TP2 Analyse de Graphes de données Neo4j : la librairie DATA SCIENCE

TANG Kevin p1501263

VINCENT Yann p1906701

KIARED Siham p2213168

**1. Donner les statistiques suivantes en utilisant les connaissances acquises durant le TP1:**

**a. Donner la valeur minimale, maximale, moyenne ainsi que l'écart type des interactions entre personnes (aucun algorithme de la GDS n’est nécessaire, il suffit d’avoir le degré).**

match (u:User)-[r]-> ()

with u as node, count(r) as degree

return min(degree), max(degree), avg(degree), stdev(degree)

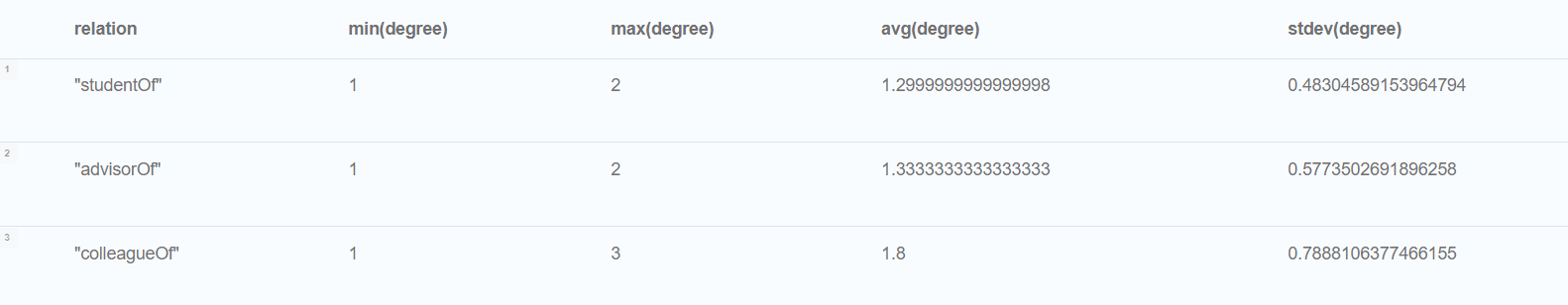


**b. Donner le degré min, max, moyen ainsi que l''écart type par type de relation (la relation KNOWS possède plusieurs types : voir l’attribut type)**

match (u:User)-[r:KNOWS]-> ()

with u,r.type as relation , count(r) as degree

return relation, min(degree), max(degree), avg(degree), stdev(degree)



**2. Créer une projection du graphe en prenant uniquement les nœuds User, l’association KNOWS sans aucun attribut.**

Call gds.graph.project("Proj1", "User", "KNOWS")

**3. Quels est le chemin le plus court entre la personne ayant l'id 5 et celle ayant l'id 10 (en terme de nombre d'arêtes) : la requête doit retourner tous les nœuds sur ce chemin. Vous pouvez utiliser l’algorithme de Dijkstra**

match (u:User{id:5}), (u2:User{id:10})

call gds.shortestPath.dijkstra.stream('Proj1',

{ sourceNode:u,

targetNode:u2

})

Yield index, sourceNode, targetNode, totalCost, nodeIds, costs, path

RETURN

gds.util.asNode(sourceNode).name AS sourceNodeName,

gds.util.asNode(targetNode).name AS targetNodeName,

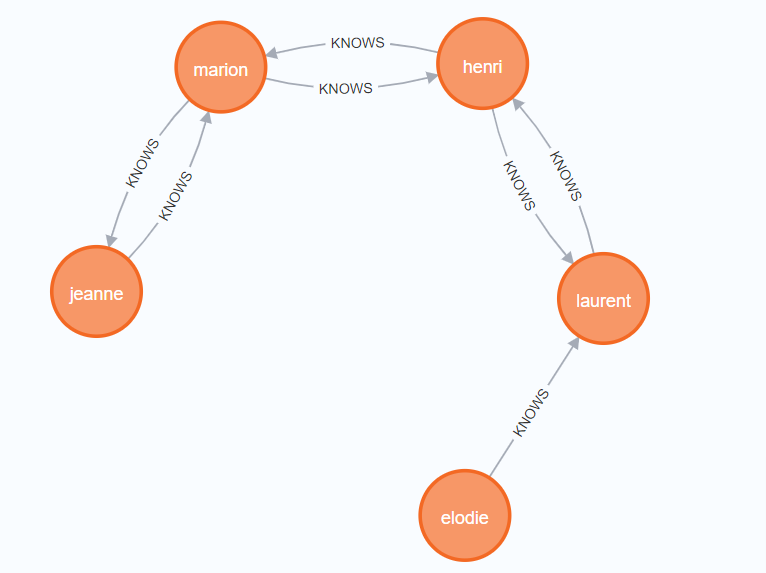
totalCost,

[nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS nodeNames,

costs,

nodes(path) as path

order by index



**4. Créer une projection du graphe en prenant uniquement les nœuds User, l’association KNOWS avec l’attribut weight.**

CALL gds.graph.project('Proj2', 'User', 'KNOWS', { relationshipProperties: 'weight' });

