RAPPORT BD graphe Neo4j

Christian QIAN | christian-qian@live.fr | Numero Etudiant : 21964319

1 - Dataset Covid

Le dataset concerne les cas d'infection du Covid en Corée. Les fichiers d'origine utilisés sont PatientInfo.csv et Region.csv. Le premier fichier comporte un patient pour chaque ligne avec son age, sexe, pays d'origine, le lieu d'infection et la raison de l'infection, occasionnellement le patient ayant causé l'infection, le nombre de personne en contact, les dates de dépistage/sortie/décès et son état (isolé, sortie,décédé). Le deuxième fichier concerne le lieu d'infection avec province, ville et le nombre d'écoles maternelles, primaires et universités.

2 - Importation Postgres

Les fichiers comportent de nombreuses erreurs de format dans le csv qu'il est nécessaire de nettoyer pour que la commande \copy fonctionne (tirets, espaces vides…).
Pour la table Région, les données sont propres pas besoin de changement.
Pour PatientInfo, il s'agit de séparer les individus des cas d'infections, pour avoir un unique patient qui peut subir plusieurs infections, on créer une table temporaire pour ajouter les données et les trier :
- trier province/villes valides et les changer par le code postal (dans Region)
- changer le string age "50s" en integer 50.
- ajout des patients uniques dans la table Patient
- ajout des cas d'infection avec l'identité du patient responsable dans InfectionOrigin
- ajout lorsque l'identité n'est pas disponible dans Infection

Pour Region, un unique triplet code postal, province, ville
Pour Patient, un unique patient_id
Pour InfectionOrigin, un unique triplet patient_id, origine et date
Pour Infection, une unique paire par patient_id et date

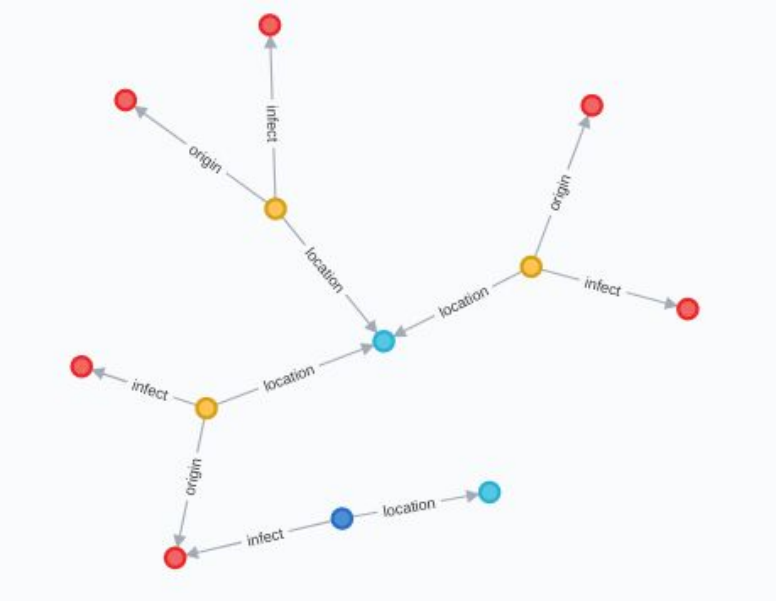
On écrit les tables dans des fichiers pour l'importation de données nettoyées sur Neo4j.

3 - Importation Neo4j

Comme pour Postgres :
- unique triplet pour Region avec Node key
- unique patient_id pour Patient avec Node key

Pour Infection et InfectionOrigin, il faut merge les nodes Region , Patient.
Ainsi chaque node Infection est relié exactement à un Patient et Region, et pour InfectionOrigine les même relations plus le noeud Patient causant l'infection.

La proriété contact_number est erroné (patient_id à la place d'un nombre réaliste), il faut aussi combler les null pour les requêtes Cypher.



