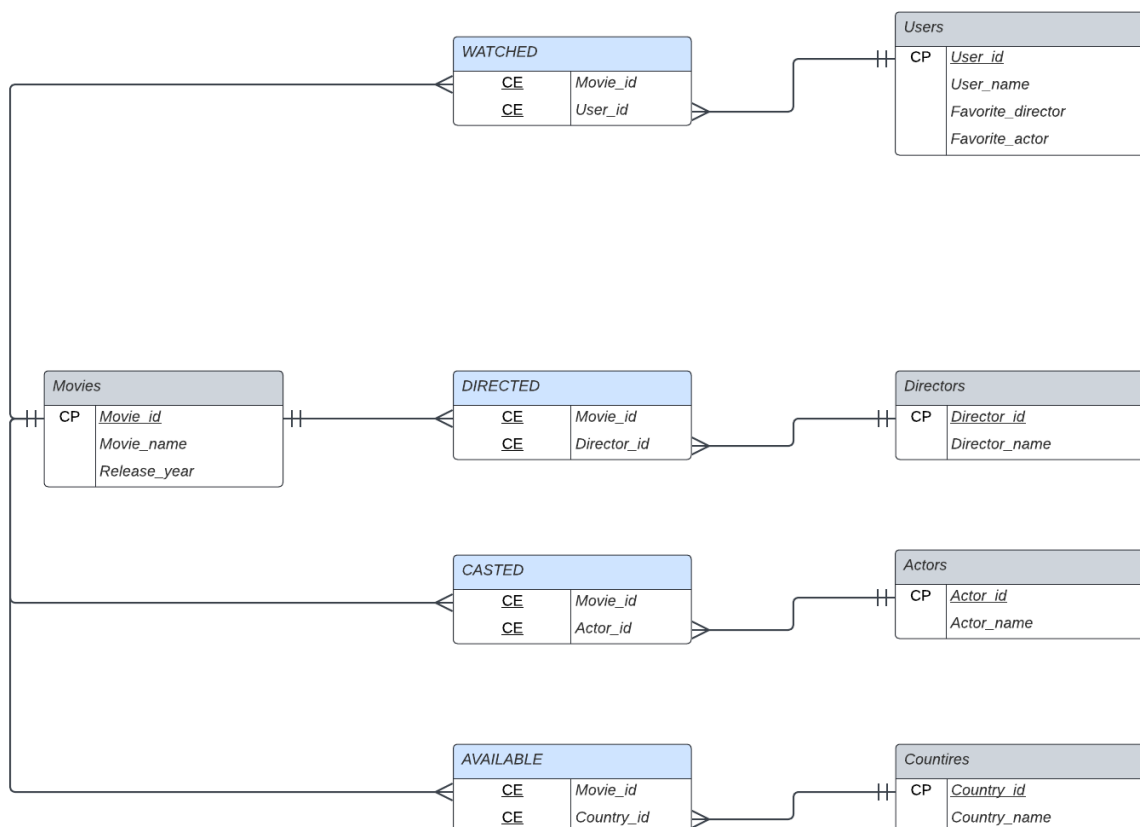


Figure 2 : Schéma d'une base de données relationnel (modélisé avec Lucidchart)

Dans ce modèle (**Figure 2**), on retrouve une représentation très similaire au modèle entité – association vu auparavant, celui-ci est plus détaillé et représente exactement le fonctionnement de cette base de données relationnelle, voici ce que l'on peut comprendre du schéma :

- CP : Clé primaire, chacune de ces entités possède une CP, celle-ci est un attribut ou un ensemble d'attributs dans une table de base de données qui garantit l'unicité des enregistrements dans cette table.
- CE : Clé étrangère, une clé étrangère est un attribut dans une table de base de données qui établit une relation entre cette table et une autre table. La clé étrangère est généralement basée sur la clé primaire d'une autre table.
- Movie_id, User_id, Director_id, Actor_id et Country_id sont tous des CP, c'est-à-dire qu'il existe une id spécifique pour chaque film, utilisateur, directeur, acteur et pays.

- Pour les tables WATCHED, DIRECTED, CASTED et AVAILABLE, leurs clés primaires sont des associations de deux CP distinctes, c'est pour cela que l'on souligne CE dans ce schéma.

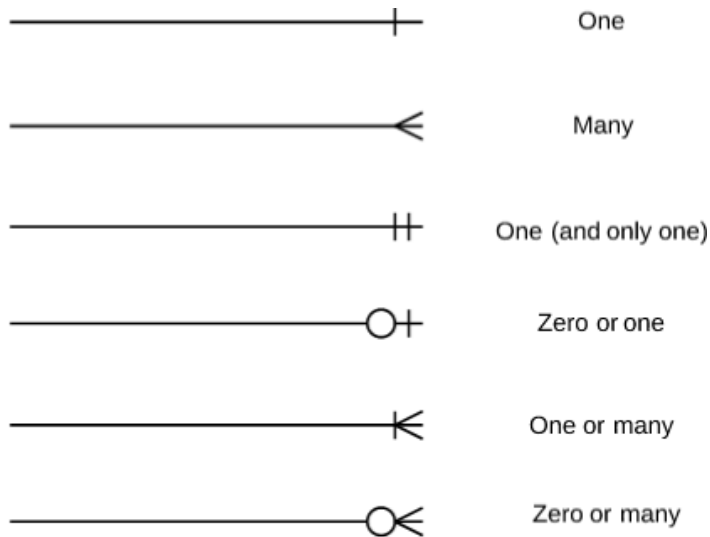


Figure 3 : Symboles des diagrammes entité-association physiques

- « La cardinalité désigne le nombre maximal de fois qu'une instance dans une entité peut être associée à des instances dans une autre entité. En revanche, l'ordinalité désigne le nombre minimum de fois qu'une instance dans une entité peut être associée à une instance de l'entité liée. » [7]

2.3 Les bases de données graphiques

Les bases de données graphiques ne sont pas de simples représentations visuelles des bases de données traditionnelles, elles constituent un type spécifique de système de gestion de base de données conçu pour le stockage et la requête efficace de données structurées sous forme de graphes. Dans ces bases, les données sont modélisées en tant que nœuds, qui représentent les entités, et en tant qu'arêtes, qui définissent les relations entre ces entités. Chaque nœud et arête peut avoir des types distincts et porter des propriétés ou attributs spécifiques, enrichissant ainsi la représentation des données.

Les nœuds sont les éléments de base de la structure de données et peuvent représenter des objets, des personnes ou tout autre élément identifiable. Les arêtes, souvent nommées par des verbes, sont, quant à elles, des liens qui connectent ces nœuds, illustrant les diverses manières dont ils peuvent interagir ou être reliés. La clé de la puissance des bases de données graphiques réside dans leur capacité à représenter et interroger des relations complexes de manière intuitive et performante. Contrairement aux bases de données relationnelles où les relations sont généralement établies via des clés étrangères et requièrent de multiples jointures, les bases de données graphiques permettent de naviguer directement entre les nœuds connectés.

Ce modèle vous permet de représenter et de stocker des données d'une manière qui reflète étroitement les relations et les connexions du monde réel.

La direction de la flèche montre comment un nœud est relié à un autre. Par exemple, dans la **Figure 4**, une flèche d'un "User1" à "Movie1" signifie que le "User1" a regardé le "Movie1", cette même flèche dans l'autre direction signifierait que le « Movie1 » a été vu par « User1 ».

La direction des flèches sert notamment faciliter la compréhension du modèle, à l'analyse de chemin et de réseau (par exemple chercher le chemin le plus court), et peut-être très important dans un contexte de réseau social, une flèche de A vers B peut indiquer que A suit B, mais pas nécessairement l'inverse.

En partant du même modèle entité association (**Figure 1**), la **Figure 4** explique le fonctionnement de cette base de données si elle était sous forme graphique.