**5. Reprendre la question 3 en considérant la qualité du chemin (propriété weight).**

MATCH (u:User{id:5}), (u2:User{id:10})

CALL gds.shortestPath.dijkstra.stream('Proj2',

{

sourceNode: u,

targetNode: u2,

relationshipWeightProperty: 'weight' // Spécifie la propriété weight pour la pondération

})

YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path

RETURN

gds.util.asNode(sourceNode).name AS sourceNodeName,

gds.util.asNode(targetNode).name AS targetNodeName,

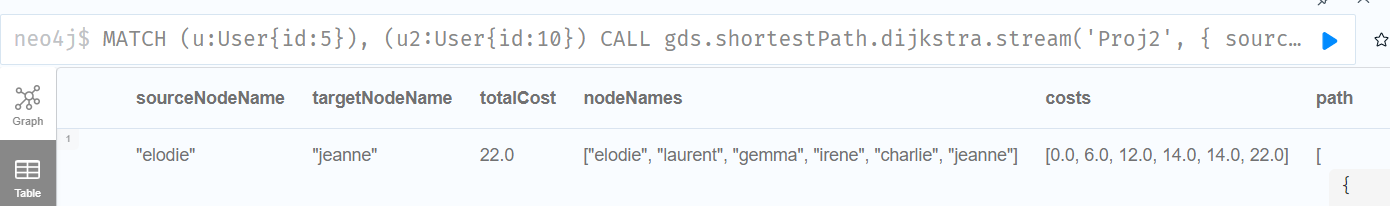
totalCost,

[nodeId IN nodeIds | gds.util.asNode(nodeId).name] AS nodeNames,

costs,

nodes(path) AS path

ORDER BY index;

****

**6. Donnez la distance en terme de nombre d’arêtes entre chaque paire de nœuds.**

match (u:User), (u2:User)

where u<> u2

call gds.shortestPath.dijkstra.stream('Proj1',

{ sourceNode:u,

targetNode:u2

})

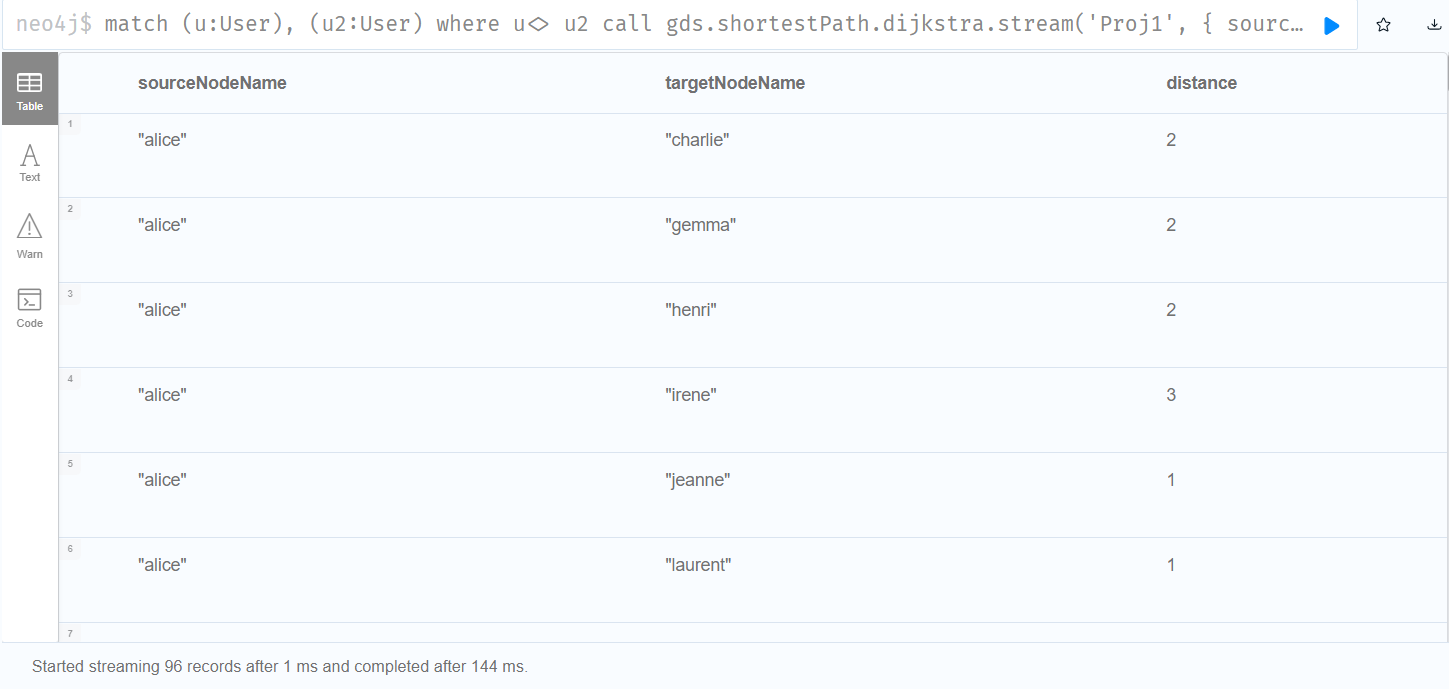
YIELD sourceNode, targetNode, path

RETURN

gds.util.asNode(sourceNode).name AS sourceNodeName,

gds.util.asNode(targetNode).name AS targetNodeName,

length(path) AS distance



**7. Reprendre la question précédente en prenant en considération les poids des arêtes**

match (u:User), (u2:User)

where u<> u2

call gds.shortestPath.dijkstra.stream('Proj2',

{ sourceNode:u,

targetNode:u2,

relationshipWeightProperty: 'weight'

