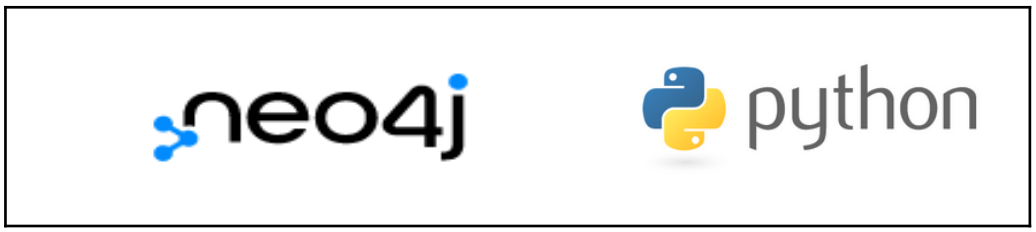
UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Faculté Sciences et Ingénierie

Rapport du projet NEO4J



Réalisateurs :

- **Ahmed Amine Ghorbel**
- **Issa Abakar**
- Filière : IAFA
- Mineur : DC

Plan du rapport

1. Introduction générale
2. Cas d'étude : Modélisation
3. Interrogation des données
4. Résultats des tests
5. Conclusion

1. Introduction générale

De nos jours, nous remarquons l'existence de plusieurs sites participatifs dans lesquels, des utilisateurs de la plateforme partage des avis qui concernent différents business tels que des restaurants, des commerciaux, des écoles et des établissements locaux. Parmi ces sites, on cite **Yelp** (https://www.yelp.com) , qui est un site participatif d'avis sur les commerces locaux et de réseautage social, où les utilisateurs peuvent soumettre un avis de leurs produits et services à l'aide d'un système de notation de 1 à 5 étoiles. Les sociétés peuvent également actualiser leurs coordonnées, leurs heures d'exploitation et d'autres renseignements de base ou ajouter des renseignements précis. En plus de rédiger des commentaires, les utilisateurs peuvent réagir aux commentaires, planifier des activités ou parler de leur vie personnelle.

L'objectif du projet est de faire l'analyse des données Yelp concernant les utilisateurs, les avis des différents utilisateurs et les restaurants sous forme des données orientés graphes à la base d'un schéma modélisé dédié pour notre cas : Création des nœuds pour chaque entité étudiée et les différentes relations correspondantes avec le logiciel neo4j desktop. Après, on interroge la base de données graphe créée avec le langage Cypher qui est langage informatique de requête orienté graphe utilisé par Neo4j. Ce projet fait partie de l'étude sur des restaurants implémentés dans la région du Delaware faite par un groupe industriel. Ce groupe souhaite inviter dans chacun de ses restaurants des utilisateurs de la plateforme Yelp qui joueront ensuite le rôle d'influenceurs auprès de leurs amis afin de les promouvoir et d'attirer un maximum de clientèle.

Pour sélectionner les utilisateurs qui sont des influenceurs pour chacun des restaurants, le groupe propose de calculer un score d'influence des utilisateurs de la plateforme.

