

Table des matières

Introduction.....	2
1 – Structure & données	3
1.1 - Structure de la base.....	3
1.2 – Type de données	4
1.3 – Schéma & modélisation conceptuelle	4
2 – Architecture technique.....	5
2.1 – Plateformes de déploiement et de développement	5
2.2 – APIs	5
2.3 – Patterns	6
3 – Usages.....	7
3.1 – Exemples de cas d'utilisations concrets	7
3.2 – Positionnement et tendance sur le marché des bases de données.....	9
3.3 – Analyse sur les usages	11
4 – Démonstration & synthèse.....	12
4.1 – Mise en place de l'architecture	12
4.2 – Dataset utilisé.....	13
4.3 – Création du graphe.....	14
4.3.1 – Configuration de NEO4J.....	14
4.3.2 – Création des nœuds.....	15
4.3.3 – Création des relations entre les nœuds.....	16
4.3.4 – La recommandation de bières.....	18
Glossaire	20

Introduction

Ce rapport présente de manière globale un système de base de données, Neo4j.

Neo4j est un système de gestion de bases de données orienté graphes qui existe depuis 2007, c'est-à-dire que l'information stocké dans une base se fait sous la forme de nœuds et de relations entre les nœuds. Une multitude de langages sont compatibles pour gérer une base et plusieurs API sont disponible rendant la technologie particulièrement intéressante. Les requêtes se font en Cypher, un langage proche du SQL.

Nous y verrons les spécificités de cette base, la structure générale ainsi que les cas d'usages préconisés pour ce type de base. Nous mettrons d'ailleurs en place une démonstration sur une machine virtuelle (VM) pour mieux expliquer les avantages du système.

Ce rapport contient quatre grandes parties : Structure & données, Architecture technique, Usages et Démonstration et synthèse.

1 – Structure & données

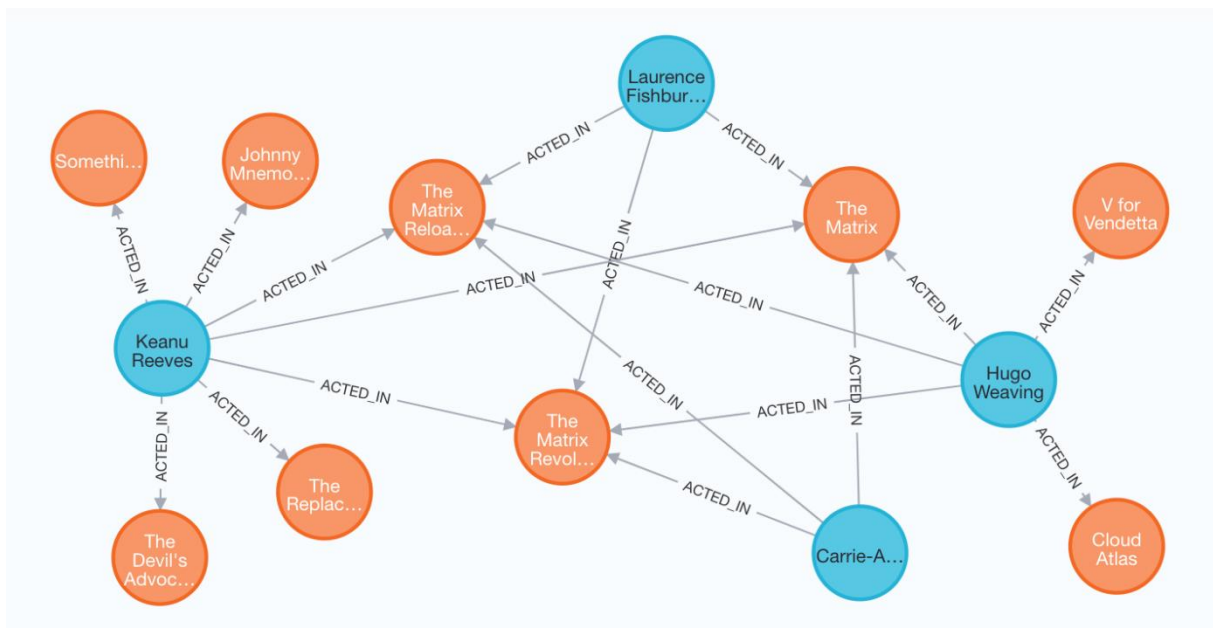
1.1 - Structure de la base

Le système Neo4j est orienté graphes, c'est-à-dire que l'information stocké se fait sous la forme de nœuds et de relations, sans passer par des tables.

Les nœuds correspondent à un objet, cela peut être par exemple une personne, un livre, un lieu, etc..., avec un ensemble d'attributs.

Les relations permettent de faire le lien entre les éléments d'une base et peuvent servir à porter une information comme une note, une distance ou encore une appartenance.