})

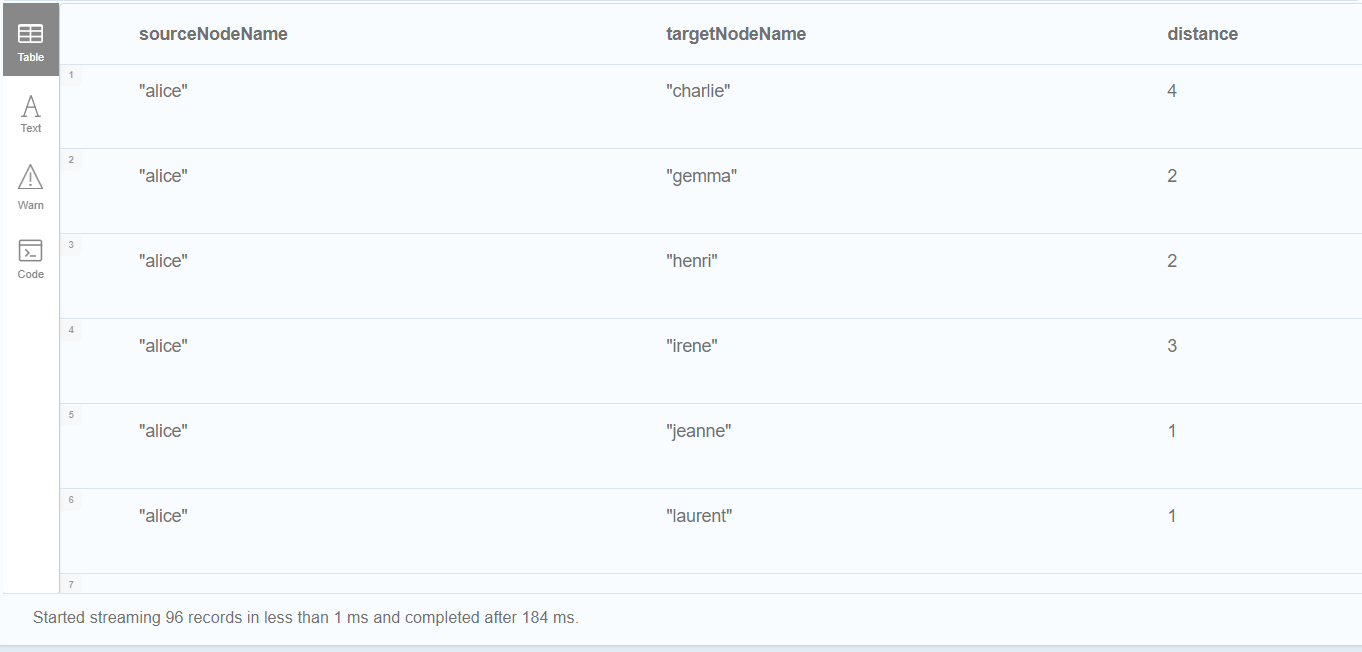
YIELD sourceNode, targetNode, path

RETURN

gds.util.asNode(sourceNode).name AS sourceNodeName,

gds.util.asNode(targetNode).name AS targetNodeName,

length(path) AS distance



**8. Rajouter une propriété « cc » aux nœuds du graphe, elle permet de spécifier la composante connexe à laquelle appartient chaque nœud ensuite afficher les nœuds avec leur composante.**

call gds.wcc.write('Proj2', {writeProperty: 'cc'});

**a. Affichez les partitions avec pour chacune sa taille et la liste des noms des personnes.**

MATCH (u:User)

RETURN u.cc AS composanteConnexe, COUNT(u) AS nbUsers, COLLECT(u.name) AS nomsUsers