2. Cas d'étude: Modélisation

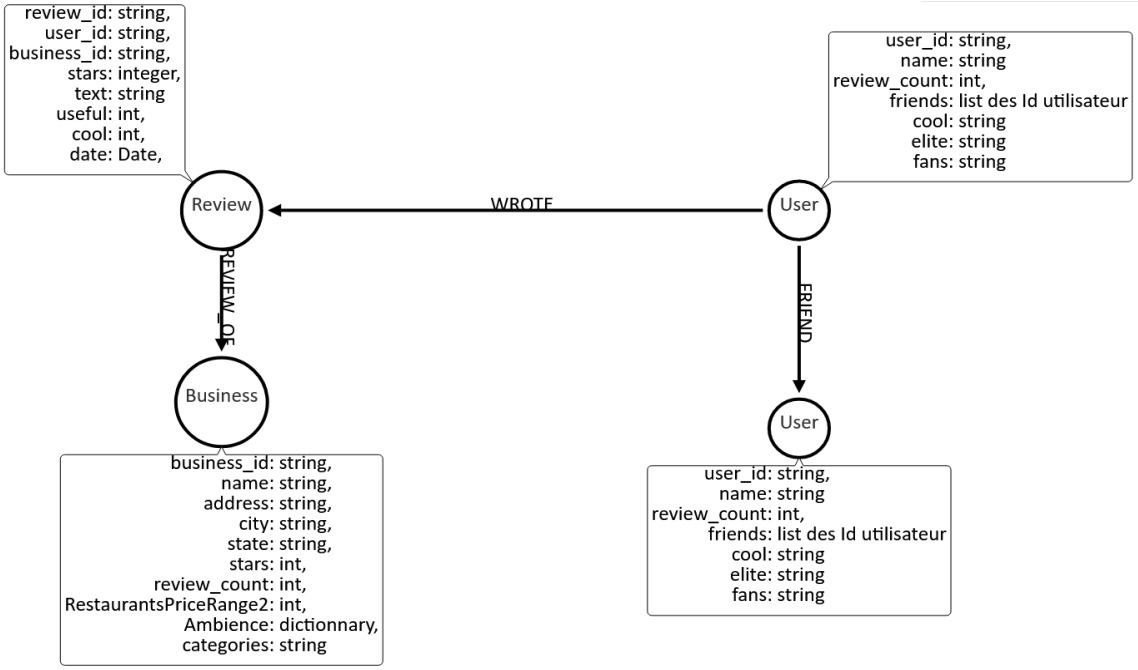


Figure 1 : Schéma de modélisation détaillé

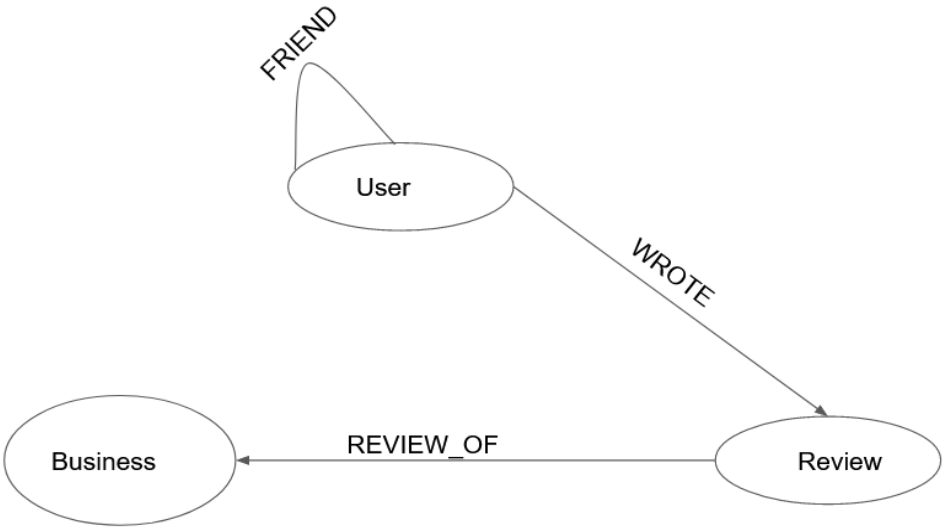


Figure 2 : Schéma de modélisation simplifié

Description de la modélisation:

- Les restaurants ***(Business)*** sont décrits par diverses informations comme : l'identifiant du restaurant, son nom, l'adresse correspondante, la ville ou il y a le restaurant, nombre de stars, nombre d'avis, gamme de prix des restaurants, les styles d'ambiance proposés par le business et les catégories correspondantes.

- Les utilisateurs ***(User)*** sont recensés avec différentes informations, parmi lesquelles nous citons : L'identifiant de l'utilisateur, son nom, le nombre d'avis qu'il propose, la liste des amis qui ont un lien d'amitié avec lui, …

- Les avis ***(Reviews)*** sont décrits par leurs identifiants et les identifiants du business et l'utilisateur qui leur correspond, une note globale sur une échelle de 1 à 5, le commentaire de chaque avis, chaque avis est horodaté, …

- Les utilisateurs peuvent poster des avis ***(WROTE)*** sur les restaurants constitués notamment d'une note globale et d'un commentaire. WROTE est une relation qui met un User en lien avec ses Reviews.