4 - Index Neo4j

Application des index sur la requête 1:

```
CREATE INDEX idx_age FOR (p:Patient) ON (p.age);
CREATE INDEX idx_elem FOR (r:Region) ON (r.elementary_school_count);
CREATE INDEX idx_kdg FOR (r:Region) ON (r.kindergarten_count);

MATCH p=(a:Patient)<-[:infect]-(i)-[:location]->(r:Region)
WHERE a.age > 50 AND (r.elementary_school_count > 150 OR r.kindergarten_count > 150)
RETURN p limit 100
```

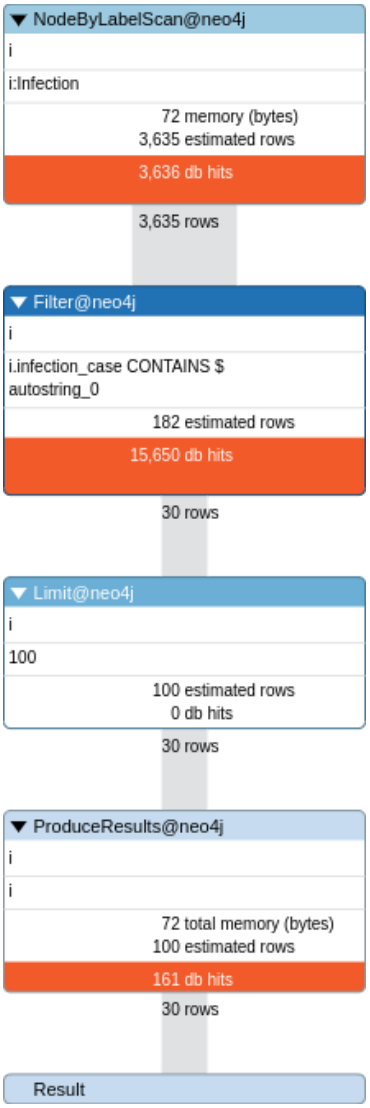
On peut remarquer l'efficacité des index sur les ranges avec 245 bd hits sans, et 15/22 db hits avec, l'opération OR est aussi moins coûteuse .

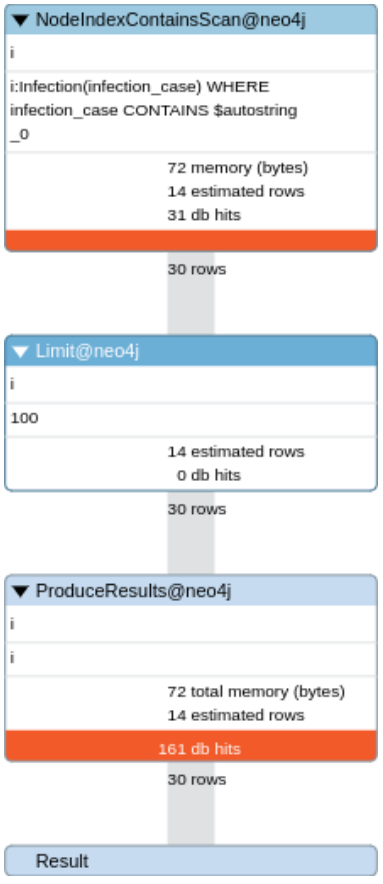
Pour la requête 0 :

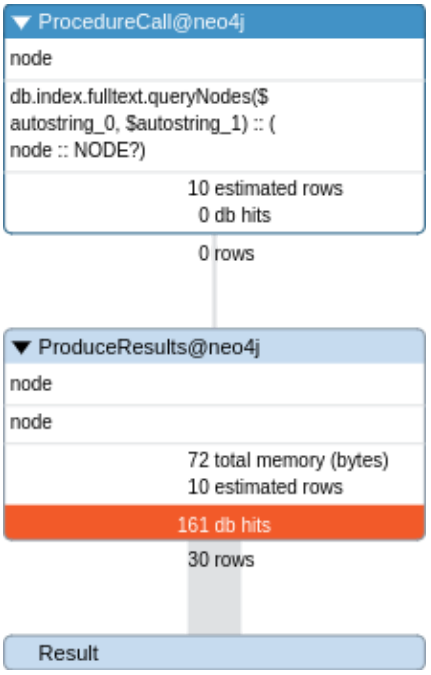
```
MATCH (i:Infection)
WHERE i.infection_case CONTAINS 'Hospital'
RETURN i limit 100
```

```
CALL db.index.fulltext.queryNodes("full_case", 'infection_case:"Hospital"') YIELD node
RETURN node
```