ORDER BY nbUsers DESC



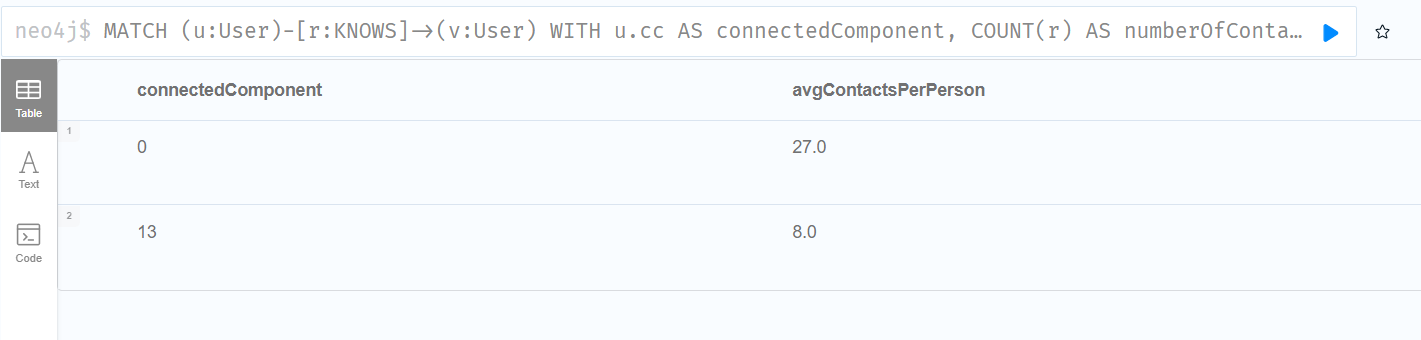
**b. Donner le nombre moyen de contacts par personne pour chaque composante connexe.**

MATCH (u:User)-[r:KNOWS]->(v:User)

WITH u.cc AS connectedComponent, COUNT(r) AS numberOfContacts

RETURN connectedComponent, AVG(numberOfContacts) AS avgContactsPerPerson

ORDER BY connectedComponent;

****

**c. Affichez les résultats obtenus par les autres algorithmes de détection de communautés (Louvain et Label Propagation). Est-ce que ces deux algorithmes retournent la même chose ?**

**Louvain:**

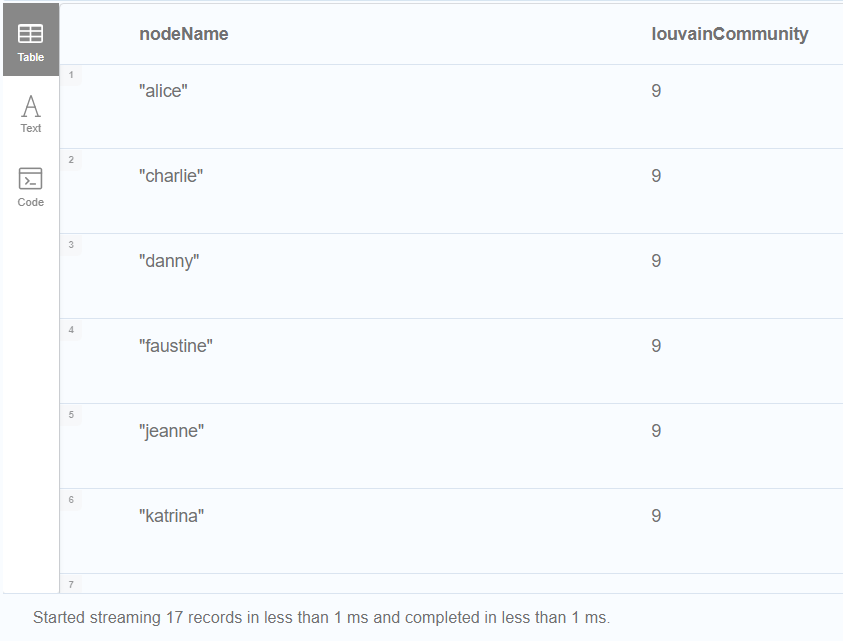
CALL gds.louvain.write('Proj2', { writeProperty: 'louvainResult' })

YIELD communityCount, modularity;

MATCH (u:User)

RETURN u.name AS nodeName, u.louvainResult AS louvainCommunity

ORDER BY louvainCommunity;



**Label Popagation :**

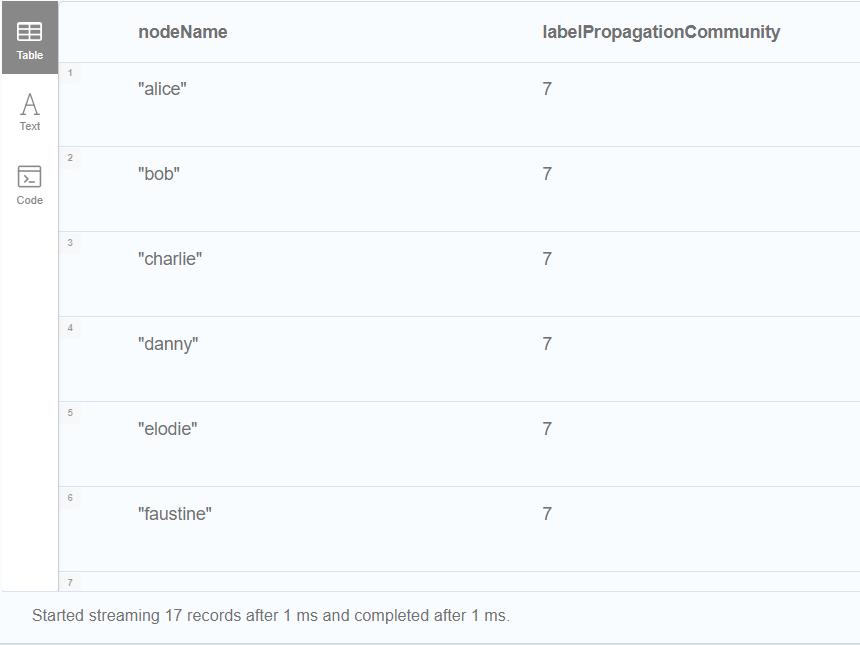
CALL gds.labelPropagation.write('Proj2', { writeProperty: 'labelPropagationResult' })