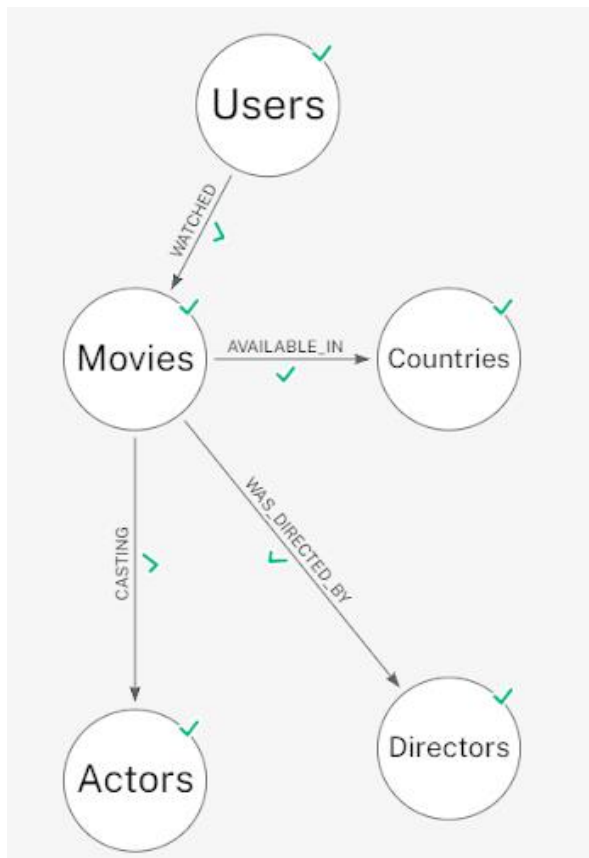


Figure 4 : Schéma d'une base de données graphique

Contrairement aux bases de données relationnelles :

- Les bases de données graphiques sont **axées sur les relations**, elles excellent dans la gestion et l'interrogation des relations entre les points de données. Elles sont donc bien adaptées aux scénarios dans lesquels la compréhension et l'interrogation des relations sont importantes, comme les réseaux sociaux, les moteurs de recommandation, la détection des fraudes et l'analyse des réseaux.
- Les bases de données graphiques sont optimisées pour **la navigation et l'interrogation des relations** dans les données. Vous pouvez facilement naviguer dans la structure du graphe pour trouver des connexions et des modèles.
- Les systèmes de gestion de bases de données de type graphe offrent souvent une **flexibilité** de schéma, ce qui vous permet d'ajouter ou de modifier la structure du graphe au fur et à mesure de l'évolution de vos données.

La **Figure 5**, représente l'exemple d'un échantillon d'une base de données graphes qui a été créée avec Neo4j à partir du schéma de la **Figure 4**, les détails de l'implémentation seront vus dans l'implémentation (Chapitre 3).

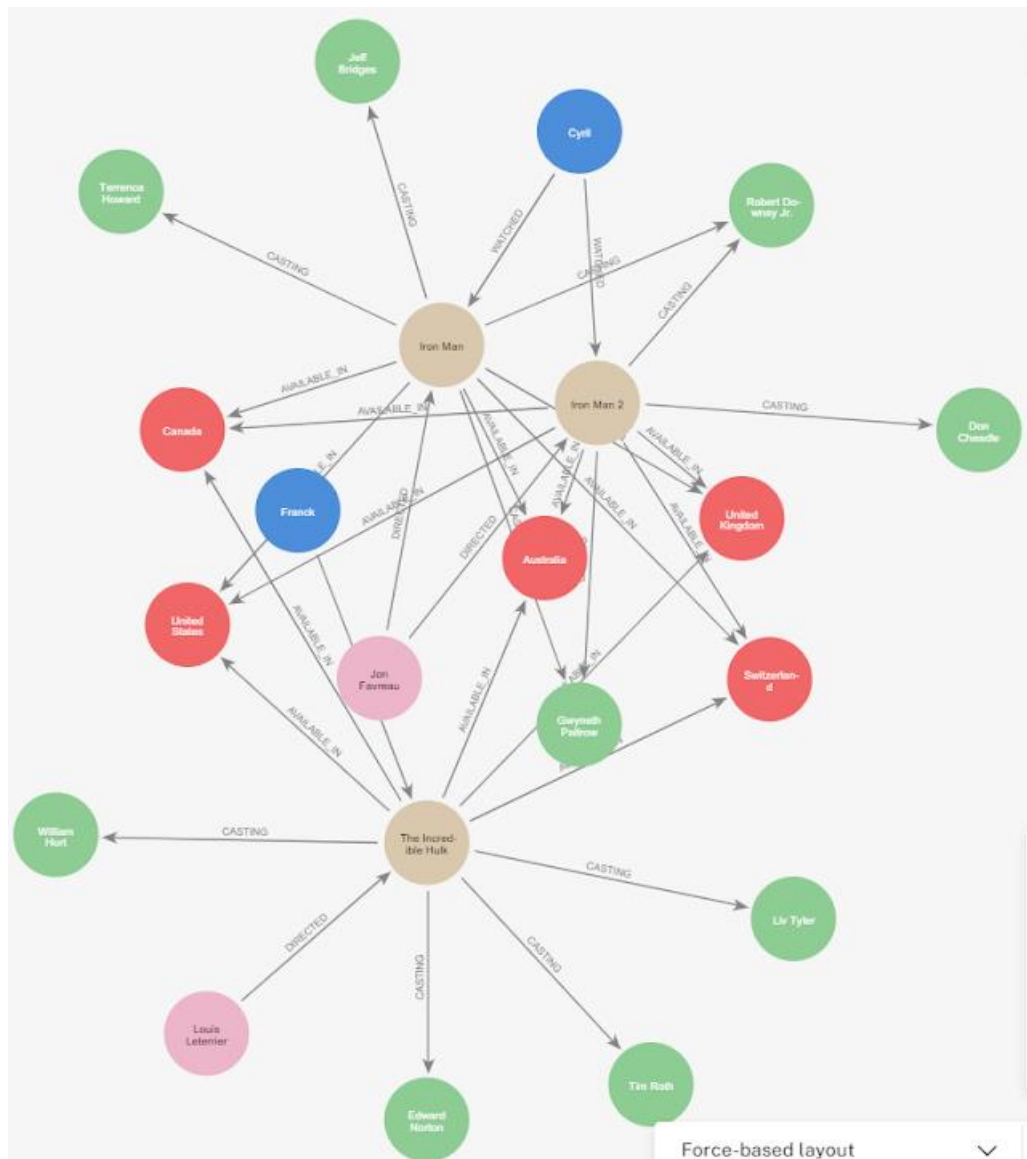


Figure 5 : Exemple d'un échantillon de base de donnée graphe créé avec Neo4j

2.4 Système de gestion des bases de données de type graphe

2.4.1 Neo4j

« Neo4j est une base de données NoSQL [8] basée sur JVM (Java virtual Machine) [9]. En tant que base de données graphique leader, son modèle est intuitif et expressif, se rapprochant du modèle de domaine sur un tableau blanc. Pour les données fortement connectées, Neo4j est des milliers de fois plus rapide que les bases de données relationnelles, ce qui en fait un choix idéal pour la gestion de données complexes dans de nombreux domaines, de la finance aux réseaux sociaux, des télécommunications à la géolocalisation. » [10]

Elle permet de modéliser des données sous forme de nœuds (entités) et de relations (liens) entre ces nœuds. Cette approche est idéale pour représenter des données interconnectées, telles que les réseaux sociaux, les recommandations de produits, les réseaux de transport, etc.

Neo4j utilise le langage de requête Cypher [11], spécialement conçu pour interroger des données graphiques. Il offre une syntaxe intuitive pour rechercher, ajouter, mettre à jour et supprimer des données dans le graphe.

Elle est optimisée pour les requêtes de graphe, ce qui en fait une solution performante, même avec des graphes massifs. Les requêtes de graphe complexes peuvent être exécutées rapidement grâce à son modèle de stockage efficace.

2.4.2 Autres SGBD

Il existe de nombreux autres systèmes de gestion de bases de données de type graphe différents de Neo4j tels que :

Amazon Neptune [13], Il s'agit d'un service de base de données de graphe entièrement géré par Amazon Web Services (AWS), compatible avec les modèles de graphe Property Graph et RDF (Resource Description Framework).

ArangoDB [14] est une base de données multi-modèles qui prend en charge les données de graphe, de document et de clé-valeur. Il offre une requête AQL (ArangoDB Query Language) puissante pour interroger les données de manière polyvalente.