Il y a les plans d'exécutions sans index, avec btree et full-index. On peut voir le nombre d'étapes qui décroît avec le type d'index utilisé, cela signifie donc un nombre db hits minimum pour l'index full-text.



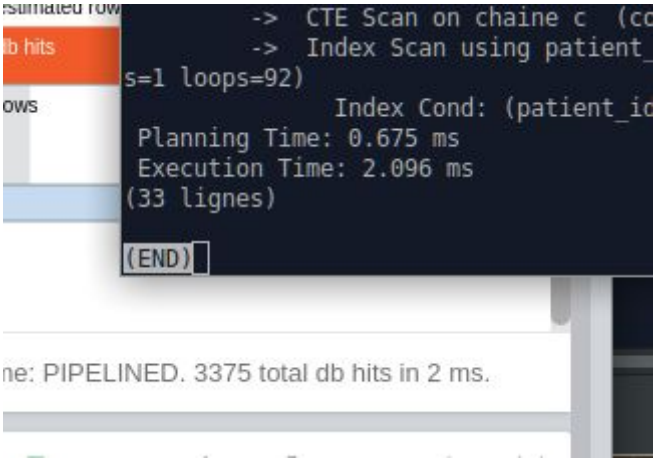


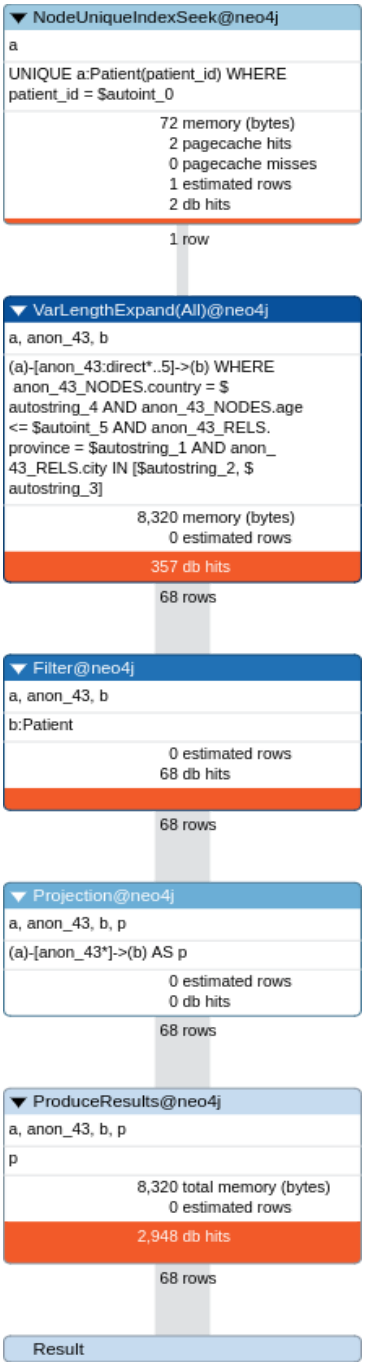


5 - Evaluation de chemin recursif

```
PROFILE
MATCH p=(a:Patient{patient_id:4100000006})-[:direct*..5]->(b:Patient)
WHERE ALL(one IN relationships(p) WHERE one.province='Chungcheongnam-do' AND (one.city='Asan-si' OR one.city='Cheonan-si'))
AND ALL(one IN nodes(p) WHERE one.country='Korea' AND one.age<=50)
RETURN p

EXPLAIN ANALYZE
WITH RECURSIVE chaine(victime, n_origine) AS (
   VALUES(4100000006::BIGINT, 0)
   UNION
   SELECT i.patient_id, n_origine + 1
   FROM chaine c, infectionorigin i, region r, patient p
   WHERE c.victime=i.infected_by AND i.code=r.code AND c.victime=p.patient_id
   AND p.country='Korea' AND r.province='Chungcheongnam-do'
   AND (r.city='Asan-si' OR r.city='Cheonan-si') AND p.age<=50
   AND i.patient_id<>4100000006
   AND n_origine < 5
)
SELECT *
FROM chaine c, patient p
WHERE c.victime=p.patient_id
ORDER BY victime;
```





Les temps d'exécution sont à peu près similaires, cela peut s'expliquer par les index présent en Postgres alors qu'ils sont absents sur la relationship :direct en Neo4j. Il y aussi la taille peu imposante du dataset et des relations possibles.