YIELD communityCount, ranIterations;

MATCH (u:User)

RETURN u.name AS nodeName, u.labelPropagationResult AS labelPropagationCommunity

ORDER BY labelPropagationCommunity;



Ne renvoi pas la même chose

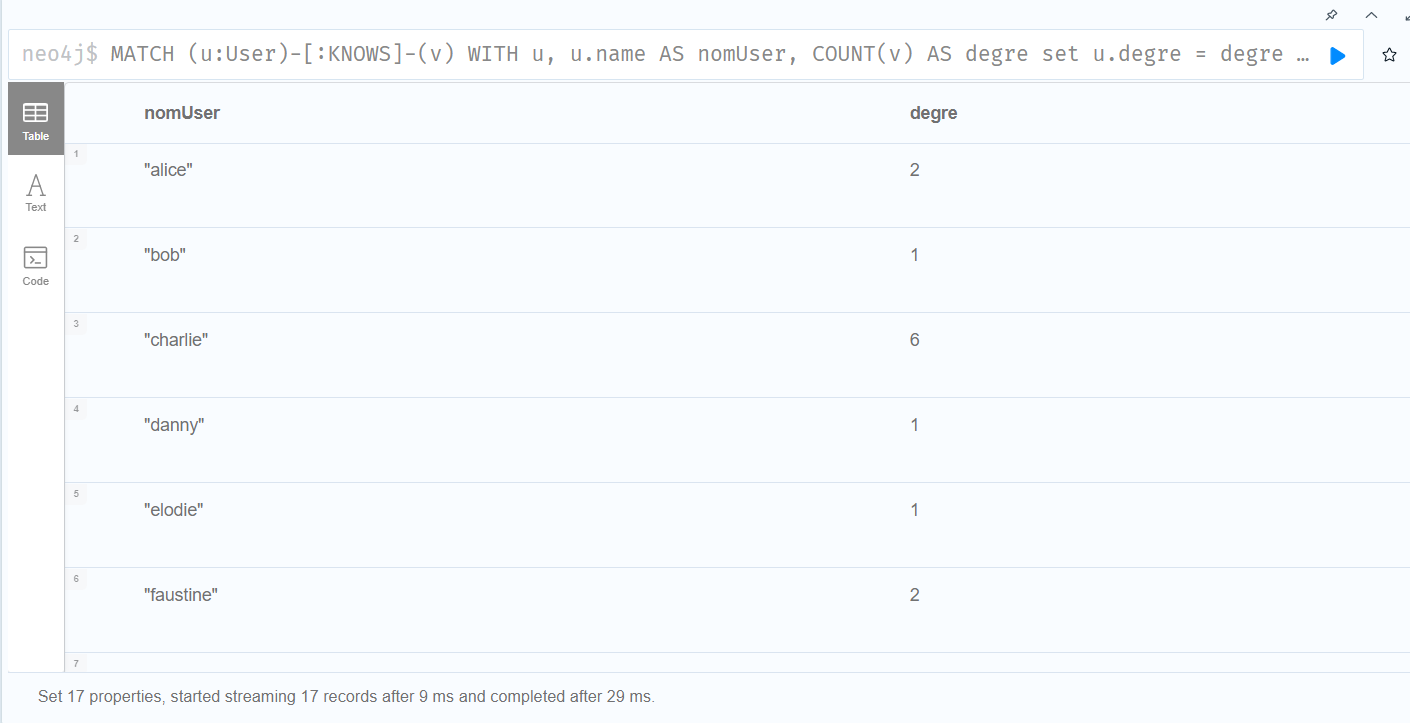
**9. L’importance d’un nœud dans un réseau peut être mesurée par le nombre d'interactions qu’il a avec les autres nœuds, donc son degré, rajoutez aux nœuds une telle propriété et affichez-la pour toutes les personnes.**

MATCH (u:User)-[:KNOWS]-(v)

WITH u, u.name AS nomUser, COUNT(v) AS degre

set u.degre = degre

return nomUser, degre

****

**10. Calculer le diamètre du graphe (le diamètre géodésique d'un graphe est défini comme le plus grand court chemin)**

MATCH (u:User), (u2:User)

CALL gds.shortestPath.dijkstra.stream('Proj2', {

sourceNode: u,

targetNode: u2,

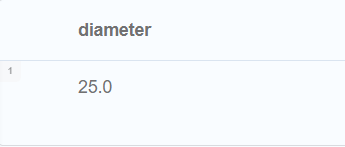
relationshipWeightProperty: 'weight'

})

YIELD sourceNode, targetNode, totalCost

WITH MAX(totalCost) AS diameter

RETURN diameter;