Le meilleurs modèle pour représenter ce type de système est le modèle entité-association (EA).



Exemple d'un graphe

Sur l'image ci-dessus nous pouvons voir les nœuds qui sont les ronds oranges et bleus, correspondant respectivement aux films et acteurs, et les relations dessinées par des flèches allant d'un nœud à un autre pour montrer dans quel films les acteurs ont joués.

1.2 – Type de données

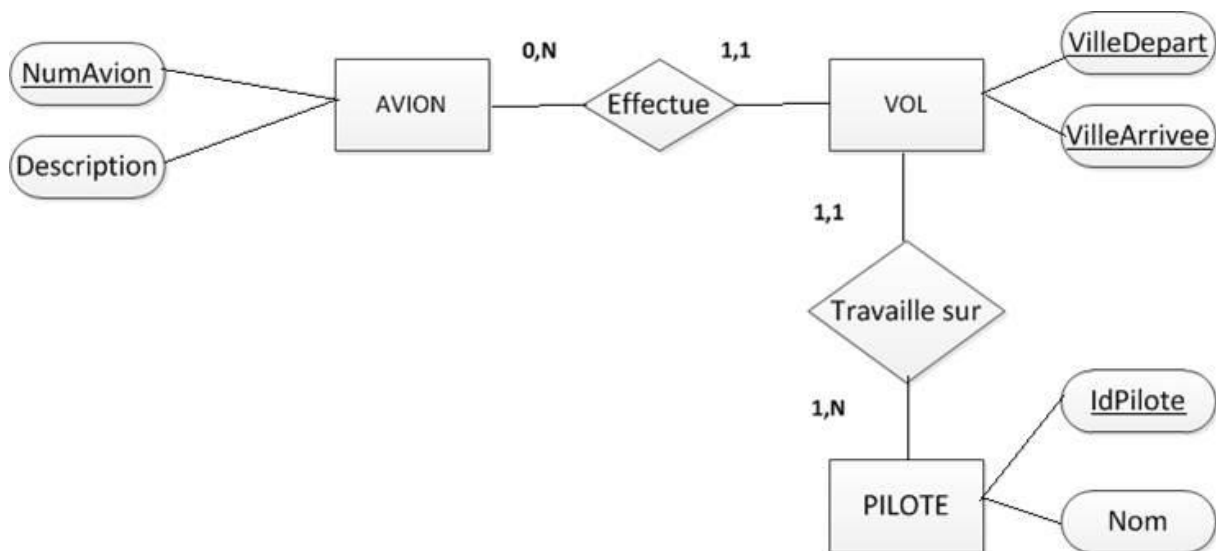
Neo4j prend en entrée des données structurées au format CSV ou JSON. Une donnée structurée est une donnée qui est formatée avant d'arriver dans une base de données, avec des champs et une structure précise. L'avantage est de faciliter les traitements puisque toutes les entités auront les mêmes caractéristiques.

Les bases sont également compatibles ACID, ce qui signifie qu'elles ont un ensemble de transactions qui garantissent la validité des données présentes dans une base de données.

1.3 – Schéma & modélisation conceptuelle

Les données importées dans une base de données Neo4j doivent respecter un schéma bien défini qui découle directement de la spécificité de la base, les graphes. En effet, les graphes suivent des règles spécifiques, il faut des nœuds, des relations et des propriétés.

Il est donc naturel d'utiliser un schéma entité-association pour modéliser en amont la base de données pour être efficace lors de l'import et de la création du graphe.



Exemple d'un schéma entité-association

Sur ce schéma nous pouvons voir les nœuds avec les rectangles et les relations avec les losanges, les nœuds possèdent des attributs et on peut ajouter des relations entre les nœuds. C'est une représentation qui s'inscrit parfaitement dans la théorie des graphes et donc dans l'environnement Neo4j.

2 – Architecture technique

2.1 – Plateformes de déploiement et de développement

Neo4j dispose de plusieurs plateformes pour aider l'installation et la gestion des bases de données, Nous pouvons citer Neo4j Desktop qui est un client lourd qui permet facilement d'organiser les différents projet, de se connecter à une base de données locale où même distante. Toutes les options de configurations basiques sont accessibles et avec des plugins supplémentaires comme APOC ou Graph Data Science cela en fait un outil tout en un puissant. Il est également possible d'importer les données et d'effectuer des requêtes directement depuis cet environnement.

Une autre solution existe depuis le 13 avril 2022, Neo4j Aura. C'est une plateforme évolutive et rapide, toujours disponible et entièrement automatisée sous la forme d'un service cloud. Ce système permet de gagner beaucoup de temps sur la partie gestion et technique pour se concentrer sur la valeur ajoutée et l'innovation. Cette solution comprend AuraDB, une base de données de graphes cloud ainsi qu'AuraDS, une plateforme de de sciences des données de graphes, destinée aux data scientists pour élaborer des modèles prédictifs.