Note : La quantité de mémoire utilisée est plus importante en Postgres (Sort Method: quicksort Memory: 32kB) comparé à Neo4j (8,320 total memory (bytes)).

6 - GDS Applications

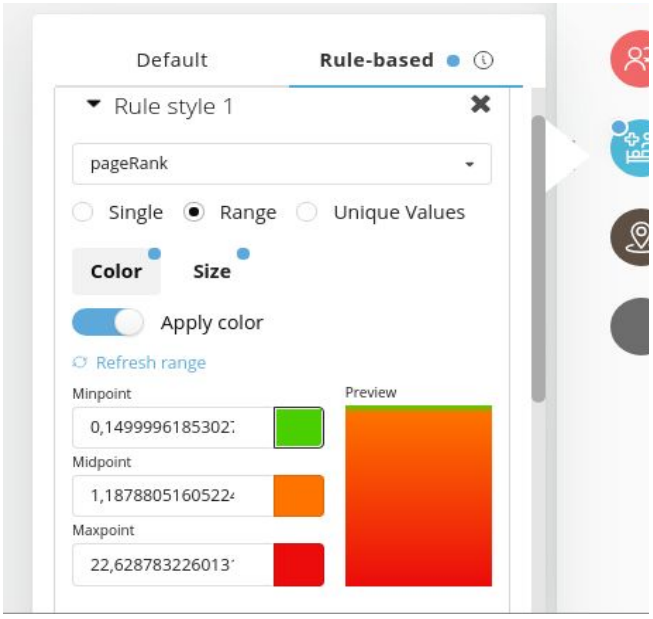
On applique l'algorithme pageRank, puisque cela permet de voir les potentiels foyers de contamination par Patient.

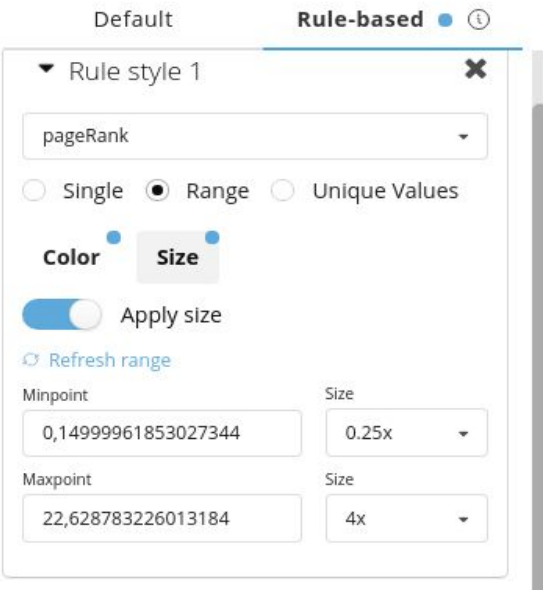
On crée tout d'abord des graphes nommés en méthode native et cypher avec la relation :direct , on remarque déjà le temps de création avec 26ms en native et 499ms en cypher.

```
CALL gds.graph.create( 'infection_direct', 'Patient',
   {direct:{
            type: 'direct', orientation: 'UNDIRECTED'
    }}
)YIELD graphName, nodeCount, relationshipCount, createMillis;
```

```
CALL gds.pageRank.write('infection_direct', {writeProperty: 'pageRank'})
YIELD writeMillis, nodePropertiesWritten, ranIterations, didConverge, centralityDistribution
RETURN centralityDistribution.max, centralityDistribution.p95, centralityDistribution.min
```

On récupère le max, le centile 95 et le min du pageRank pour le range dans rule-based style permettant de régler la couleur et la taille de l'affichage.