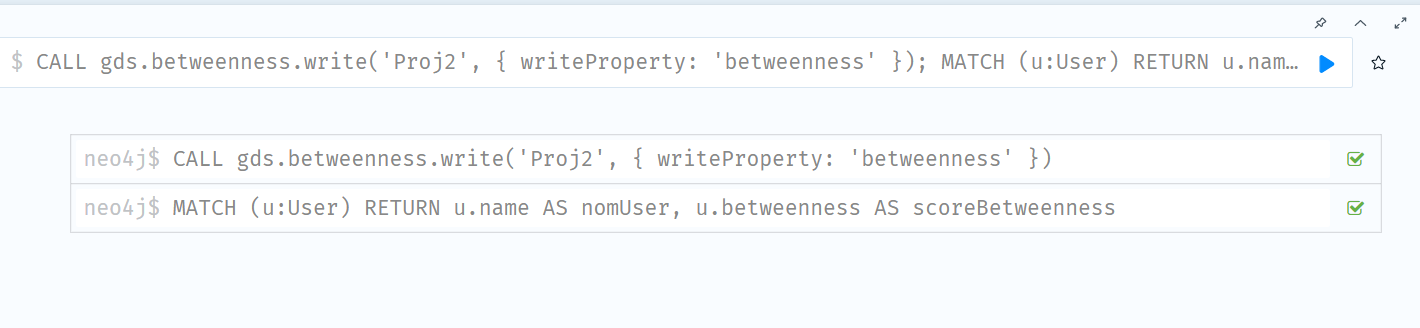


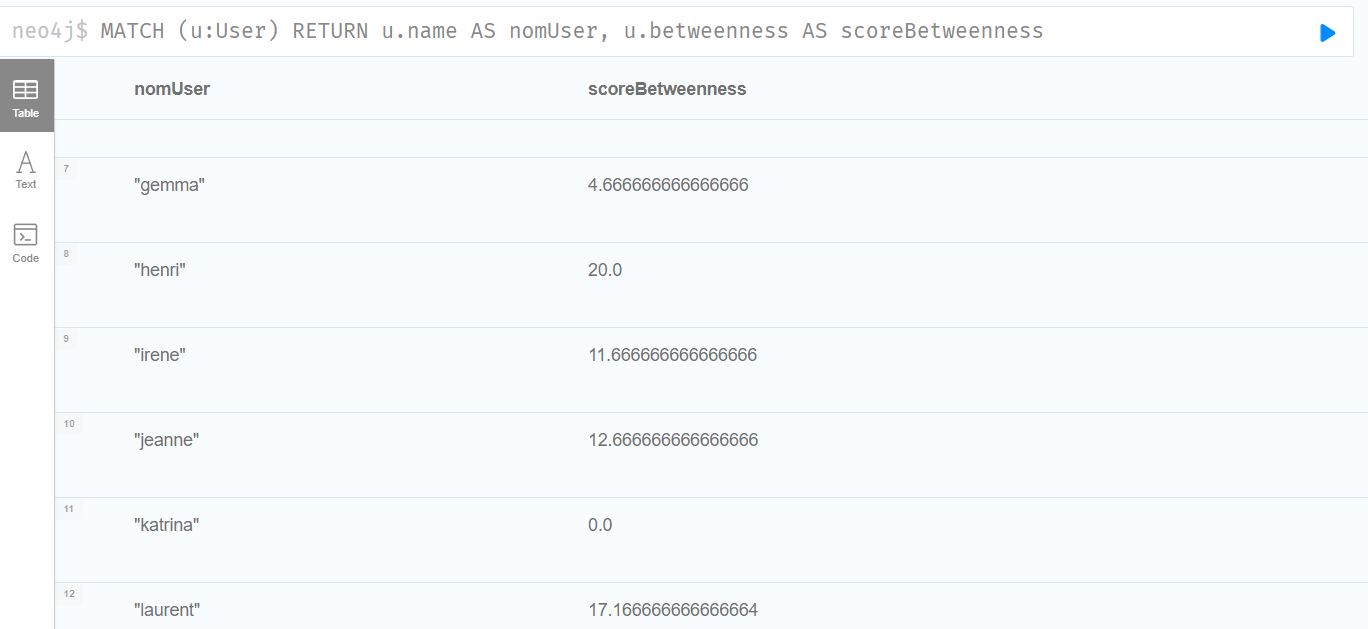
**11. Rajouter une propriété qui mesure combien un nœud est stratégique (un nœud est stratégique si les informations transitent souvent par lui. Une telle position intermédiaire donne à ce nœud pouvoir et influence : en général il est reconnu par le nombre de courts chemins passant par lui).**

CALL gds.betweenness.write('Proj2', { writeProperty: 'betweenness' });

MATCH (u:User)

RETURN u.name AS nomUser, u.betweenness AS scoreBetweenness





**12. Quels sont les nœuds qui ont les distances les plus courtes à tous les autres nœuds ?**

CALL gds.beta.closeness.stream('Proj2')