Tout ces outils sont disponibles sur OS, Linux et Windows.

2.2 – APIs

Neo4j étant un système qui a déjà 15ans d'expériences, de nombreuses APIs ont vu le jour via des bibliothèques et des packages. Voici une liste de langages pris en charge par Neo4j :

- Python
- JavaScript
- GraphQL
- Java
- Spring Boot
- .NET
- Go

Tout ces langages peuvent être utilisé pour discuter avec des bases Neo4j et peuvent créer des requêtes Cypher, le langage par défaut pour fabriquer des requêtes et construire une base de données.

2.3 – Patterns







D'un point de vue réplication et tolérance aux pannes, ce système est excellent, en effet Neo4j utilise une structure de cluster maître-esclave pour garantir qu'une réplique complète du graphe est stockée sur chaque machine. Les écritures sont répliquées à partir du maître vers les esclaves à des intervalles fréquents, ce qui veut dire qu'à chaque instant au moins le maître et plusieurs esclaves ont une copie conforme du graphe.

Si il faut dépasser la capacité d'un cluster, il est possible de partitionner le graphe sur plusieurs instances via la construction d'une logique de sharding dans l'application. Il faudra dans ce cas utiliser un identifiant synthétique pour la jointure des données. Tout les graphes ne sont pas fait pour le partitionnement donc c'est un aspect à prendre en compte en amont au niveau de la conception pour anticiper au moment de l'implémentation.

3 – Usages

3.1 – Exemples de cas d'utilisations concrets

Ci-dessous quelques exemples des principaux axes où Neo4j est le plus pertinent :

 Détection et analyse des fraudes L'analyse en temps réel des relations entre les données est essentielle pour découvrir les réseaux de fraude et autres escroqueries sophistiquées avant que les fraudeurs et les criminels ne causent des dommages durables. Voir Cas d'utilisation →	 Surveillance de l'infrastructure réseau et base de données pour les opérations informatiques Les bases de données de graphes sont intrinsèquement plus adaptées que le RDBMS pour donner un sens aux interdépendances complexes essentielles à la gestion des réseaux et de l'infrastructure informatique. Voir Cas d'utilisation →	 Moteur de recommandation et système de recommandation de produits Les moteurs de recommandation basés sur des graphiques aident les entreprises à personnaliser leurs produits, contenus et services en exploitant une multitude de connexions en temps réel. Voir Cas d'utilisation →
 Gestion des données de référence Organisez et gérez vos données de base avec le modèle de base de données graphique flexible et sans schéma afin d'obtenir des informations en temps réel et une vue à 360° de vos clients. Voir Cas d'utilisation →	 Graphiques des médias sociaux et des réseaux sociaux Exploitez facilement les connexions sociales ou inférez des relations basées sur l'activité lorsque vous utilisez une base de données graphique pour alimenter votre application de réseau social. Voir Cas d'utilisation →	 Gestion des identités et des accès Suivez rapidement et efficacement les utilisateurs, les actifs, les relations et les autorisations lorsque vous utilisez une base de données graphique pour la gestion des identités et des accès. Voir Cas d'utilisation →

Sur ces différents points nous pouvons noter que c'est dans des circonstances où l'on cherche des liens, des rapprochements entre des points, par exemple des algorithmes de recommandations sont très efficaces à l'aide de graphes puisqu'il suffit d'utiliser des algorithmes de plus court chemin comme Dijkstra. Sur les fraudes cela peut montrer les personnes qui sont liées à beaucoup de fraudes pour déterminer les zones sensibles.

Ci-dessous des exemples où Neo4j a été utilisé par des entreprises :






3.2 – Positionnement et tendance sur le marché des bases de données

Avant de lancer ce projet, nous n'avions que peu d'expérience sur les bases de données de graphes et en effectuant des recherches nous avons remarqué qu'il existe en réalité une multitude de modèles différents, Wikipédia en recense 18.

