Ashley Music

June 25, 2020

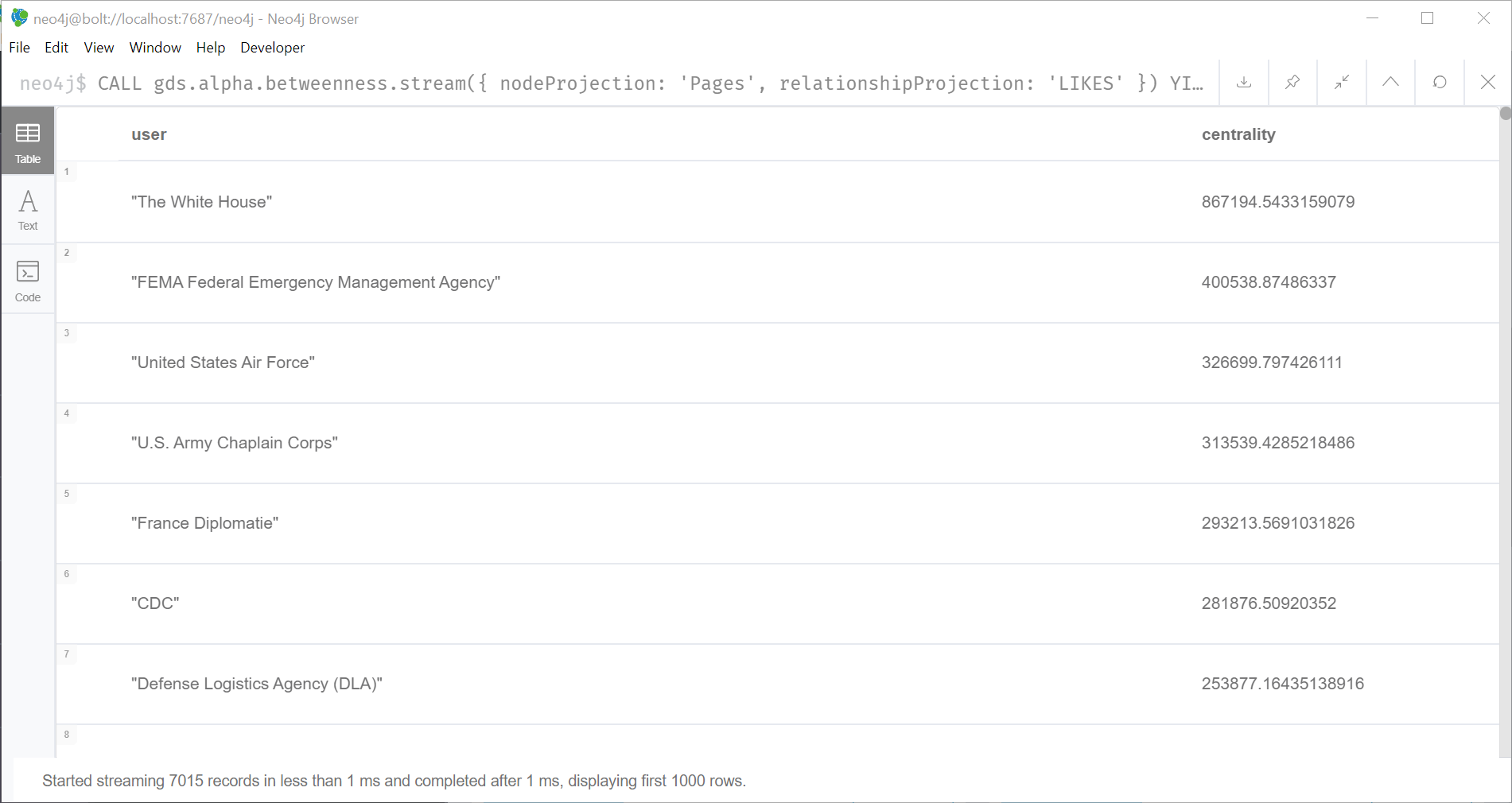
Week 8 – Final Lab

BDAT 365

**Choose a graph algorithm that you have used in this course and run the dataset through it in Neo4j. These algorithms could be any of the following: Centralities, Community Detection, Pathfinding, Triangle Count, Label Propagation, or Louvain.**

I am continuing to use the data from Lab 3, [The Network Repository](http://networkrepository.com/fb-pages-government.php) , that represents a community of Government Facebook Pages and the mutual likes between them. The nodes represent government Facebook pages and the edges represent the mutual likes between those Facebook pages. You even see some recursive relationships in this graph – the Facebook page likes itself. I think this represents a great social network and wanted to run some additional algorithms on this data.

I’d like to run some centrality algorithms on this data, so I’ll start with the betweenness algorithm.