Il en existe encore bien d'autres tels que **OrientDB** [15], **Dgraph** [16] ou **JanusGraph** [17], chacun de ces systèmes de gestion de base de données de graphes ont leurs propres caractéristiques, avantages et inconvénients, cependant dans le cadre de cette thèse de bachelor, nous allons uniquement utiliser neo4j.

2.5 Langage de requête NoSQL

Les langages de requête NoSQL (Not Only SQL) sont des langages de programmation conçus pour la gestion et la manipulation de données dans des bases de données non relationnelles.

Contrairement aux bases de données relationnelles traditionnelles qui utilisent SQL (Structured Query Language) [11] pour interagir avec les données, les bases de données NoSQL offrent des modèles de données plus flexibles et adaptés à diverses applications.

2.5.1 Le langage Cypher

« Cypher est un langage d'interrogation déclaratif pour les graphes de propriétés. Cypher permet d'interroger et de modifier des données, ainsi que de spécifier des définitions de schémas.

Une requête Cypher prend en entrée un graphe de propriétés et produit un tableau. Ces tableaux peuvent être considérés comme des liens pour les paramètres qui témoignent de certains modèles dans un graphe, avec un traitement supplémentaire effectué sur eux.

Cypher structure les requêtes de manière linéaire. Cela permet aux utilisateurs de penser que le traitement des requêtes commence au début du texte de la requête et progresse linéairement jusqu'à la fin. Chaque clause d'une requête est une fonction qui prend une table et produit une table qui peut à la fois augmenter le nombre de champs et ajouter de nouveaux tuples. La requête entière est alors la composition de ces fonctions. Cet ordre linéaire des clauses est purement déclaratif - les implémentations sont libres de réorganiser l'exécution des clauses si cela ne modifie pas la sémantique de la requête. Ainsi, plutôt que de déclarer la projection au début de la requête comme le fait SQL avec SELECT, dans Cypher la projection est placée à la fin de la requête en tant que RETURN.» [12]

2.5.2 Requêtes linéaires Cypher

Pour connaître les bases du fonctionnement du langage Cypher, voici les principes de base pour établir des requêtes simples en Cypher :

Syntaxe de base :

Pour créer un nœud, vous utilisez la syntaxe (n:Label {prop1: valeur1, prop2: valeur2}), où n est une variable, Label est une étiquette (optionnelle) pour le nœud, et {prop1: valeur1, prop2: valeur2} sont les propriétés du nœud.

Le « Label » est le nom attribué au nœud, « prop1 » et « prop2 » sont des noms que l'on a donnés aux propriétés 1 et 2, dans le cadre de notre exemple dans la **Figure 5**, les labels sont les noms qui sont affichés sur les nœuds, les propriétés attribuées ne sont visibles qu'en sélectionnant un nœud en particulier.

Pour créer une relation entre deux nœuds, vous utilisez la syntaxe ()-[r:RELATION]->(), où r est une variable de relation et RELATION est le type de relation (également facultatif).

Requêtes de correspondance :

Pour interroger un graphe, vous pouvez utiliser la clause MATCH. Par exemple, MATCH (n:Person) RETURN n récupère tous les nœuds avec l'étiquette "Person".

Filtres et conditions :

Vous pouvez ajouter des filtres aux requêtes à l'aide de la clause WHERE. Par exemple, MATCH (n:Person) WHERE n.age > 30 RETURN n récupère les personnes dont l'âge est supérieur à 30 ans.

Projection de données :

Utilisez la clause RETURN pour spécifier les données que vous souhaitez extraire du graphe dans le résultat de la requête. Par exemple, MATCH (n:Person) RETURN n.name renvoie les noms des personnes.

Agrégation de données :

Vous pouvez utiliser des fonctions d'agrégation telles que COUNT, SUM, AVG, etc., avec RETURN pour effectuer des calculs sur les données extraites.

Création et modification de données :

Vous pouvez utiliser les clauses CREATE, MERGE, SET, DELETE, etc., pour créer, mettre à jour ou supprimer des nœuds et des relations dans le graphe.

Tri et pagination :

Vous pouvez trier les résultats avec ORDER BY et paginer les résultats avec SKIP et LIMIT.



Figure 6 : Exemple 1 de requête Cypher dans Neo4j

Dans la requête en haut de la **Figure 6**, grâce à **WHERE**, on va demander parmi tous les nœuds dans « Movies », de nous retourner uniquement les nœuds avec « release_year » > 2020.

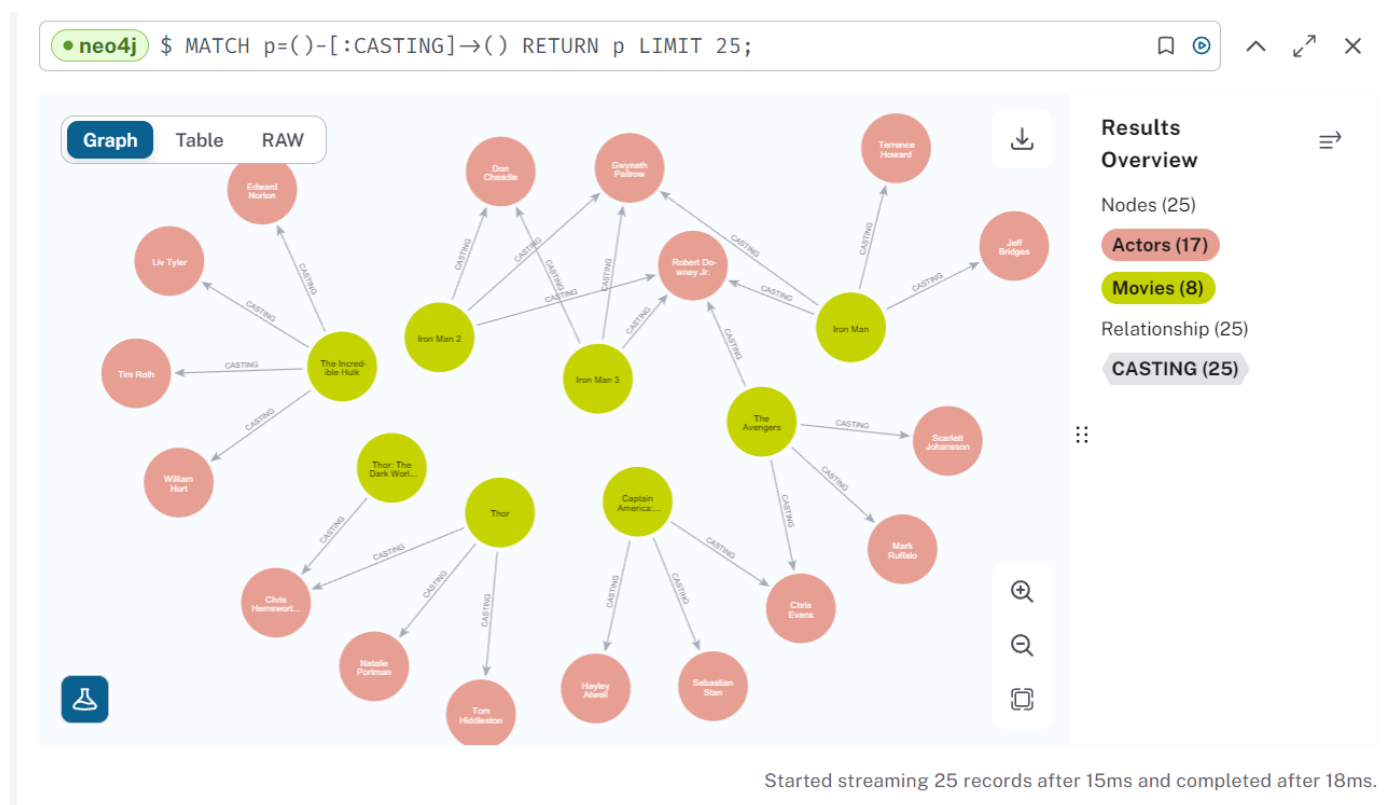


Figure 7 : Exemple 2 de requête Cypher dans Neo4j

Dans la requête en haut de la **Figure 7**, on va demander tous les nœuds avec la relation « CASTING », on va aussi limiter à 25 nœuds max pour une meilleure présentation (indiquer dans la requête avec « LIMITE 25 »).