Le nombre de patients est trop élevé pour afficher directement l'ensemble des patients (limité à 1000 par défaut) et l'affichage devient très lent. On sélectionne un nombre $max de Patient avec le pageRank les plus élevés.

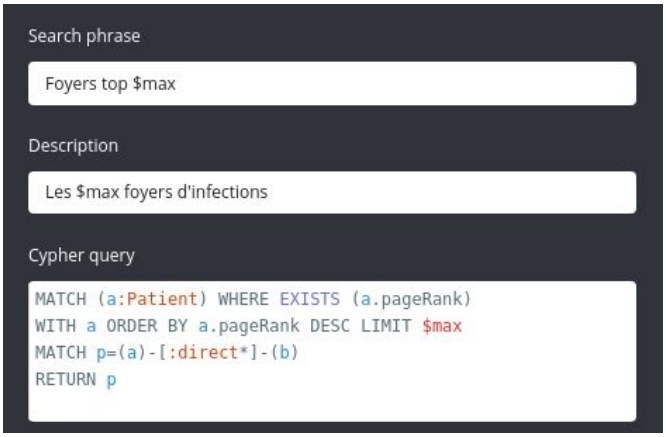
Pour appliquer une requête Cypher sur Bloom, il faut passer par le Static Search Phrase. Cela permet à Bloom d'associer une phrase à un query passé en objet.

Ainsi, on associe la phrase :