Name	Language(s)	Notes
AllegroGraph	SPARQL	RDF triplestore
ArangoDB	AQL, JavaScript, GraphQL	Base de données orientée documents, Base de données orientée graphe et système clé-valeur
DEX/Sparksee	C++, Java, .NET, Python	Base de données orientée graphe
DataStax Enterprise Graph	Java, C#, C++, Python, Node.js, Gremlin (Apache Tinkerpop)	Base de données distribuées s'appuyant sur Apache Cassandra et intégrant un moteur graphe inspiré par TitanDB.
Dgraph	Go, Javascript, Python, GraphQL	Base de données orientée graphe avec un support natif de GraphQL.
FlockDB	Scala	Base de données orientée graphe
IBM DB2	SPARQL	RDF triplestore depuis la DB2 10
InfiniteGraph	Java	Base de données orientée graphe
JanusGraph	Java, Gremlin (Apache Tinkerpop)	Base de données orientée graphe
MarkLogic	Java, JavaScript, SPARQL, XQuery	Base de données multi-modèle et RDF triplestore
Neo4j	Cypher, Gremlin (Apache Tinkerpop)	Base de données orientée graphe
OpenLink Virtuoso	C++, C#, Java, SPARQL	Middleware et RDF triplestore
Oracle	SPARQL 1.1	RDF triplestore
OrientDB	Java, Structured Query Language, Gremlin (Apache Tinkerpop)	Base de données multi-modèle, Base de données orientée documents et base de données orientée graphe.
OWLIM	Java, SPARQL 1.1	RDF triplestore
Sqrrl Enterprise	Java	base de données orientée graphe
Wikibase	SPARQL	RDF triplestore
Agensgraph	Cypher, SQL	Base de donnée orientée graphe, dérivée de PostgreSQL

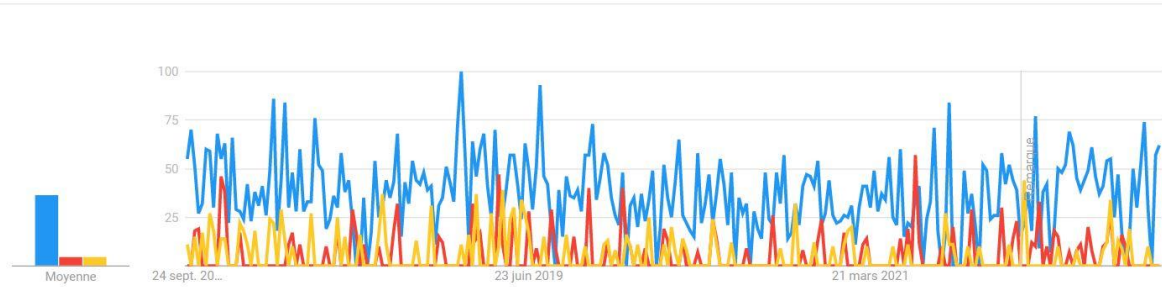
Pour voir lesquelles se démarquent et prennent le plus de place nous avons pris quelques exemple pour les comparer sur google trends, un site qui permet de mesurer l'intérêt de la population sur une recherche.

En France, sur les 5 dernières années en comparant Neo4j avec ArangoDB et OrientDB, on remarque un intérêt accru pour Neo4j. Il semble donc que ce soit la base de données à privilégier lorsque l'on souhaite mettre en place un graphe.

 Neo4j Sujet	 ArangoDB Logiciel	 OrientDB Logiciel	+ Ajouter une comparaison
---	---	---	---------------------------

France ▼ Cinq dernières années ▼ Toutes catégories ▼ Recherche sur le Web ▼

Évolution de l'intérêt pour cette recherche ?



3.3 – Analyse sur les usages

Après avoir effectué des recherches sur les autres bases de données orientées graphes existantes, il semble que Neo4j est la meilleure option. En effet, Cette technologie existe depuis plus de 10 ans et possède donc une grande maturité, elle dispose d'un grand nombre de packages et d'extensions pour être utilisé dans plusieurs langages différents ce qui permet de convenir à tout type de projets. C'est facile de prise en main, et suffisamment flexible pour s'adapter à tous les besoins, que ce soit un petit projet personnel, jusqu'au grand projet d'entreprise.

Pourquoi les grandes entreprises choisissent-elles Neo4j ?

- Neo4j est le leader incontesté des performances en matière de technologie graphique.
- Les développeurs de graphes et les scientifiques des données trouvent que Neo4j est le chemin le plus rapide vers la productivité et les informations.
- Contient les workflows d'analyse graphique et d'apprentissage automatique les plus avancés et les plus faciles à utiliser.
- Fiabilité, flexibilité et intégrité de la production pour les charges de travail transactionnelles/analytiques à volume élevé.

4 – Démonstration & synthèse

4.1 – Mise en place de l'architecture

La création de la base de données s'est faite sur une machine distante, il a donc fallu passer par *putty.exe* pour s'y connecter.