Ceci n'est qu'une petite introduction à Cypher, on remarquera que Neo4j simplifie au maximum l'utilisation de Cypher, en proposant des fonctionnalités qui génère automatiquement du code Cypher à partir de quelque clique afin de rendre son utilisation le plus convivial que possible.

Jusque-là, rien d'extraordinaire, ce qui a été fait dans ces exemples peut aussi être fait dans un système de gestion de base de données relationnel assez facilement, nous allons maintenant procéder à une implémentation qui montrera alors réellement le potentiel de Neo4j.

3 Implémentation

Étude de cas : Les films Marvel

3.1 Génération des données

Dans cette étude de cas, nous avons utilisé une base de données générée à l'aide de ChatGPT basée sur des données réelles prises depuis le Wikipédia « Liste des films de l'univers cinématographique Marvel » [3]

Pour d'abord pouvoir comprendre cette méthode, il faut savoir comment importer des données dans Neo4j :

1^{ère} méthode :

Inscrire les données à la main grâce au code Cypher comme vu auparavant grâce à CREATE :

```
CREATE (p:Movies {movie: 'Iron man', release_year: 2008})
```

Ici, p est une variable qui représente le nœud, Movies est un label pour le nœud, et {movie: 'Iron man', release_year: 2008} sont les propriétés du nœud.

Pour créer une relation entre deux nœuds, vous devez d'abord avoir ces nœuds dans votre base de données. Par exemple, pour créer une relation WAS_DIRECTED_BY entre un film et un directeur :

```
CREATE (p:Directors {director: 'Jon Favreau'})
```

```
MATCH (a:Movies {movie: 'Iron Man'}), (b:Directors {director: 'Jon Favreau'})
```

```
CREATE (a)-[:WAS_DIRECTED_BY]->(b)
```

Ici, MATCH est utilisé pour trouver les nœuds existants, et CREATE crée une relation FRIEND de a vers b.

On peut également ajouter ou modifier des propriétés à un nœud ou une relation grâce à SET :

```
MATCH (p:Movies {movie: 'Iron Man'})
```

```
SET p.release_year = 2009
```

SET va soit ajouter la propriété release_year = 2009 au film Iron Man, soit si cette propriété existe déjà, sa valeur sera mise à jour avec la nouvelle valeur fournie.

Bien entendu, on peut aussi supprimer des nœuds et relations, grâce à DELETE :

```
MATCH (p:Movies {movie: 'Iron man'})
```

```
DETACH DELETE p
```

Pour supprimer, il suffit de MATCH le nœud ou la relation que l'on veut supprimer, puis DETACH DELETE va supprimer le nœud et toutes ses relations, pour les relations, il suffit d'utiliser DELETE uniquement.

2^{ème} méthode (méthode utilisée) :

Importer les données sous forme de fichiers CSV



```
movie.csv X
C: > Users > sonic > Desktop > Bachelor arbeit > Data > movie.csv
1  movie,release_year
2  Iron Man,2008
3  The Incredible Hulk,2008
4  Iron Man 2,2010
5  Thor,2011
6  Captain America: The First Avenger,2011
7  The Avengers,2012
8  Iron Man 3,2013
9  Thor: The Dark World,2013
10 Captain America: The Winter Soldier,2014
11 Guardians of the Galaxy,2014
12 Avengers: Age of Ultron,2015
13 Ant-Man,2015
14 Captain America: Civil War,2016
15 Doctor Strange,2016
16 Guardians of the Galaxy Vol. 2,2017
17 Spider-Man: Homecoming,2017
18 Thor: Ragnarok,2017
19 Black Panther,2018
20 Avengers: Infinity War,2018
21 Ant-Man and the Wasp,2018
22 Captain Marvel,2019
23 Avengers: Endgame,2019
24 Spider-Man: Far From Home,2019
25 Black Widow,2021
26 Shang-Chi and the Legend of the Ten Rings,2021
27 Eternals,2021
28 Spider-Man: No Way Home,2021
29 Doctor Strange in the Multiverse of Madness,2022
30 Thor: Love and Thunder,2022
31 Black Panther: Wakanda Forever,2022
32 Ant-Man and the Wasp: Quantumania,2023
33 Guardians of the Galaxy Vol. 3,2023
34 The Marvels,2023
35 Deadpool 3,2024
36 Captain America: Brave New World,2024
37 Thunderbolts,2024
38 Blade,2025
39 Fantastic Four,2025
40 Avengers: The Kang Dynasty,2026
```

Figure 8 : Données dans fichier CSV pour le nœud « Movies »

Figure 9 : Interface Neo4j : association du fichier CSV au nœud « Movies »

Neo4j propose une méthode très visuelle pour insérer les données, il suffit de :

- 1) Créer un nœud
- 2) Associer un fichier CSV à ce nœud
- 3) Créer des propriétés correspondant aux colonnes du fichier CSV (exemple **Figure 9**, on crée une propriété « movie » correspondant à la colonne « movie » du fichier CSV)

Ceci va alors créer des nœuds pour chaque nom de film dans « movie » du fichier CSV.

Pour créer des relations, c'est le même principe, mais au lieu de créer un autre nœud, on les relie avec des flèches (illustré dans la **Figure 9**) :

Figure 10 : Interface Neo4j : association du fichier CSV à la relation « CASTING »

Le fichier CSV (**Figure 11**) quant à lui est simplement l'ensemble de toutes les relations que l'on veut ajouter :



```
movie.actor.csv
C: > Users > sonic > Desktop > Bachelor arbeit > Data > movie.actor.csv
1 movie,actor
2 Iron Man,Robert Downey Jr.
3 Iron Man,Terrence Howard
4 Iron Man,Jeff Bridges
5 Iron Man,Gwyneth Paltrow
6 The Incredible Hulk,Edward Norton
7 The Incredible Hulk,Liv Tyler
8 The Incredible Hulk,Tim Roth
9 The Incredible Hulk,William Hurt
10 Iron Man 2,Robert Downey Jr.
11 Iron Man 2,Gwyneth Paltrow
12 Iron Man 2,Don Cheadle
13 Thor,Chris Hemsworth
14 Thor,Natalie Portman
15 Thor,Tom Hiddleston
16 Captain America: The First Avenger,Chris Evans
17 Captain America: The First Avenger,Hayley Atwell
18 Captain America: The First Avenger,Sebastian Stan
19 The Avengers,Robert Downey Jr.
20 The Avengers,Chris Evans
21 The Avengers,Mark Ruffalo
22 The Avengers,Scarlett Johansson
23 Iron Man 3,Robert Downey Jr.
24 Iron Man 3,Gwyneth Paltrow
25 Iron Man 3,Don Cheadle
26 Thor: The Dark World,Chris Hemsworth
27 Thor: The Dark World,Natalie Portman
28 Thor: The Dark World,Tom Hiddleston
29 Captain America: The Winter Soldier,Chris Evans
30 Captain America: The Winter Soldier,Scarlett Johansson
31 Captain America: The Winter Soldier,Sebastian Stan
32 Guardians of the Galaxy,Chris Pratt
33 Guardians of the Galaxy,Zoe Saldana
34 Guardians of the Galaxy,Dave Bautista
35 Guardians of the Galaxy,Vin Diesel (voice of Groot)
36 Avengers: Age of Ultron,Robert Downey Jr.
37 Avengers: Age of Ultron,Chris Evans
38 Avengers: Age of Ultron,Mark Ruffalo
39 Avengers: Age of Ultron,Scarlett Johansson
40 Ant-Man,Paul Rudd
41 Ant-Man,Michael Douglas
42 Ant-Man,Evangeline Lilly
43 Ant-Man,Corey Stoll
44 Captain America: Civil War,Chris Evans
45 Captain America: Civil War,Robert Downey Jr.
46 Captain America: Civil War,Scarlett Johansson
47 Captain America: Civil War,Sebastian Stan
```

Figure 11 : Données dans le fichier CSV pour la relation « CASTING »

Ajouter les données devient alors très facile et rapide.