- Certains utilisateurs peuvent posséder ou non une liste d'amis (***Friend)*** qui sont également des utilisateurs de la plateforme Yelp. De ce fait, FRIEND est une relation entre deux Users, qui sont amis a priori.
- Un Review est destiné à un Business et c'est décrit par la relation ***REVIEW_OF.***

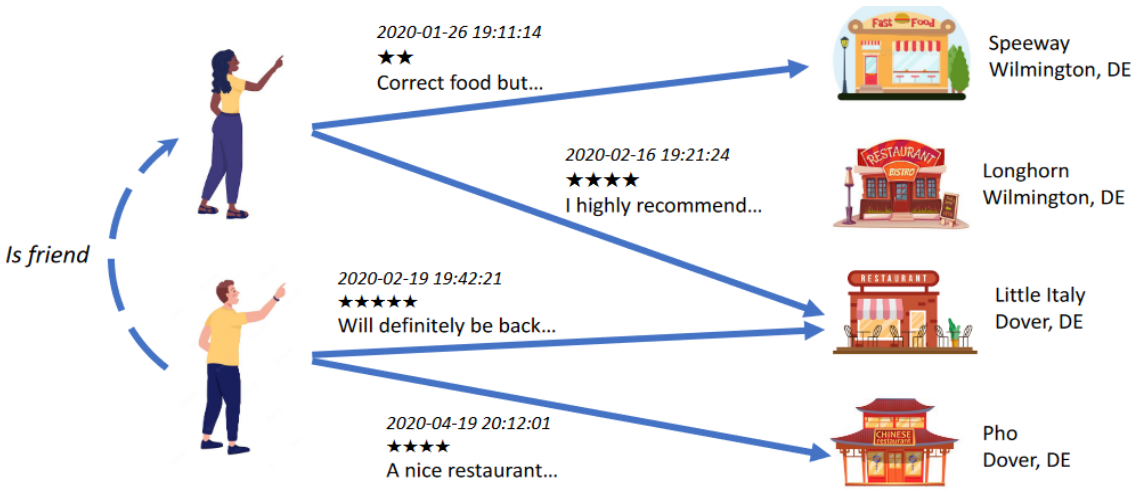


Figure 3 : Fonctionnement de la plateforme participative d'avis Yelp

3. Interrogation des données

- *Donner le nombre de nœuds par label :*

```
MATCH (user:User) return (count(user))
```

→ 23082 users.

```
MATCH (business:Business) return (count(business))
```

→ 961 business.

```
MATCH (review:Review) return (count(review))
```

→ 49150 reviews.

- *Donner le nombre de relations par type :*

```
MATCH ()-[r]->() RETURN type(r), count(*);
```

"type(r)"	"count(*)"
"WROTE"	49150
"FRIEND"	42776
"REVIEW_OF"	49150

- *Donner la ou les catégories du restaurant 8th & Union Kitchen :*

```
MATCH (business:Business{name: "8th & Union Kitchen" })
RETURN business.categories;
```

"business.categories"
"Vietnamese, Asian Fusion, Bars, American (New), Restaurants, Breakfast & Brunch, Nightlife, Thai, Gluten-Free"

- *Donner la ou les ambiances du restaurant 8th & Union Kitchen :*

```
MATCH ((business:Business({name: "8th & Union Kitchen" })
RETURN business["attributes.Ambience"]
```

"business["attributes.Ambience"]"
"{'touristy': False, 'hipster': False, 'romantic': False, 'divey': False, 'intimate': False, 'trendy': True, 'upscale': False, 'classy': True, 'casual': True}"

- *Donner les reviews du restaurant Tokyo Sushi :*

```
Match (r:Review)-[:REVIEW_OF]->(b:Business)
Where b.name = "Tokyo Sushi"
Return COLLECT(r.review_id)
```

"COLLECT(r.review_id)"
["1687","1599","2761","2636","1305","866","760","583","558","467","261","105","86","26","6","5","1078","1045","1002"]