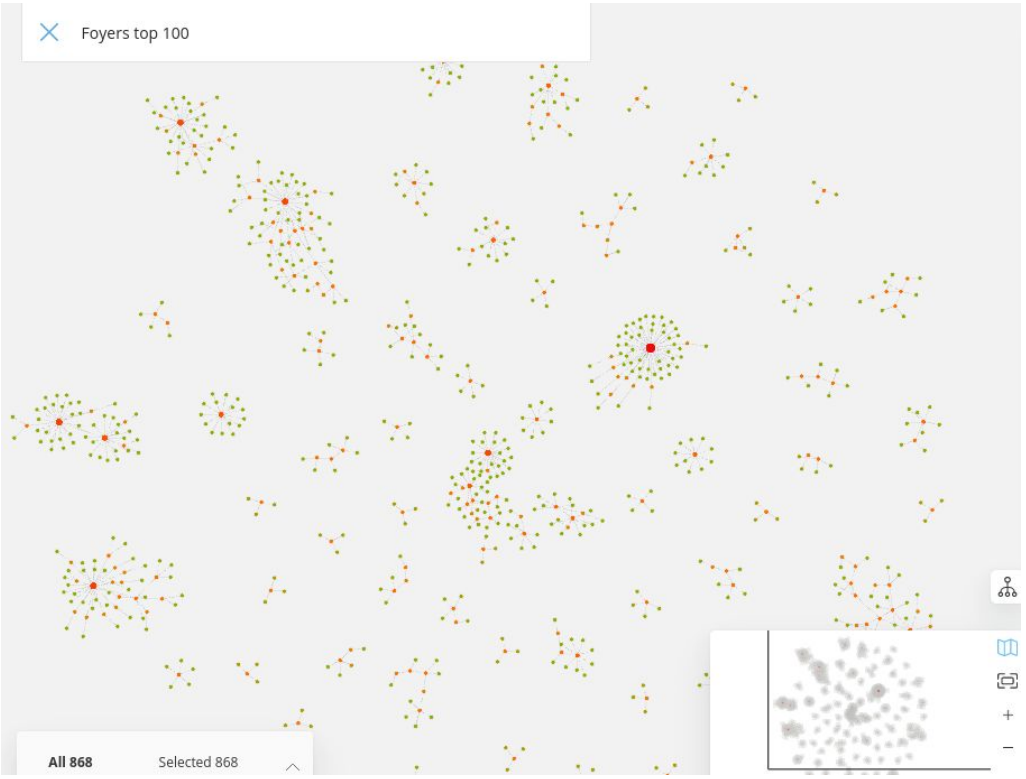
Foyers top $max

à la requête

```
MATCH (a:Patient) WHERE EXISTS (a.pageRank)
WITH a ORDER BY a.pageRank DESC LIMIT $max
MATCH p=(a)-[:direct*]-(b)  RETURN p
```



Il suffit d'appliquer la phrase dans la barre de recherche pour avoir l'affichage des 10 clusters :



On peut remarquer qu'il y a quelques cas de transmissions par foyers mais moins je ne le pensais. Peut-être que les consignes de sécurité sont respectées par la populations?
On veut maintenant afficher les villes les plus infectées et la migration du virus entre ces villes. On crée un graphe avec les page par rapport aux lieux d'infections.

```
 CALL gds.graph.create(
   'region_inf',
   ['Region','Infection','InfectionOrigin'],
   {location:{
            type: 'location',
            orientation: 'UNDIRECTED'
    }}
)YIELD graphName, nodeCount, relationshipCount, createMillis;
```

On applique pageRank comme pour l'exemple avec Patient :

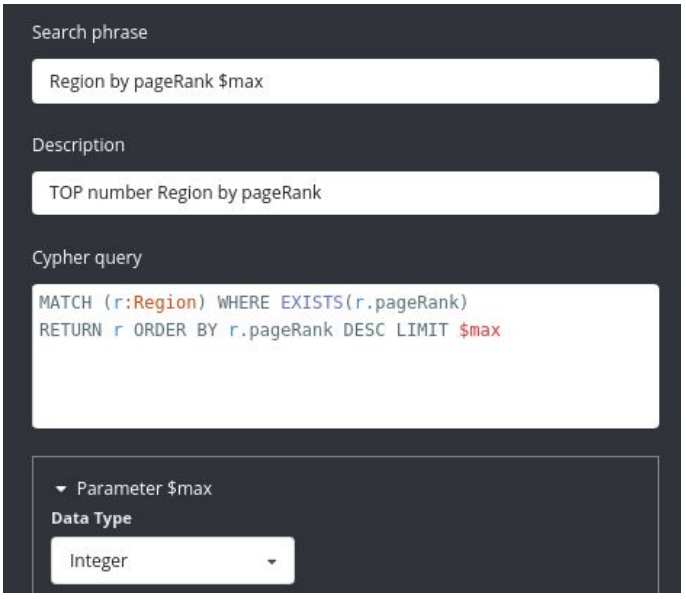
```
CALL gds.pageRank.write('region_inf', {writeProperty: 'pageRank'})
YIELD writeMillis, nodePropertiesWritten, ranIterations, didConverge, centralityDistribution
RETURN centralityDistribution.max, centralityDistribution.min, centralityDistribution
//max : 281.8574209213257 ,  min :0.9670543998479844
```

L'algorithme écrit les pageRank pour Infection et InfectionOrigin, mais ces nœuds ont une relation :location au plus, ils ont donc la valeur minimale de pageRank (0.14) et ne se mélangent pas avec les villes.
Le min(pageRank)=0.967054 des villes est récupéré avec une recherche simple. Les valeurs de centile pour la couleur du milieu sont trop petite pageRank > 10, on prend donc valeur du milieu = 100.