Une fois connecté, on commence par télécharger le package Neo4j :

- `$ sudo apt install neo4j`
- `$ sudo systemctl enable neo4j.service`
- `$ sudo systemctl start neo4j.service`

On peut ensuite vérifier si le service tourne correctement :

- `$ sudo systemctl status neo4j.service`

```
Output
• neo4j.service - Neo4j Graph Database
  Loaded: loaded (/lib/systemd/system/neo4j.service; enabled; vendor preset: enabled)
  Active: active (running) since Fri 2020-08-07 01:43:00 UTC; 6min ago
  Main PID: 21915 (java)
  Tasks: 45 (limit: 1137)
  Memory: 259.3M
  CGroup: /system.slice/neo4j.service
  . . .
```

Si tout est fonctionnel il est maintenant possible d'utiliser la commande suivante pour se connecter une première fois à l'interface de commande de Neo4j :

- `$ cypher-shell`

Il demandera un nom d'utilisateur et un mot de passe, il faut renseigner « neo4j » comme *username* et *password* puis changer le mot de passe.

Si vous voulez pouvoir vous connecter à la base depuis un autre poste connecté au même réseau, il faut dans ce cas décommenter une ligne dans le fichier `/etc/neo4j/neo4j.conf` :

```
. . .
# *****
# Network connector configuration
# *****

# With default configuration Neo4j only accepts local connections.
# To accept non-local connections, uncomment this line:
dbms.default_listen_address=0.0.0.0
. . .
```

Il suffira ensuite de renseigner l'adresse de la machine pour s'y connecter :

- `$ cypher-shell -a neo4j://10.8.2.35:7687`

Ou alors come ceci depuis l'interface graphique neo4j desktop :

Example Project

Name: Remote DBMS

Connect URL: neo4j://10.8.2.35:7687

Local DBMS
Remote connection
File

Cancel Next

4.2 – Dataset utilisé

Pour réaliser une démonstration pertinente, nous avons dû chercher un dataset qui mette en avant les avantages liés à la technologie des graphes. Comme vu précédemment, les moteurs de recommandations sont une bonne piste. Nous avons donc choisi un dataset sur les bières.

- <https://www.kaggle.com/datasets/rdoume/beerreviews>

Le dataset contient des avis d'utilisateurs concernant un large panel de bières, au total c'est 1,5 millions d'avis qui sont disponible dans ce fichier. Pour chaque avis on peut voir une note générale, des notes spécifiques à des critères, des arômes ressentis, le type de bière... . On peut distinguer 3 classes différentes qui serviront à réaliser notre graphe :

- Les *Reviewer*, qui sont les personnes qui laisse des avis.
- Les *Reviews*, qui sont les avis laissés par une personne.
- Les *Beers*, qui sont les bières présentes dans la base de données



4.3 – Création du graphe

Dans cette partie nous allons nous concentrer sur l'aspect technique du projet. Nous allons voir comment configurer NEO4J pour avoir des performances optimales. Nous verrons également comment créer des nœuds à partir d'un jeu de données, ainsi que de créer des relations entre ces nœuds. Nous terminerons par la recommandation de bières en fonction des notes données par les utilisateurs.

4.3.1 – Configuration de NEO4J

Il est essentiel de commencer cette partie par la **configuration de la mémoire** de NEO4J. Le logiciel est puissant mais demande beaucoup de ressources lors de la création de nœuds et de relations. Pour une utilisation fluide, nous ne pouvons que recommander cette étape, qui n'est pourtant pas obligatoire lors de l'installation. En effet, nous allons modifier les paramètres par défaut du logiciel afin qu'il utilise la mémoire disponible de manière optimale.

Dans notre cas, NEO4J est installé sur une machine virtuelle (VM) Linux basée sur un serveur distant. La VM possède 4G de mémoire, pour obtenir la configuration mémoire optimale il suffit de :

1. Se placer dans le répertoire d'installation : `/etc/neo4j`
2. Exécuter la commande suivante : `neo4j-admin memrec --memory=4G`

Dans notre cas, NEO4J nous a recommandé de paramétrer la mémoire de la façon suivante :

- `dbms.memory.heap.initial_size=2g`
- `dbms.memory.heap.max_size=2g`
- `dbms.memory.pagecache.size=512m`

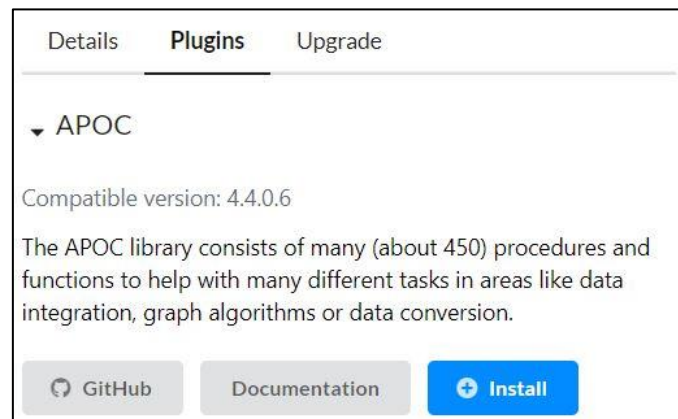
Il maintenant aller modifier le fichier `neo4j.conf` en ajoutant les 3 lignes ci-dessus.