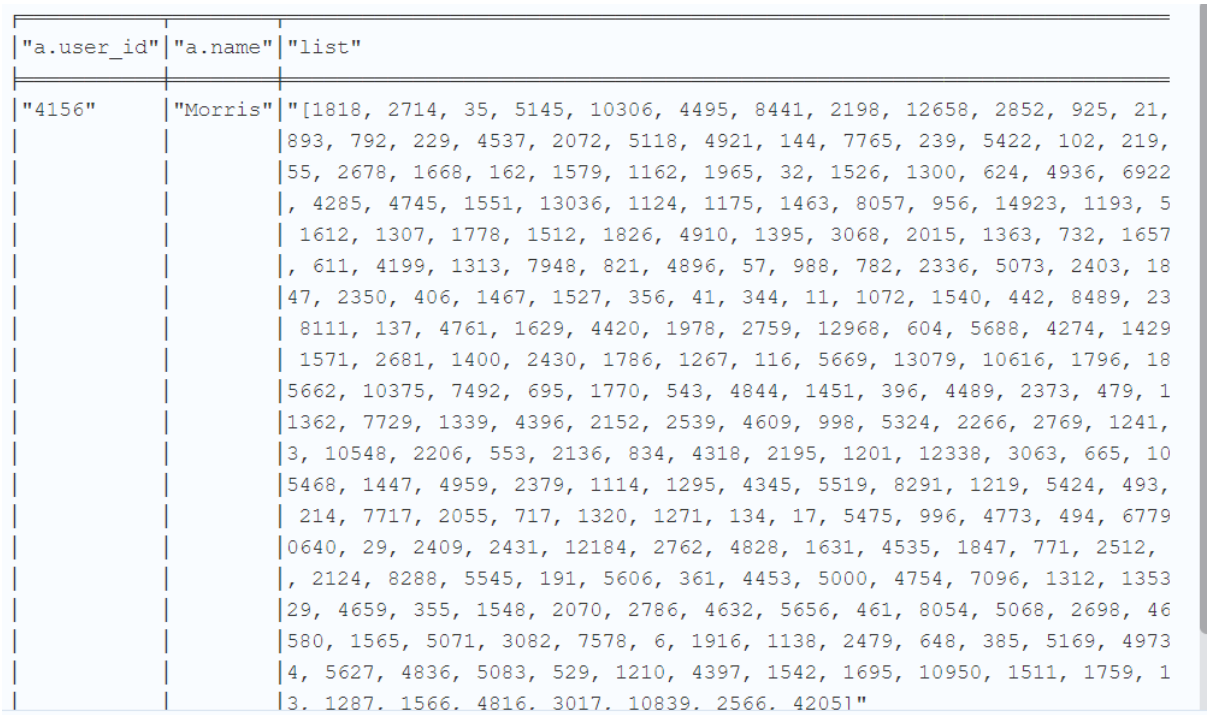
- *Donner les couples d'utilisateurs amis qui ont reviewé le restaurant Tokyo Sushi :*

```
Match (f1:User)-[:FRIEND]->(f2:User)
Match
(f2)-[:WROTE]->(r1:Review)-[:REVIEW_OF]->(b:Business{name:"Toky
o Sushi"})
Match
(f1)-[:WROTE]->(r2:Review)-[:REVIEW_OF]->(b:Business{name:"Toky
o Sushi"})
Return f1.user_id,f2.user_id,Count(b)
```