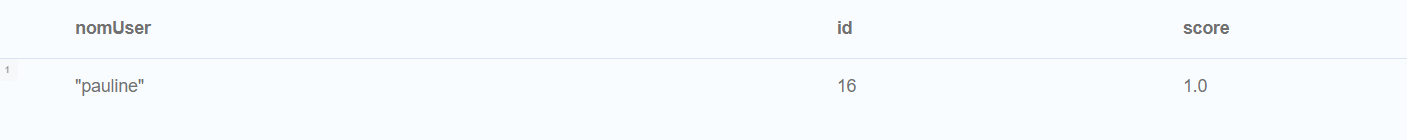
YIELD nodeId, score

WITH gds.util.asNode(nodeId) AS node, score

RETURN node.name AS nomUser, node.id AS id, score

ORDER BY score DESC

LIMIT 1



**13. D’après le résultat de la question précédente c’est Pauline qui est la plus proche de tous les autres ? comment rendre le résultat plus pertinent ?**

On pourrait afficher un graphe avec Pauline au centre avec une distance avec les autres nœuds selon les poids des arêtes.

**14. Rajoutez une propriété qui mesure l’efficacité avec laquelle chacun profite de ses relations et affichez le résultat pour tous nœuds (pagerank).**

CALL gds.pageRank.write('Proj2', {

maxIterations: 20,

dampingFactor: 0.85,

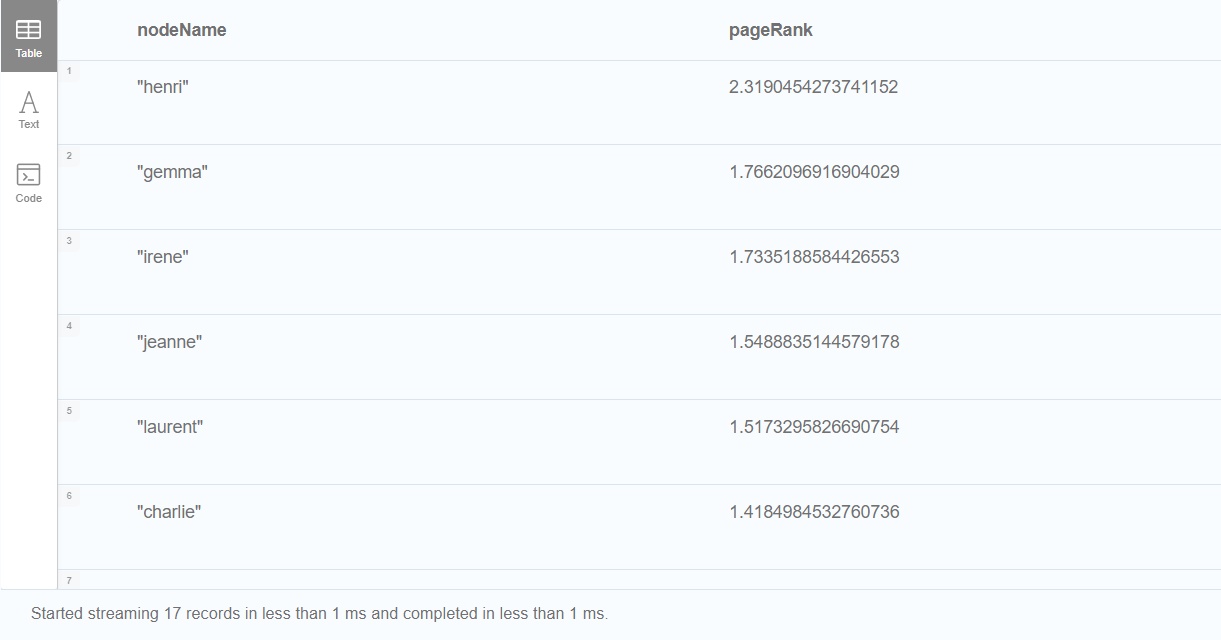
writeProperty: 'pagerank'

});

MATCH (u:User)

RETURN u.name AS nodeName, u.pagerank AS pageRank

ORDER BY u.pagerank DESC;



**15. Les propriétés rajoutées dans les questions 9, 11 et 14 mesurent chacune de manière différente et d’un point de vue différents l’importance d’un nœud ou son influence. Ecrivez une requête qui affiche les 3 valeurs.**

MATCH (u:User)