Pour mon implémentation, il m'a fallu une certaine quantité de données à utiliser pour pouvoir tester l'ensemble des fonctionnalités de Neo4j. Pour cela, j'ai fait appel à ChatGPT 3.5 pour m'aider à générer une quantité suffisante de données.

Voici le processus :

- Copier-coller la page Wikipédia Marvel à ChatGPT
- J'ai ensuite moi-même rédigé une partie de l'échantillon de données fourni par Wikipédia en style fichier CSV, j'ai ensuite demandé à ChatGPT de continuer cette liste dans la **Figure 12**.

can you complete the csv file below like this with the rest of the movies :

```
movie,director
Iron man, Jon Favreau
The Incredible Hulk, Louis Leterrier
Iron Man 2, Jon Favreau
```



ChatGPT

Certainly, I can help you complete the CSV file with the information you provided for the Marvel Cinematic Universe films. Here's the complete CSV file with the rest of the movies:

```
CSV Copy code
movie,director
Iron Man,Jon Favreau
The Incredible Hulk,Louis Leterrier
Iron Man 2,Jon Favreau
Thor,Kenneth Branagh
Captain America: The First Avenger,Joe Johnston
The Avengers,Joss Whedon
Iron Man 3,Shane Black
Thor: The Dark World,Alan Taylor
Captain America: The Winter Soldier,Anthony and Joe Russo
Guardians of the Galaxy,James Gunn
Avengers: Age of Ultron,Joss Whedon
Ant-Man,Peyton Reed
Captain America: Civil War,Anthony and Joe Russo
Doctor Strange,Scott Derrickson
Guardians of the Galaxy Vol. 2,James Gunn
Spider-Man: Homecoming,Jon Watts
Thor: Ragnarok,Taika Waititi
```

Figure 12 : Création des données avec ChatGPT

Le processus a été le même pour tous les autres nœuds tels que actors, countries, directors, et toutes les relations que l'on voit apparaître sur la **Figure 8**, ce qui a rendu la génération de données à tester rapide et efficace.

3.2 Importation des données

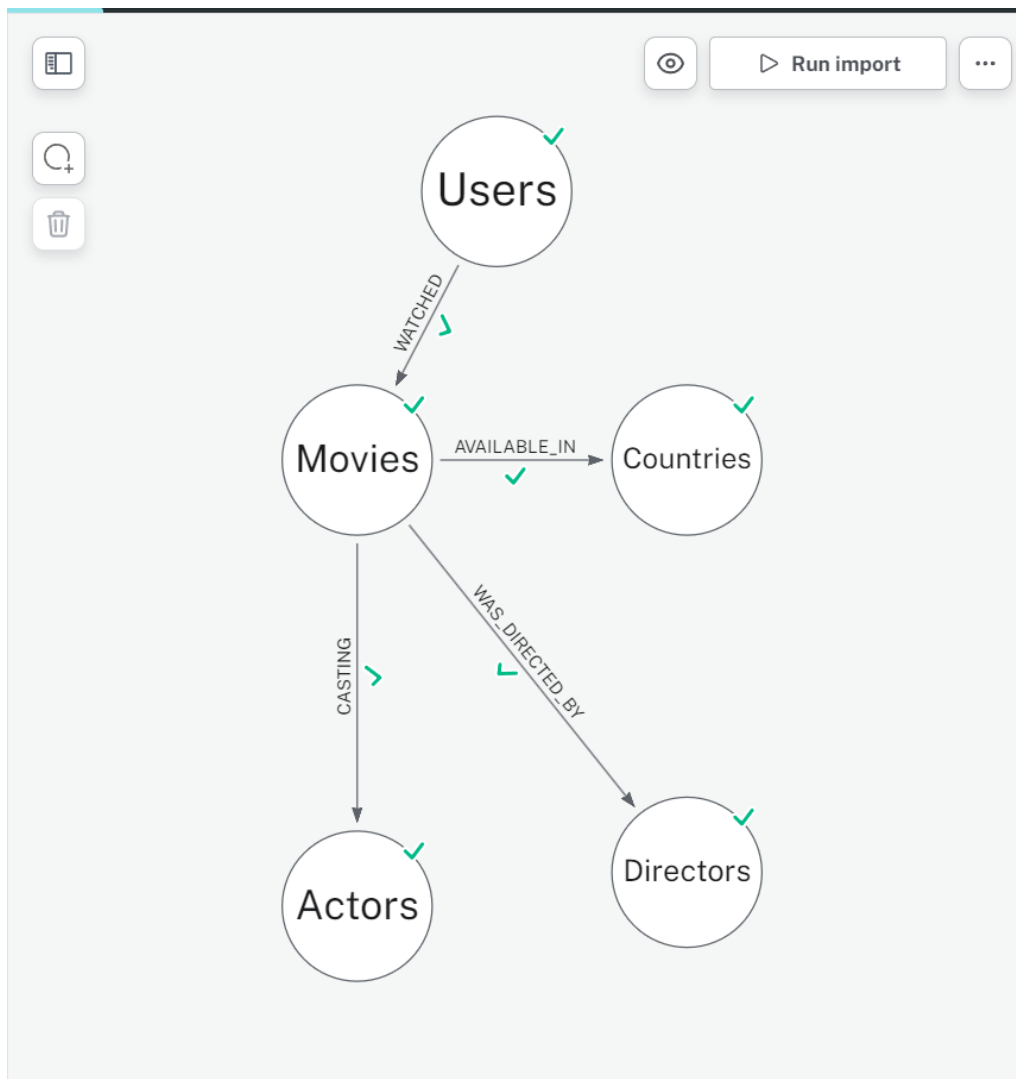


Figure 13 : Schéma final créé sur Neo4j avant de confirmer l'importation (même que Figure 4)

La **Figure 13** est le schéma final utilisé pour l'importation des données, voici à quoi chaque nœud fait référence :

- Movies : Movies est la liste de tous les films Marvel, contenant les propriétés `release_year`, et `movie` (nom du film)
- Actors : Actors est la liste de tous les acteurs Marvel
- Directors : Directors est la liste de tous les directeurs des films Marvel
- Countries : Countries est la liste de tous les pays où les films Marvel sont disponibles (subjective, ne correspond pas à la réalité)
- Users : Users est une liste d'utilisateurs qui regardent un certain nombre de films Marvel, chaque User contient la propriété `user` (nom de l'utilisateur).

3.3 Résultat et exploration

Maintenant que tout est prêt, nous pouvons tout importer et générer notre base de données graphique avec Neo4j :

La **Figure 14** est notre résultat généré par Neo4j avec la totalité de nos données importées.

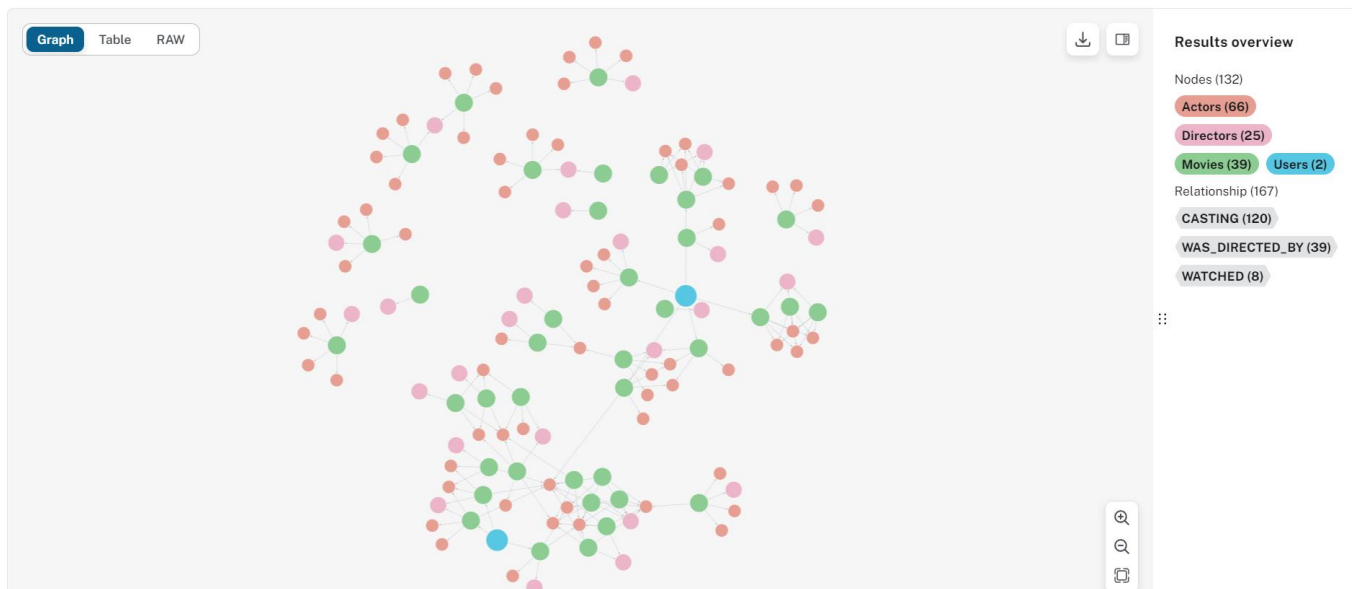


Figure 14 : Graphe généré par Neo4j avec la totalité de nos données

La **Figure 14** permet un aperçu global de notre base de données, Neo4j nous permet de zoomer et d'explorer cette base de données pour pouvoir l'analyser en détails.