"f1.user_id"	"f2.user_id"	"Count(b)"
"415"	"32"	2
"32"	"415"	2

- *L'utilisateur qui a le plus d'amis :*

```
MATCH (a:User)
RETURN a.user_id ,a.name, a.friends AS list ORDER BY size(list)
DESC LIMIT 1
```



```
"a.user_id"|"a.name"|"list"
"4156"     |"Morris"|"[1818, 2714, 35, 5145, 10306, 4495, 8441, 2198, 12658, 2852, 925, 21,
           |        |893, 792, 229, 4537, 2072, 5118, 4921, 144, 7765, 239, 5422, 102, 219,
           |        |55, 2678, 1668, 162, 1579, 1162, 1965, 32, 1526, 1300, 624, 4936, 6922
           |        |, 4285, 4745, 1551, 13036, 1124, 1175, 1463, 8057, 956, 14923, 1193, 5
           |        | 1612, 1307, 1778, 1512, 1826, 4910, 1395, 3068, 2015, 1363, 732, 1657
           |        |, 611, 4199, 1313, 7948, 821, 4896, 57, 988, 782, 2336, 5073, 2403, 18
           |        |47, 2350, 406, 1467, 1527, 356, 41, 344, 11, 1072, 1540, 442, 8489, 23
           |        | 8111, 137, 4761, 1629, 4420, 1978, 2759, 12968, 604, 5688, 4274, 1429
           |        | 1571, 2681, 1400, 2430, 1786, 1267, 116, 5669, 13079, 10616, 1796, 18
           |        |5662, 10375, 7492, 695, 1770, 543, 4844, 1451, 396, 4489, 2373, 479, 1
           |        |1362, 7729, 1339, 4396, 2152, 2539, 4609, 998, 5324, 2266, 2769, 1241,
           |        |3, 10548, 2206, 553, 2136, 834, 4318, 2195, 1201, 12338, 3063, 665, 10
           |        |5468, 1447, 4959, 2379, 1114, 1295, 4345, 5519, 8291, 1219, 5424, 493,
           |        | 214, 7717, 2055, 717, 1320, 1271, 134, 17, 5475, 996, 4773, 494, 6779
           |        |0640, 29, 2409, 2431, 12184, 2762, 4828, 1631, 4535, 1847, 771, 2512,
           |        |, 2124, 8288, 5545, 191, 5606, 361, 4453, 5000, 4754, 7096, 1312, 1353
           |        |29, 4659, 355, 1548, 2070, 2786, 4632, 5656, 461, 8054, 5068, 2698, 46
           |        |580, 1565, 5071, 3082, 7578, 6, 1916, 1138, 2479, 648, 385, 5169, 4973
           |        |4, 5627, 4836, 5083, 529, 1210, 4397, 1542, 1695, 10950, 1511, 1759, 1
           |        |3, 1287, 1566, 4816, 3017, 10839, 2566, 42051"
```

- *Les amis des reviewers du restaurant Tokyo Sushi qui ont reviewers des restaurants avec les mêmes ambiances :*

```
match (u:User)-[r:REVIEW_OF]->(n:Business{name: 'Tokyo Sushi'}),
(u2:User)-[:FRIEND]->(u:user),
(u2:User)-[:REVIEW_OF]->(n2:Business)
where n['Attributes.Ambiences'] = n2['Attributes.Ambiences']
return distinct u2.name,u.name
```

- *combien de paires de personnes ont au moins 10 restaurants aimés en commun ? (un restaurant est 'aimé' si la review a une note (stars) >= 4)*

```
Match (f1:User)-[:FRIEND]->(f2:User)
Match (f2)-[:WROTE]->(r1:Review)-[:REVIEW_OF]->(b1:Business)
Match (f1)-[:WROTE]->(r2:Review)-[:REVIEW_OF]->(b2:Business)
where toInteger(b1.stars)>= 4 AND toInteger(b2.stars)>= 4
```

```
Return f1.user_id,f2.user_id,count(b1)>10,count(b2)>10
```

On prend que les lignes qui présentent true - true :

"f1.user_id"	"f2.user_id"	"count(b1)>10"	"count(b2)>10"
"18842"	"406"	false	false
"12590"	"406"	false	false
"6940"	"406"	false	false
"4134"	"406"	false	false
"4156"	"406"	false	false
"2579"	"406"	false	false
"363"	"406"	false	false
"1085"	"2324"	true	true
"553"	"2324"	false	false
"4253"	"384"	false	false

4. Résultats des tests

- *la bibliothèque py2neo et quelques tests d'initiation:*

!pip install py2neo

Py2neo est une bibliothèque cliente et un ensemble d'outils pour travailler avec Neo4j à partir des applications Python. Py2neo est utilisé pour connecter Neo4j à Python. Neo4j fournit des pilotes (drivers) qui nous permettent d'établir une connexion avec la base de données et de développer des applications qui créent, lisent, mettent à jour et suppriment des informations du graphe.

from py2neo import Graph

Un mappeur de graphes d'objets construit sur le driver python de Neo4j.

```
graph = Graph("bolt://localhost:7687", auth=("neo4j", "mot
de passe de la base de données"))
```