Puis on crée une nouvelle relation :migration avec le poids selon le nombre de transmission entre chaque ville :

```
MATCH p=(r1:Region)<--(i1:InfectionOrigin)-->(a:Patient)<--(i2:InfectionOrigin)-->(r2:Region)
WHERE r1.code<>r2.code
WITH r1,r2,count(*) AS weight
MERGE p=(r1)-[:migration{weight:weight}]-(r2)
RETURN p
```

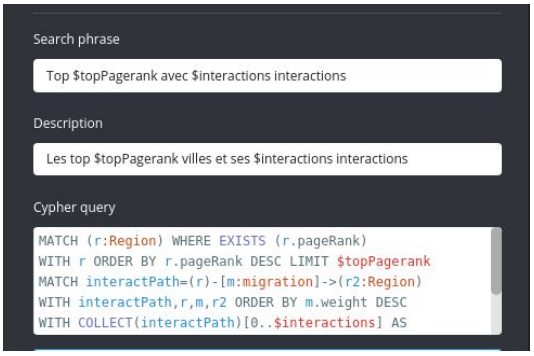
Pour voir les interactions on affiche d'abord les villes avec top pageRank sans les relations avec une variable dans la requête ("Region by pageRank 50" pour top 50) :



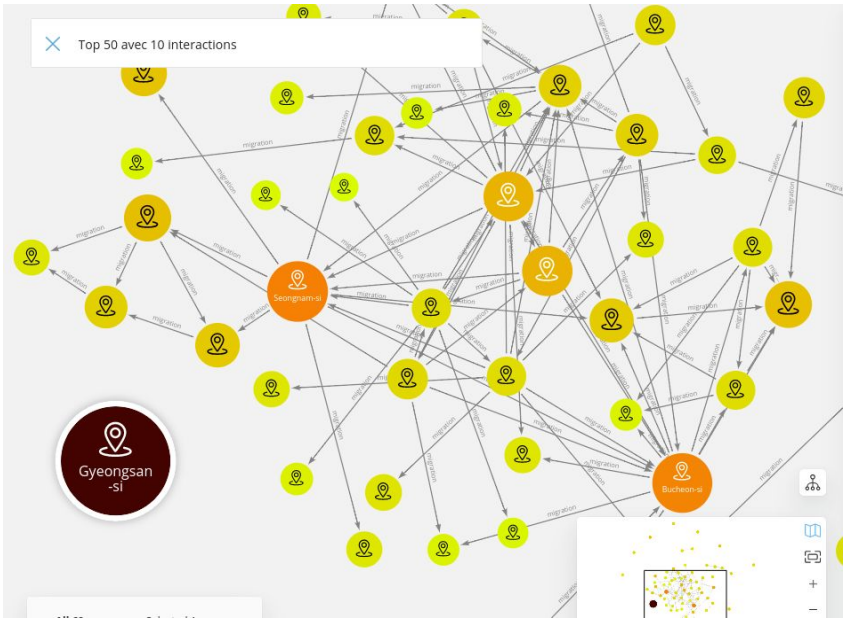
Puis on filtre ces villes si elles ont une relation :migration et on garde les relations avec les poids les plus important :

```
MATCH (r:Region) WHERE EXISTS (r.pageRank)
WITH r ORDER BY r.pageRank DESC LIMIT $topPagerank
MATCH interactPath=(r)-[m:migration]->(r2:Region)
WITH interactPath,r,m,r2 ORDER BY m.weight DESC
WITH COLLECT(interactPath)[0..$interactions] AS topInteractions, r
RETURN topInteractions
```

On ajoute la requête sur Bloom :



On peut effectuer la recherche les top 50 villes infectées avec leurs 10 relations :



On peut de suite voir que la ville avec le plus de cas d'infection (600~) n'a pas de connexion avec une autre ville. Cela peut-être dû aux lieux d'enseignements ?
Gyeongsangbuk-do,Gyeongsan-si,31,61,10 (primaires, maternelles, universités)
Gyeongsangbuk-do,Gyeongsangbuk-do,471,707,33

Gyeongsan-si est loin d'avoir un grand nombre de lieux d'éducation mais il y en a relativement assez pour être un foyer.

On peut conclure que les villes avec de nombreux cas d’infection n’ont pas toujours le plus de migration.