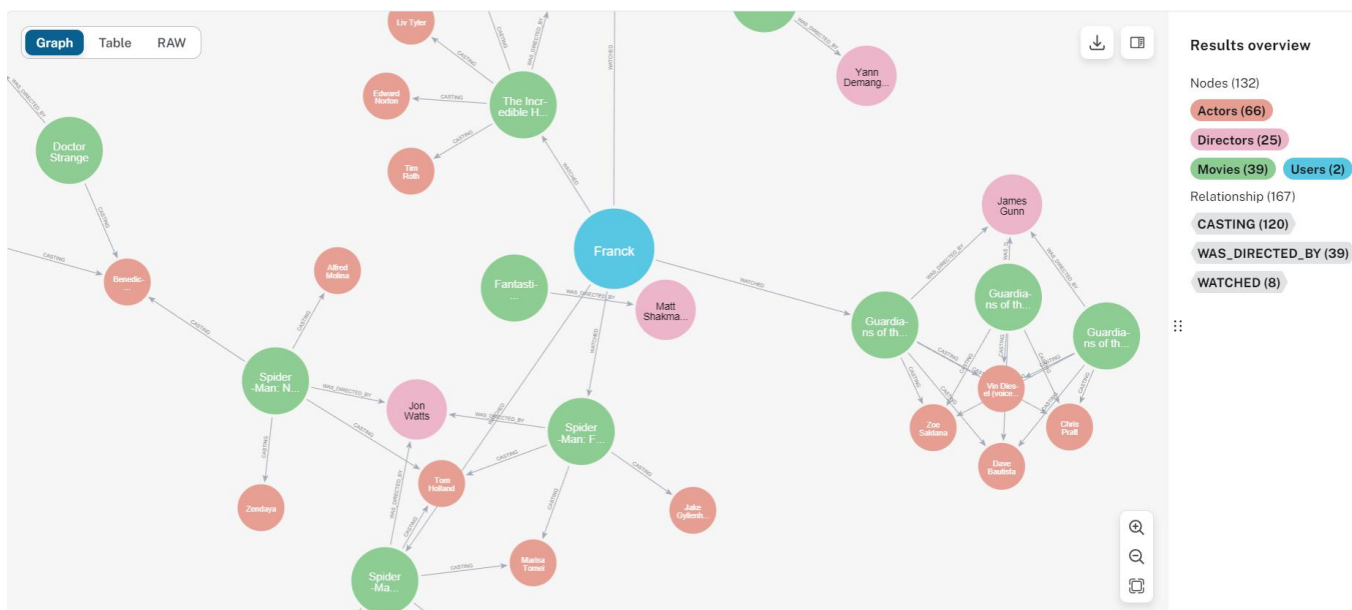


Figure 15 : Graphe généré par Neo4j avec la totalité de nos données (zoomé)

Dans la **Figure 15**, nous avons effectué un zoom et nous pouvons constater toutes les relations entre les nœuds.

Ce graphe enveloppe la globalité des données que l'on a importées, l'exploration d'une base de données graphe dans son intégralité présente plusieurs avantages, mais elle comporte également des inconvénients potentiels.

Avantages :

- Permet une compréhension approfondie de la structure de données
- Permet la découverte de nouvelles informations
- Permet la détection d'anomalies (nœuds ou relations inhabituelles)

Désavantages :

- Coût en temps et en ressources
- Surcharge d'informations

L'exploration complète d'une base de données graphe sans aucun filtre peut être bénéfique, mais ne peut pas être utilisée sur une quantité trop importante d'information.

3.4 Analyse avec Query Neo4j

Neo4j propose une grande quantité de fonctionnalités afin de permettre une analyse plus approfondie des bases de données graphes.

Voici quelques exemples d'utilisation :

Figure 16 : Interface Query de Neo4j

Sur la **Figure 16**, on a simplement cliqué sur « Movies », ce qui fait apparaître un échantillon de 25 films (automatiquement limité par Neo4j, peut-être changé manuellement en augmentant « LIMIT » dans la requête) de la base de données, on peut faire la même chose avec n'importe

quel autre nœud ou relation que l'on trouve dans notre base de données, si on souhaite avoir un échantillon plus grand, il suffit de monter la limite.

The screenshot displays the Neo4j Query interface. On the left, the 'Database information' panel shows 140 nodes and 362 relationships. The 'CASTING' relationship is highlighted with a red circle and arrow. The 'Property keys' section lists various attributes like 'actor', 'country', 'data', 'director', 'id', 'movie', 'name', 'nodes', 'relationships', 'release_year', 'style', 'user', and 'visualisation'. The main query area shows the Cypher query: `neo4j $ MATCH p()-[:CASTING]->() RETURN p LIMIT 25;`. The results are displayed as a graph with 25 nodes and 25 relationships. The 'Results overview' panel on the right indicates 25 nodes (17 Actors, 8 Movies) and 25 relationships (CASTING).

Figure 17 : Interface Query de Neo4j (2)

Même principe pour la Figure 17, ici, on affiche un échantillon de 25 nœuds portant la relation « CASTING ».

Ces fonctionnalités vont simplement créer du code Cypher et effectuer une requête, Neo4j propose aussi une partie « Explore » qui nous permettra d'effectuer une recherche personnalisée.

3.5 Analyse avec « Explore » de Neo4j

« Explore » nous permet de créer une scène où vont s'ajouter au fur et à mesure les données que l'on souhaite afin de pouvoir y effectuer une analyse.

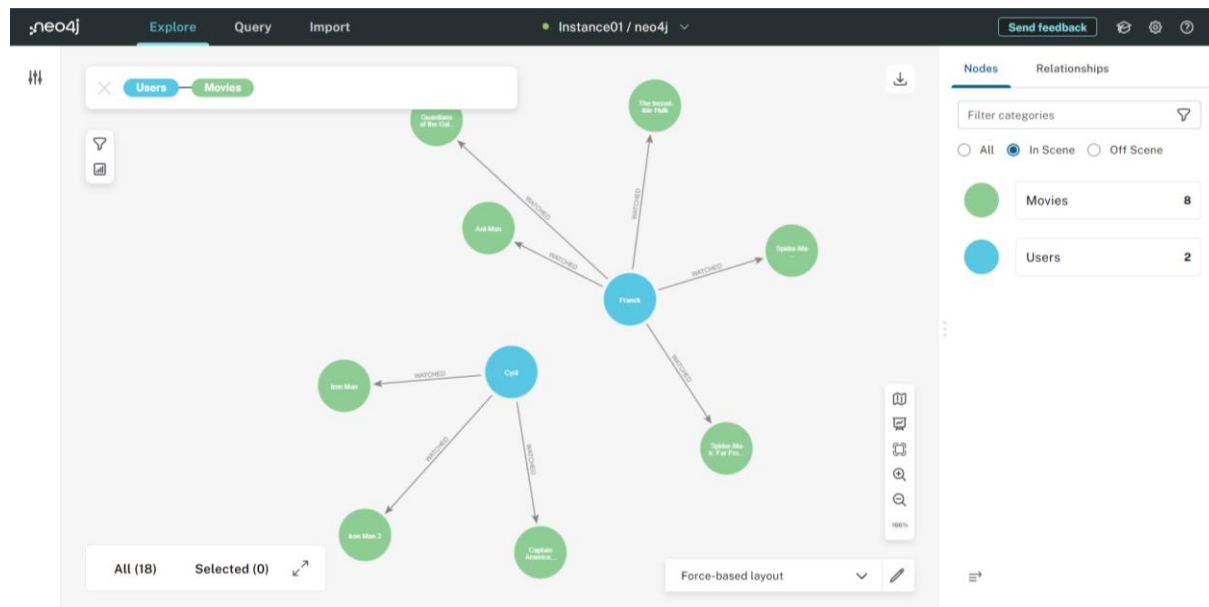


Figure 18 : Interface Explore de Neo4j, création d'une scène

Par exemple, dans la Figure 18, on décide d'abord d'ajouter tous les « Users » et leurs relations avec les « Movies ».