Test de quelques requêtes :

```
q = "match (u:USER) return count(u)"
res = graph.run(q).to_table()
for record in res :
    print(record[0])
```

```
→ 23082
  ------------------------------------------------------
```

```
q = "match (restau:RESTAURANT) return count(restau)"
res = graph.run(q).to_table()
for record in res :
    print(record[0])
```

```
→ 961
  ------------------------------------------------------
```

```
q = "match (review:REVIEW) return count(review)"
res = graph.run(q).to_table()
for record in res :
    print(record[0])
```

```
→ 49150
  ------------------------------------------------------
```

```
q = "MATCH (restau:RESTAURANT{name: '8th & Union Kitchen'
}) RETURN restau.categories;"
res = graph.run(q).to_table()
for record in res :
```

```
    print(record[0])
```

```
Vietnamese, Asian Fusion, Bars, American (New), Restaurants, Breakfast & Brunch,
Nightlife, Thai, Gluten-Free
```

```
  ----------------------------------------------------
q = "MATCH (restau:RESTAURANT{name: '8th & Union Kitchen'
}) RETURN restau['attributes.Ambience']"
res = graph.run(q).to_table()
for record in res :
    print(record[0])
  ----------------------------------------------------
```

```
{'touristy': False, 'hipster': False, 'romantic': False, 'divey': False, 'intimate':
False, 'trendy': True, 'upscale': False, 'classy': True, 'casual': True}
```

```
q = "Match (r:Review)-[:REVIEW_OF]->(b:Business) Where
b.name = 'Tokyo Sushi' Return COLLECT(r.review_id)"
res = graph.run(q).to_table()
for record in res :
    print(record[0])
```

```
['1687', '1599', '1305', '2761', '2636', '866', '760', '583', '558', '467', '261',
'1078', '1045', '1002', '105', '86', '26', '6', '5']
```

- *Test de l'application : moteur de recommandation d'influenceurs:*

1) L'entrée de l'application :entrée des informations sur un restaurant à promouvoir :

Pour ville (chaîne de caractères) :

```
print('Entrer la ville:')
ville = input()
print('La ville :' + ville)
```

Pour ambiances (tableau de chaînes de caractères) :

```
ambiances = input('Enter les ambiances : ')
print("\n")
ambiances = ambiances.split()
# print list des ambiances
print('list des ambiances: ', ambiances)
```

Pour catégories (tableau de chaînes de caractères) :

```
categories = input('Enter les categories : ')
print("\n")
categories = categories.split()
# print list des catégories
print('list des categories: ', categories)
```

catégorie tarifaire (entier) :

```
print('Entrer la categories tarifaire (entier):')
categorietarifaire = input()
print('la catégorie tarifaire : ' + categorietarifaire)
```

Ici, on va essayer de voir comment lire une valeur entrée et l'utiliser dans une requête d'une façon dynamique :

```
q = "Match (r:Review)-[:REVIEW_OF]->(b:Business) Where b.name = 'Tokyo Sushi' Return COLLECT(r.review_id)"
```

on reprend cette requête la, et on essaye de passer le nom du restau `'Tokyo Sushi'` dans la requête après avoir récupéré la valeur de l'invite de commande :

```
print('Entrer le nom du restau:')
name = input()
print('Le nom du restau :' + name)
print(type(name))

# 'Tokyo Sushi'
```

```
q = "Match (r:Review)-[:REVIEW_OF]->(b:Business) Where b.name =
$name Return COLLECT(r.review_id)"

params = {'name':name}
res = graph.run(q,params).to_table()

for record in res :
    print(record[0])
```

name sera le `'Tokyo Sushi'` donc on a comme résultat :

```
Entrer le nom du restau:
Tokyo Sushi
Le nom du restau :Tokyo Sushi
<class 'str'>
['1687', '1599', '1305', '2761', '2636', '866', '760', '583', '558', '467', '261',
'1078', '1045', '1002', '105', '86', '26', '6', '5']
```

Remarque :

Nous avons rencontré un souci pendant le calcul des scores au début notamment à la ligne 72 du script_31 : nous voulons récupérer max_fans (Qui est le nombre maximum de fans d'un User appartenant à l'ensemble des Users) mais on avait pas converti l'attribut fans en entier dans les fichiers csv. Ce qui fait qu'on n'avait pas la bonne valeur de max_fan. En effet, on avait un score qui dépassait 1 (celui de 529) et on a presque les mêmes scores pour tous les tests.