Maintenant que l'utilisation mémoire a été configurée, nous pouvons **installer les packages** dont nous aurons besoins dans ce projet. Les 4 packages de bases sont APOC, Graph Data Science Library, NEO4J Streams et Neosemantics. Chacun possède un intérêt unique et répond à des besoins précis en mettant à disposition des procédures (fonctions) facilitant l'utilisation du langage cypher.

« *La bibliothèque de l'APOC se compose de nombreuses procédures et fonctions (environ 450) qui facilitent de nombreuses tâches différentes dans des domaines tels que l'intégration de données, les algorithmes graphiques ou la conversion de données.* » [documentation NEO4J](#)

Le package APOC facilite grandement l'utilisation de NEO4J et beaucoup de codes présent sur internet utilisent des procédure de ce package. Il est donc préférable de l'installer dès le début.

Pour l'installer depuis la version desktop, il suffit de sélectionner notre projet puis il n'y a plus qu'à cliquer sur installer.



Pour finaliser la configuration de NEO4J et pour être prêt à utiliser NEO4J, il ne nous reste plus qu'à déposer le jeu de données sur la VM. Pour transférer le fichier csv, nous utilisons l'exécutable pscp.exe et plaçons le fichier dans le dossier approprié : `/var/lib/neo4j/import/beer_reviews.csv`

4.3.2 – Création des nœuds

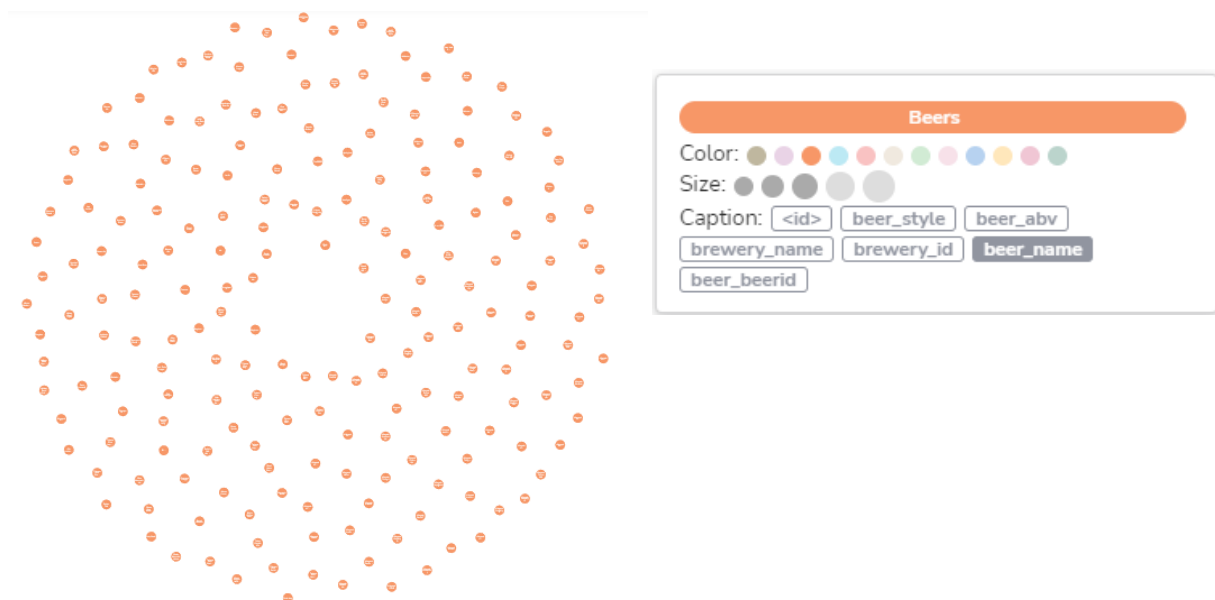
Nous avons créé 3 types de nœuds, comme vu dans la description du dataset. Les nœuds sont : Beers, Reviews et Reviewers. Le dataset étant énorme, nous avons décidé de ne pas le charger entièrement car nous n'avons pas l'utilité de 1,5 million de reviews. Nous avons uniquement chargé les bières, reviews et reviewers associées au 25 premiers brewery_id.