Maintenant, imaginons que l'on aimerait pouvoir recommander des films pour l'utilisateur « Cyril », Cyril a beaucoup aimé le film « Captain America », on va alors utiliser la fonction « expand » afin de voir tous les autres nœuds liés au nœud « Captain America » (illustré dans la Figure 19)

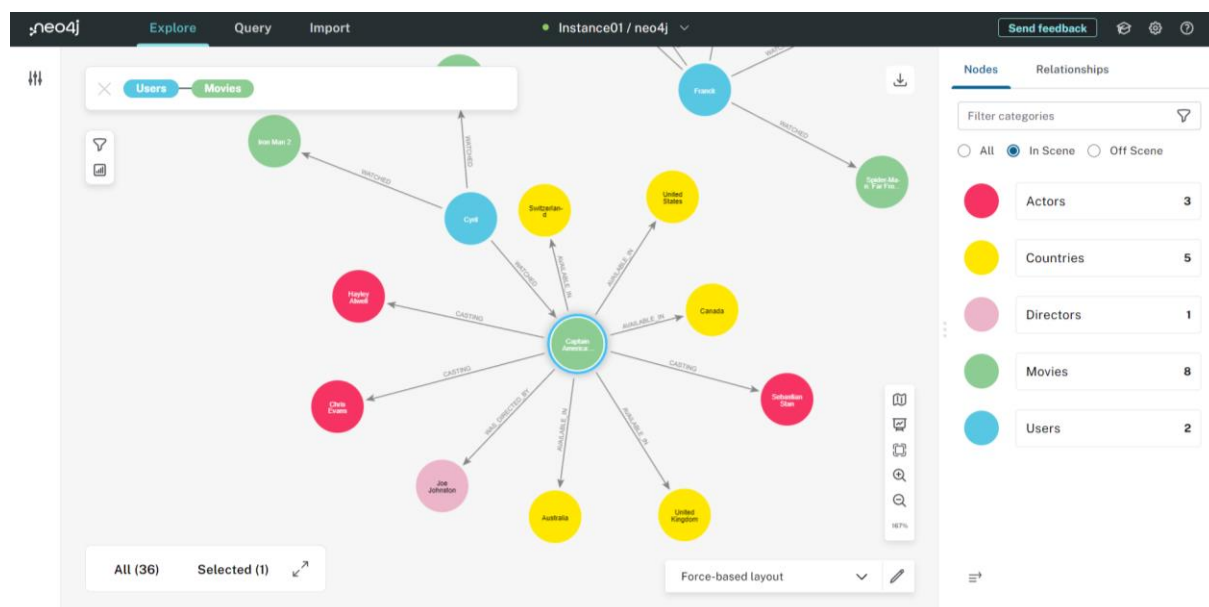


Figure 19 : Interface Explore de Neo4j, exploration des données

On remarque dans la **Figure 19**, 3 acteurs principaux : Chris Evans, Hayley Atwell et Sebastian Stan. Maintenant, allons voir tous les autres films joués par Chris Evans, pour cela, on va « expand » une nouvelle fois (illustré dans la **Figure 20**) :

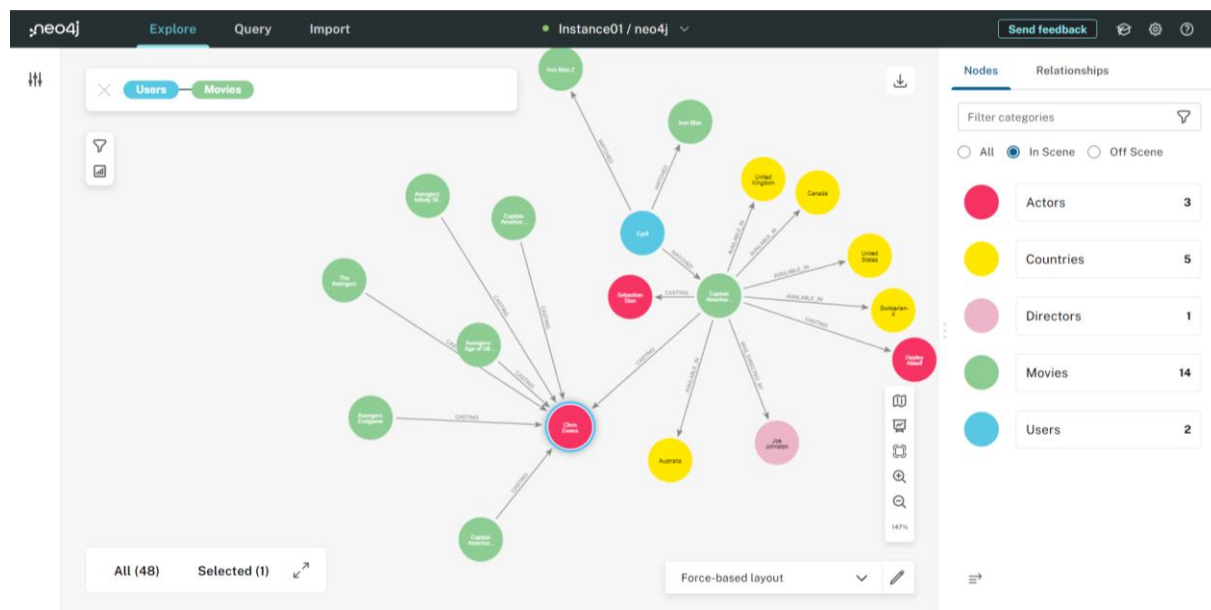


Figure 20 : Interface Explore de Neo4j, exploration des données (2)

On remarque alors dans la **Figure 20**, cette fois que l’acteur Chris Evans a aussi joué dans les films « The Avengers », « Avengers Endgame » etc (tous les nouveaux nœuds en vert), tous ces films pourraient potentiellement plaire à l’utilisateur Cyril !

On peut utiliser ce même principe dans plein d’autres exemples :

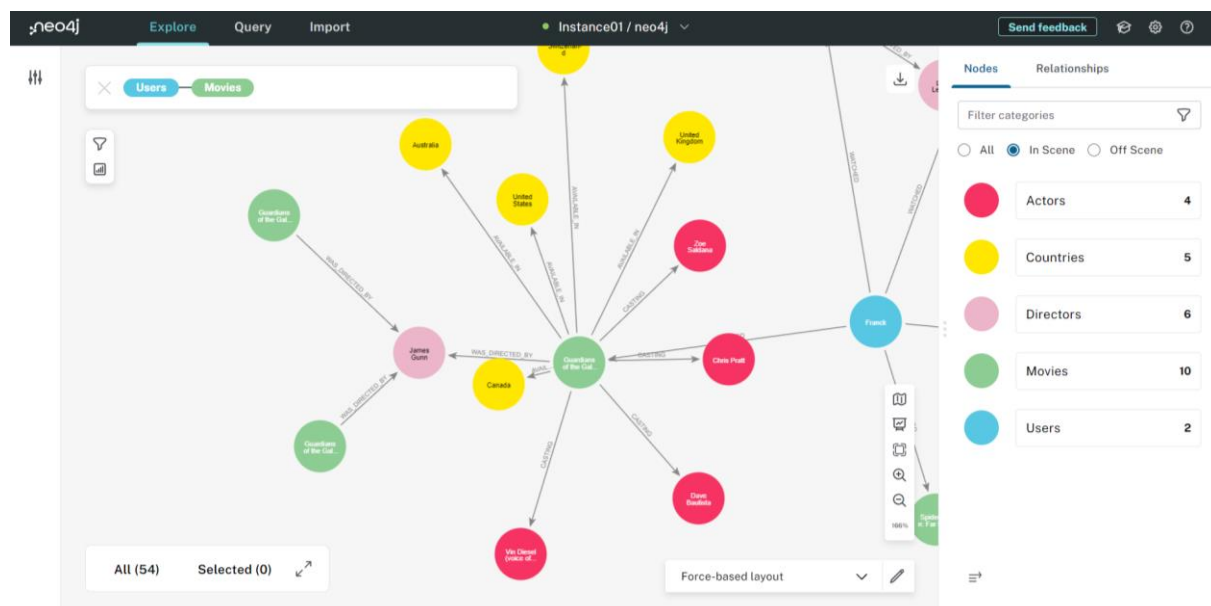


Figure 21 : Interface Explore de Neo4j, exploration des données (3)