Après modification, on avait des bons résultats mais on a fait des tests sur 3000 Users au lieu de 23082 par manque de temps sachant qu'un test peut prendre jusqu'à 1h30 d'attente.

Voici la ligne concerné :

```
# centrality_s3
fans = "Match (u:User{user_id: $id}) return u.fans"
max_fans_ancien = "Match (u:User) return u.fans ORDER BY u.fans DESC LIMIT 1"
max_fans = "Match (u:User) return toInteger(u.fans) ORDER BY toInteger(u.fans) DESC LIMIT 1"
```

On a gardé la valeur dans max_fans à la fin.

Exécution des tests avec max_fans_ancien:

- Test 1 :

```
"ville" : "Wilmington",
"Ambiances" :  ["casual"],
"Categories" :  ["Pizza","Burgers","Italian"],
"pricerange" : 1
```

- Test 2 :

```
"ville" : "Wilmington",
"Ambiances" :  ["casual","romantic"],
"Categories" :  ["Chinese"],
"pricerange" : 2
```

Les 10 meilleures utilisateurs :

```
100%|| 23082/23082 [1:37:17<00:00,  3.95it/s]
[(529, 3.631853826015675), (84, 0.9549358180405055), (159, 0.9321995966336962), (4156,
0.9298546602549067), (7448, 0.8583564960539637), (1287, 0.8167084672950886), (889, 0.8139402790398268),
(245, 0.7332823241346722), (32, 0.7166519157658049), (14237, 0.6785879107529034)]
```

Les 10 pires utilisateurs :

```
[(522, 7.4487895716946e-05), (546, 7.4487895716946e-05), (628, 7.4487895716946e-05), (744,
7.4487895716946e-05), (878, 7.4487895716946e-05), (903, 7.4487895716946e-05), (1011, 7.4487895716946e-05),
(1029, 7.4487895716946e-05), (1142, 7.4487895716946e-05), (1153, 7.4487895716946e-05)]
```

- Test 3 :

```
"ville" : "Wilmington",
"Ambiances" :  ["hipster"],
"Categories" :  ["Nightlife","Bars"],
"pricerange" : 1
```

- Test 4 :

```
"ville" : "New Castle",
"Ambiances" :  ["causal","classy"],
"Categories" :  ["Coffee & Tea"],
```

```
"pricerange" : 2
```

- Test 5 :

```
"ville" : "New Castle",
"Ambiances" :  ["classy"],
"Categories" :  ["Seafood"],
"pricerange" : 1
```

Les 10 meilleures utilisateurs :

```
Exécution de test 5 en cours
100%|██████████| 23082/23082 [57:40<00:00,  6.67it/s][(529, 3.631853826015675), (84,
0.9550095754982536), (159, 0.9321987614506222), (4156, 0.9298522656188914), (7448,
0.8583561986871469), (1287, 0.8167078684448811), (889, 0.813940152305467), (245,
0.7332814551739492), (32, 0.716667738189164), (14237, 0.6785871661512652)]
```

Les 10 pires utilisateurs :

```
[(522, 7.4487895716946e-05), (546, 7.4487895716946e-05), (628, 7.4487895716946e-05), (744,
7.4487895716946e-05), (878, 7.4487895716946e-05), (903, 7.4487895716946e-05), (1011,
7.4487895716946e-05), (1029, 7.4487895716946e-05), (1142, 7.4487895716946e-05), (1153,
7.4487895716946e-05)]
```

Nous avons fait les tests nécessaires pour être sûr du bon fonctionnement de notre fonction de calcul de score.