Vous trouverez ci-après comme exemple, le code permettant de créer les nœuds des bières :

```
LOAD CSV WITH HEADERS FROM 'file:///beer_reviews.csv' AS row
WITH
  toInteger(row.brewery_id) as brewery_id,
  toInteger(row.beer_beerid) as beer_beerid,
  toString(row.brewery_name) as brewery_name,
  toString(row.beer_name) as beer_name,
  toString(row.beer_style) as beer_style,
  toInteger(row.beer_abv) as beer_abv
WHERE brewery_id < 25
MERGE (b:Beers {brewery_id: brewery_id, beer_beerid:beer_beerid})
SET
  b.brewery_id = brewery_id,
  b.beer_beerid = beer_beerid,
  b.brewery_name = brewery_name,
  b.beer_name = beer_name,
  b.beer_style = beer_style,
  b.beer_abv = beer_abv
RETURN
  count(b);
```

Nous pouvons observer que la syntaxe est très proche de celle du SQL.

Nous obtenons le rendu visuel suivant après avoir exécuté la requête. Nous pouvons voir l'ensemble des bières représentées par les points oranges. Sur l'image de gauche, nous pouvons voir tous les attributs associés à chaque bière.



Nous avons procédé de la même manière avec les reviews et les reviewers.

4.3.3 – Création des relations entre les nœuds

Maintenant que nous avons créé les 3 types de nœuds, il nous faut les lier entre eux. Pour cela nous allons créer des relations. Nous allons créer 2 relations : reviewed et did_a_review. La relation

reviewed signifie qu'une bière s'est faite notée dans une review. La relation did_a_review lie une review et un reviewer, en d'autres terme elle lie une review à son auteur.

Ces deux relations nous permettrons entre autres de voir l'ensemble des bières notées par un utilisateur, ou l'ensemble d'utilisateurs ayant notés une bière. Ce qui nous sera utile pour recommander une bière dans la prochaine partie.

Voici le code permettant de créer la relation reviewed :

```
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM 'file:///beer_reviews.csv' AS row
WITH toInteger(row.brewery_id) AS brewery_id, toInteger(row.beer_beerid) AS beer_beerid, toString(row.review_profilename) AS review_profilename
MATCH (p:Reviews {brewery_id: brewery_id, beer_beerid: beer_beerid, review_profilename: review_profilename})
MATCH (o:Beers {brewery_id: brewery_id, beer_beerid: beer_beerid})
MERGE (o)-[rel:reviewed {brewery_id: brewery_id, beer_beerid: beer_beerid, review_profilename: review_profilename}]->(p)
RETURN count(rel);
```

Nous obtenons les rendus visuel suivant :

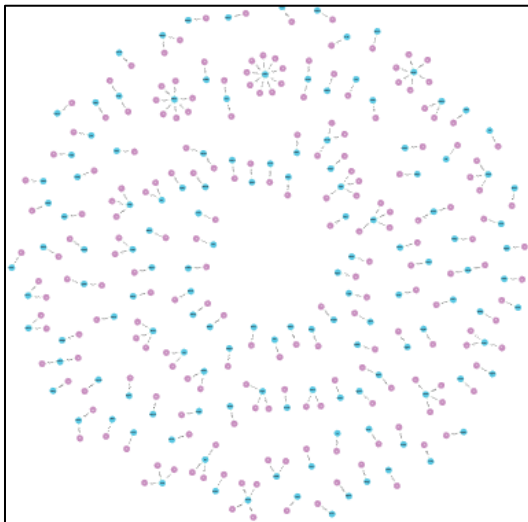


Figure 2- rendu visuel de la relation did_a_review

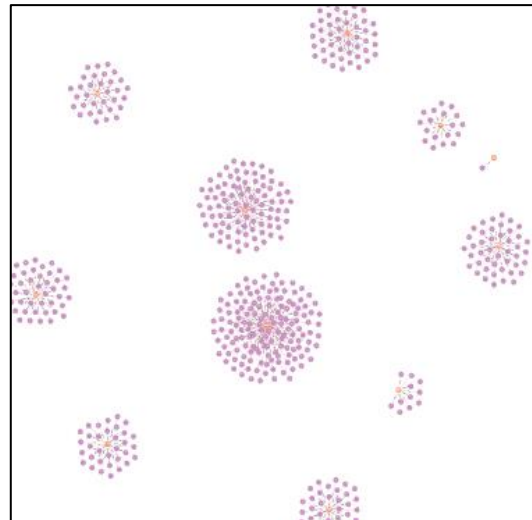


Figure 1- rendu visuel de la relation reviewed

Sur les images ci-dessus, le nombre de nœuds affiché a été limité pour ne pas surcharger les images. Nous pouvons observer sur l'image le rendu visuel de did_a_review (gauche) que la majorité de reviewers (bleu) ne font qu'une review, mais certains en font jusqu'à 10, cela montre une certaine disparité. Sur l'image de gauche, nous observons nettement que certaines bières (orange) sont

beaucoup plus notées (violet) que d'autres. Nous pouvons déjà nous interroger sur la probable popularité des bières les plus notées.