RETURN

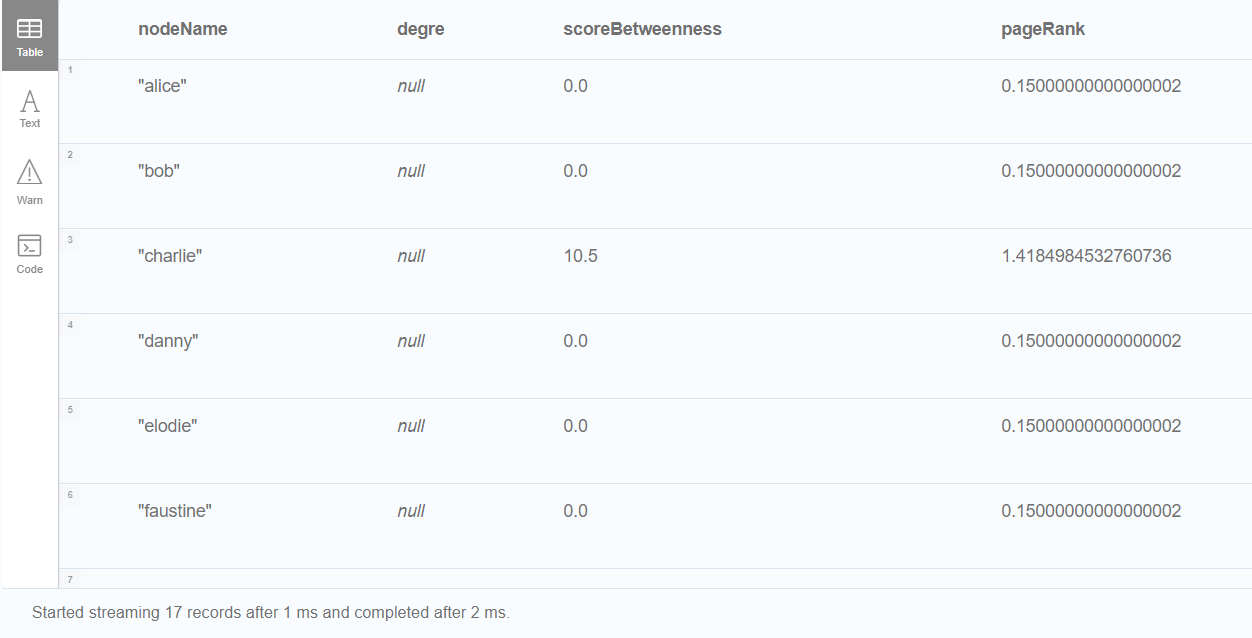
u.name AS nodeName,

u.degre AS degre,

u.betweenness AS scoreBetweenness,

u.pagerank AS pageRank

ORDER BY u.name;



**16. Combien de triangles on a dans ce graphe pour chaque nœud? calculer le coefficient de clustering du graphe.**CALL gds.beta.localClusteringCoefficient.write('Proj2', {

writeProperty: 'localClusteringCoefficient'

})

YIELD averageClusteringCoefficient, nodeCount

**17. Rajouter un nœud par composante connexe, ensuite une relation IN\_COMMUNITY allant d’un nœud vers sa composante connexe.**MATCH (u:User)

WITH u.cc AS composanteConnexe, COLLECT(u) AS nodes

FOREACH (node IN nodes |

MERGE (c:Community { name: 'Community ' + composanteConnexe })

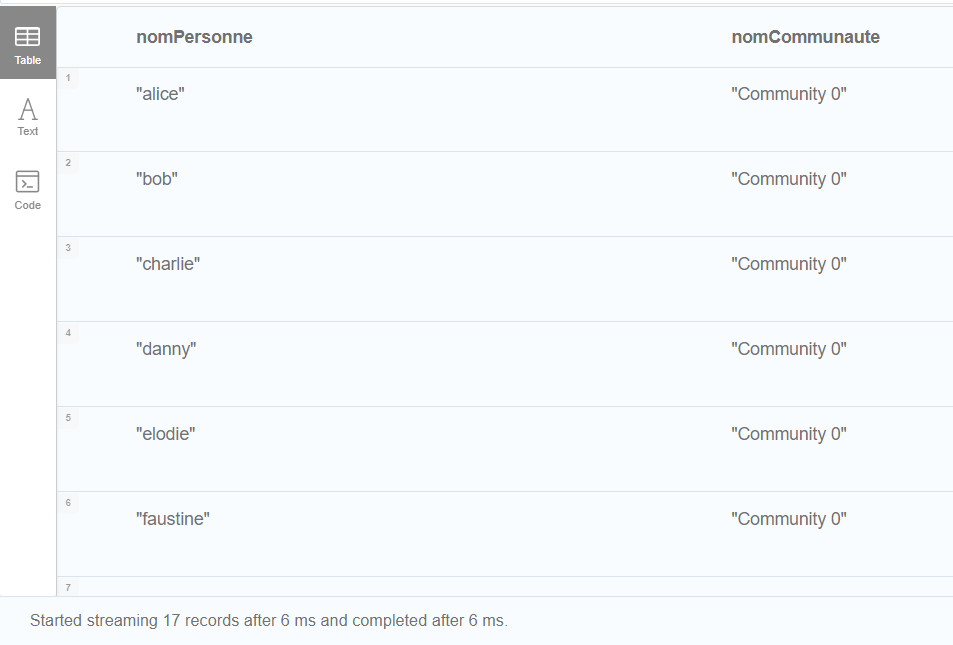
MERGE (node)-[:IN\_COMMUNITY]->(c)

);

MATCH (u:User)-[:IN\_COMMUNITY]->(c:Community)

RETURN u.name AS nomPersonne, c.name AS nomCommunaute

ORDER BY nomPersonne;



**18. Calculer le pagerank des nœuds par composante connexe.**

CALL gds.pageRank.write('Proj2', { writeProperty: 'pageRank', maxIterations: 20, relationshipWeightProperty: 'weight' });

MATCH (u:User)

RETURN u.cc AS composanteConnexe, u.name AS nomPersonne, u.pageRank AS pagerank

ORDER BY composanteConnexe, pagerank DESC;