Exécution des tests avec 3000 Users :

Ses résultats ne prennent en compte que les 3000 premiers Users et retournent les 10 utilisateurs les plus pertinents :

```
TEST1_3000 = """
            [(84, 0.22201032265740103), (2939, 0.2179419812489107), (2712, 0.20811883797828362), (2655, 0.20546139265762173),
            (297, 0.20339329012979906), (2644, 0.2021142041023019), (2286, 0.18714257463362843), (1569, 0.1822092070676416 3),
            (1175, 0.1764654284213893), (1320, 0.17373673948085297)]
            """

TEST2_3000 = """
            [(1980, 0.25491173925195565), (2425, 0.25491173925195565), (84, 0.22201078429955448), (1175, 0.1762574379434059 3),
            (1320, 0.17346651496660595), (985, 0.16889710559318794), (2939, 0.16792554199196513),
             (529, 0.16756811172996117), (32, 0.16323896742013388), (2504, 0.15622156636122828)]
            """
TEST4_3000 = """
           [(84, 0.22201032265740103), (1175, 0.1764654284213893), (1320, 0.17346484059784298), (985, 0.16857863425560832),
            (529, 0.16756811172996117), (2939, 0.16729263059956007), (32, 0.163302327186304), (2504, 0.15622156636122828),
            (297, 0.15262405936056828), (2600, 0.1519466379277465)]
            """
```

```
TEST3_3000 = """
            [(985, 0.2694960654482689), (84, 0.22201032265740103), (2311, 0.21707509322259344), (2378, 0.20631745997003637),
            (2167, 0.20628157089358531), (1274, 0.20315185743088865), (2286, 0.18714257463362843), (1175, 0.1764654284213893),
            (1320, 0.17346484059784298), (436, 0.1713381799973485)]
            """

TEST5_3000 = """
            [(84, 0.2220845417573025), (2600, 0.20279409555486513), (1716, 0.18462341295097112), (1175, 0.17625085916916314),
            (1320, 0.17346484059784298), (985, 0.16857863425560832), (529, 0.16756811172996117), (2939, 0.16729263059956007),
            (32, 0.1632547898434929), (2504, 0.15622156636122828)]
            """
```

Exécution du tests 4 :

Finalement, nous avons réussi à effectuer le Test 4 avec l'ensemble de 23082 Users en utilisant le bon nombre de max_fans.

Voici le résultat en capture :

```
Exécution de test 4 en cours
100%|██████████| 23082/23082 [1:01:20<00:00,  6.27it/s]
[(7146, 0.4023656236138967), (21174, 0.40189389490380323), (18600, 0.3052921334134525),
(21847, 0.3052825114060152), (12531, 0.3039399536906079), (16935, 0.3038649132794126),
(20381, 0.303854561047423), (17184, 0.3033733766461039), (20171, 0.30270687666294066),
(12978, 0.30236949510412975)]
[(522, 7.4487895716946e-05), (546, 7.4487895716946e-05), (628, 7.4487895716946e-05), (744,
7.4487895716946e-05), (878, 7.4487895716946e-05), (903, 7.4487895716946e-05), (1011,
7.4487895716946e-05), (1029, 7.4487895716946e-05), (1142, 7.4487895716946e-05), (1153,
7.4487895716946e-05)]
```

5. Conclusion

Dans ce projet, on a fait recours à l'utilisation des bases données orientés graphes pour analyser les données d'un site participatif d'avis qui est Yelp. Globalement, des utilisateurs donnent leur avis sur des restaurants et ils possèdent ou non des amis qui sont aussi des utilisateurs du site. On a à notre disposition trois sources de données qui sont celles des utilisateurs, celles des restaurants et celles des revues. Dans un premier temps, on les convertit sous un format csv avec Python pour qu'elles soient importées après dans le neo4j desktop. Avec l'obtention des nœuds et les relations existantes, on a interrogé la base de données par des requêtes pour des fins d'analyses et d'interprétations. Ensuite, nous développons une application python qui est moteur de recommandation d'influenceurs, en effet, l'idée de l'application c'est qu'un groupe industriel sélectionne ces utilisateurs-influenceurs pour chacun des restaurants implémentés dans la région du Delaware à la base de calcul des facteurs basés sur des scores d'influences, ces facteurs sont le facteur de centralité de l'utilisateur dans le réseau $f_{centralite}$, facteur de validité des commentaires $f_{validite}$, facteur d'adéquation au restaurant $f_{adequation}$ et le facteur géographique $f_{geographique}$. Au final le score le score d'un utilisateur u_i est calculé comme suit:

$$s_i = \alpha \times f_{centralite} + \beta \times f_{validite} + \gamma \times f_{adequation} + \delta \times f_{geographique}$$