Maintenant que les relations sont créées, nous pouvons faire des requêtes. Nous pouvons par exemple, afficher l'ensemble des reviews d'une bière ou encore l'ensemble des reviews et des bières d'un utilisateur.

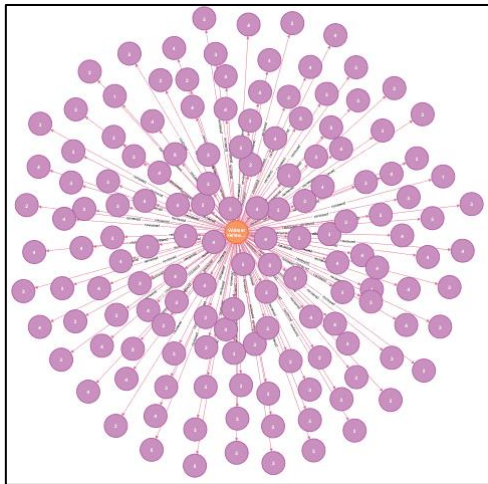


Figure 3- l'ensemble des reviews (violet) d'une bière (orange)

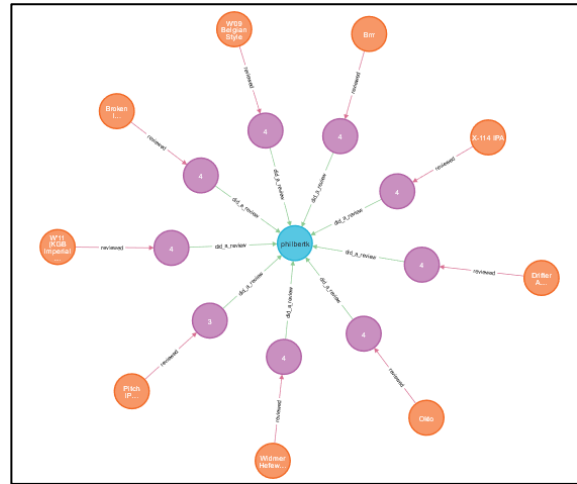


Figure 4 - l'ensemble des reviews (violet) et des bières (orange) d'un utilisateur (bleu)

Sur l'image de gauche, nous pouvons voir l'ensemble des reviews (violet) sur une bière (orange), avec les notes globales attribuées à la bière dans les ronds violets. Tandis que sur l'image de droite, on peut voir l'ensemble des bières (orange) notées (violet) par l'utilisateur (bleu).

4.3.4 – La recommandation de bières

NEO4J est très utilisé pour faire des recommandations. Nous avons donc voulu tester cette fonctionnalités. Nous sommes partis de la documentation de NEO4J qui est très détaillée, puis nous avons adapté les codes à notre cas d'usage.

Nous avons construit l'algorithme pour remonter les bières similaires d'une bière données. Pour établir si une bière est similaire à une autre, nous avons choisit de procéder par étapes. Dans un premier temps, nous sélectionnons les bières ayant été notées par les utilisateurs ayant fait une review sur une bière donnée. Ensuite, parmi les bières que nous avons, nous ne gardons que les bières ayant une note au moins égale à 4/5.

Ainsi, nous avons une liste de bières similaires à notre bière. Nous avons ensuite trié les bières en fonction du nombre de fois qu'elles apparaissent dans la liste.

Voici le code permettant d'obtenir les recommandations :

```
//Displaying 5 most similar beers to "Widmer Hefeweizen" according to the overall review:
//selecting Reviewers who did a Review which reviewed the Beer called "Widmer Hefeweizen"
MATCH (a:Beers {beer_name: "Widmer Hefeweizen"})-[:reviewed]->(b:Reviews)-[:did_a_review]->(c:Reviewer)
//selecting Beers being reviewed by the profilenames who also reviewed "Widmer Hefeweizen"
MATCH (c)-[:did_a_review]->(e:Reviews)-[:reviewed]->(h:Beers)
//We only keep profilenames who rated our beer at min 4
WHERE b.review_overall>=4
RETURN h.beer_name as similar_beers, count(e) AS number_of_reviews
ORDER BY number_of_reviews DESC
SKIP 1 //we skip first row as it is the beer we are comparing
LIMIT 5
```

Cela nous donne la liste de bière les plus similaires à la bière nommée : « Widmer Hefeweizen » :

similar_beers	number_of_reviews
"Drifter Pale Ale"	7
"Brrr"	5
"W'11 (KGB Imperial Stout)"	5
"Deadlift Imperial IPA"	3
"X-114 IPA"	3

Glossaire

VM	Virtual Machin (Machine Virtuelle)
EA	(Modèle) Entité-association
CSV	Comma-Separated Values
JSON	JavaScript Object Notation
ACID	Atomicité, Cohérence, Isolation, Durabilité