L’utilisateur « Franck » a regardé le film « Guardians of the Galaxy 1 » réalisé par le directeur « James Gunn », celui-ci a aussi réalisé « Guardians of the Galaxy 2 » et « Guardians of the Galaxy 3 » (nœuds en vert) !

Il est même possible de partir directement d'un nœud en particulier et d'explorer à partir de celui-ci (illustré dans la **Figure 22**)

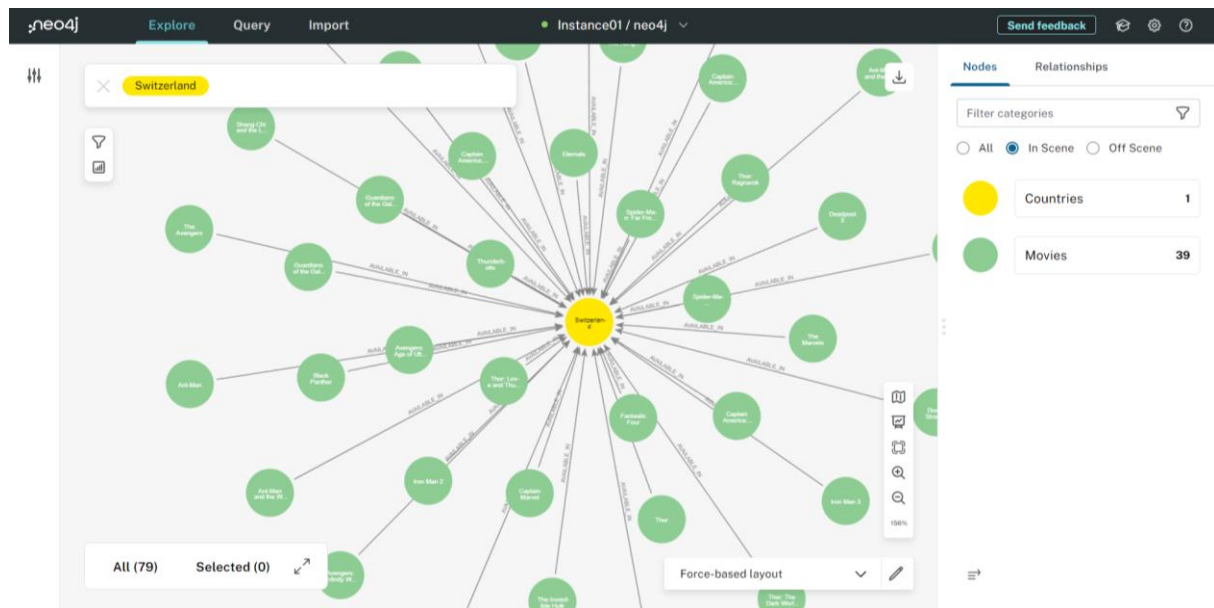


Figure 22 : Interface Explore de Neo4j, exploration des données (4)

Dans la **Figure 22**, on a simplement recherché « Switzerland » et utilisé « expand » et on trouve tous les films disponibles en Suisse !

3.6 Récapitulatif de Neo4j

La réalisation d'un niveau d'exploration approfondi tel que celui offert par Neo4j semble tout simplement inimaginable au sein d'un système de gestion de base de données relationnelle traditionnel. Non seulement le processus nécessiterait considérablement plus de temps, mais il exigerait également un niveau de compétence bien plus élevé. Neo4j se démarque en proposant une solution à la fois simple et efficace, permettant une exploration rapide des données. Cette différence cruciale souligne l'importance de choisir des outils adaptés aux besoins spécifiques, où la simplicité et l'efficacité de Neo4j offrent une alternative remarquablement avantageuse par rapport aux méthodes plus complexes et chronophages des systèmes de gestion de bases de données relationnelles traditionnels. En définitive, opter pour des solutions novatrices comme Neo4j représente un choix judicieux pour ceux qui cherchent à maximiser leur efficacité dans l'exploration et la gestion des données.

4 Conclusion

4.1 Conclusion

En conclusion, notre objectif principal était d'explorer et de comparer les nuances et les capacités des systèmes de gestion de bases de données graphiques par rapport aux systèmes de gestion de bases de données relationnelles. Pour ce faire, nous avons initié notre étude par une explication détaillée des concepts fondamentaux, suivie d'une analyse comparative mettant en lumière les avantages et les inconvénients respectifs de chaque système. Nous avons ensuite illustré concrètement ces différences en schématisant le fonctionnement des deux types de bases de données. Enfin, une étude de cas concrète impliquant l'utilisation de Neo4j a été réalisée pour mettre en pratique et visualiser les concepts abordés. Cette exploration approfondie nous a permis de mettre en évidence le potentiel significatif des SGBD de type graphe, particulièrement dans le contexte actuel où les données deviennent de plus en plus interconnectées et complexes.

Pour l'avenir, nous pourrions étendre notre recherche en développant une application de recommandation basée sur notre implémentation actuelle dans Neo4j. Cette application servirait à démontrer l'utilité pratique des SGBD graphiques dans des scénarios du monde réel. De plus, nous pourrions enrichir notre base de données en intégrant un éventail plus large de films, en incluant, par exemple, les univers de DC Comics en plus des films Marvel. Cette expansion nous permettrait d'explorer les connexions fascinantes et peut-être inattendues entre ces deux univers cinématographiques.

References

- [1] Wikipédia. https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es
- [2] Neo4j. <https://neo4j.com/>
- [3] Wikipédia. https://fr.wikipedia.org/wiki/Liste_des_films_de_l%27univers_cin%C3%A9matographique_Marvel
- [4] Wikipédia. https://fr.wikipedia.org/wiki/Mod%C3%A8le_entit%C3%A9-association
- [5] Lucidchart. <https://www.lucidchart.com/pages/fr/diagramme-entite-association>
- [6] Lucidchart. <https://lucid.app/>
- [7] Lucidchart. <https://www.lucidchart.com/pages/fr/symboles-et-notation-des-diagrammes-entite-association>
- [8] Wikipédia. <https://fr.wikipedia.org/wiki/NoSQL>
- [9] Wikipédia. https://en.wikipedia.org/wiki/Java_virtual_machine
- [10] ACM Digital Library. <https://dl.acm.org/doi/10.1145/2384716.2384777> (Dernière visite. Décembre, 2023).
- [11] Wikipédia. https://fr.wikipedia.org/wiki/Structured_Query_Language
- [12] ACM Digital Library. <https://dl.acm.org/doi/pdf/10.1145/3183713.3190657> (Dernière visite. Décembre, 2023)
- [13] Amazon Neptune. <https://aws.amazon.com/fr/neptune/>
- [14] ArangoDB. <https://arangodb.com/>
- [15] OrientDB. <https://orientdb.org/docs/3.0.x/misc/Overview.html>
- [16] Dgraph. <https://dgraph.io/>
- [17] JanusGraph. <https://janusgraph.org/>
- [18] ChatGPT. <https://chat.openai.com/>
- [19] Neo4j Aura. <https://console.neo4j.io/>