****

**Analysis:**

Not surprisingly, we see that the White House has the biggest centrality (or as the [Neo4j site](https://neo4j.com/docs/graph-data-science/current/algorithms/betweenness-centrality/) calls it, the “main broker in this network”. We can see many government pages have influence based on the betweenness centrality weight.

**Code:**

CALL gds.alpha.betweenness.stream({

nodeProjection: 'Pages',

relationshipProjection: 'LIKES'

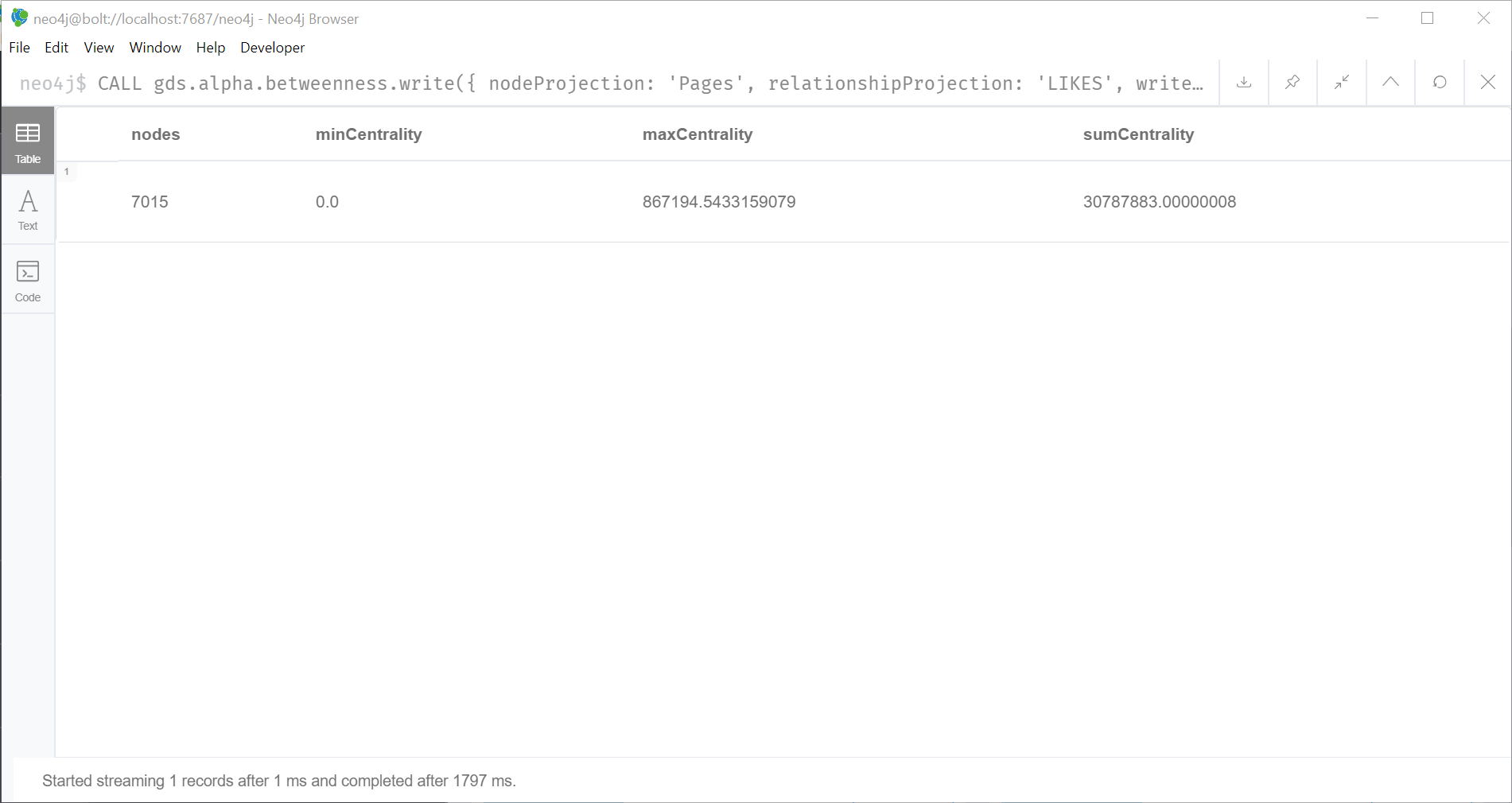
})

YIELD nodeId, centrality

RETURN gds.util.asNode(nodeId).name AS user, centrality

ORDER BY centrality DESC

Running the algorithm and writing back results:



**Analysis:**

We can see that we looked at 7015 nodes and found the max centrality weight to be 867194.5433 and the sum centrality weight to be 30787883.00000

**Code:**

CALL gds.alpha.betweenness.write({

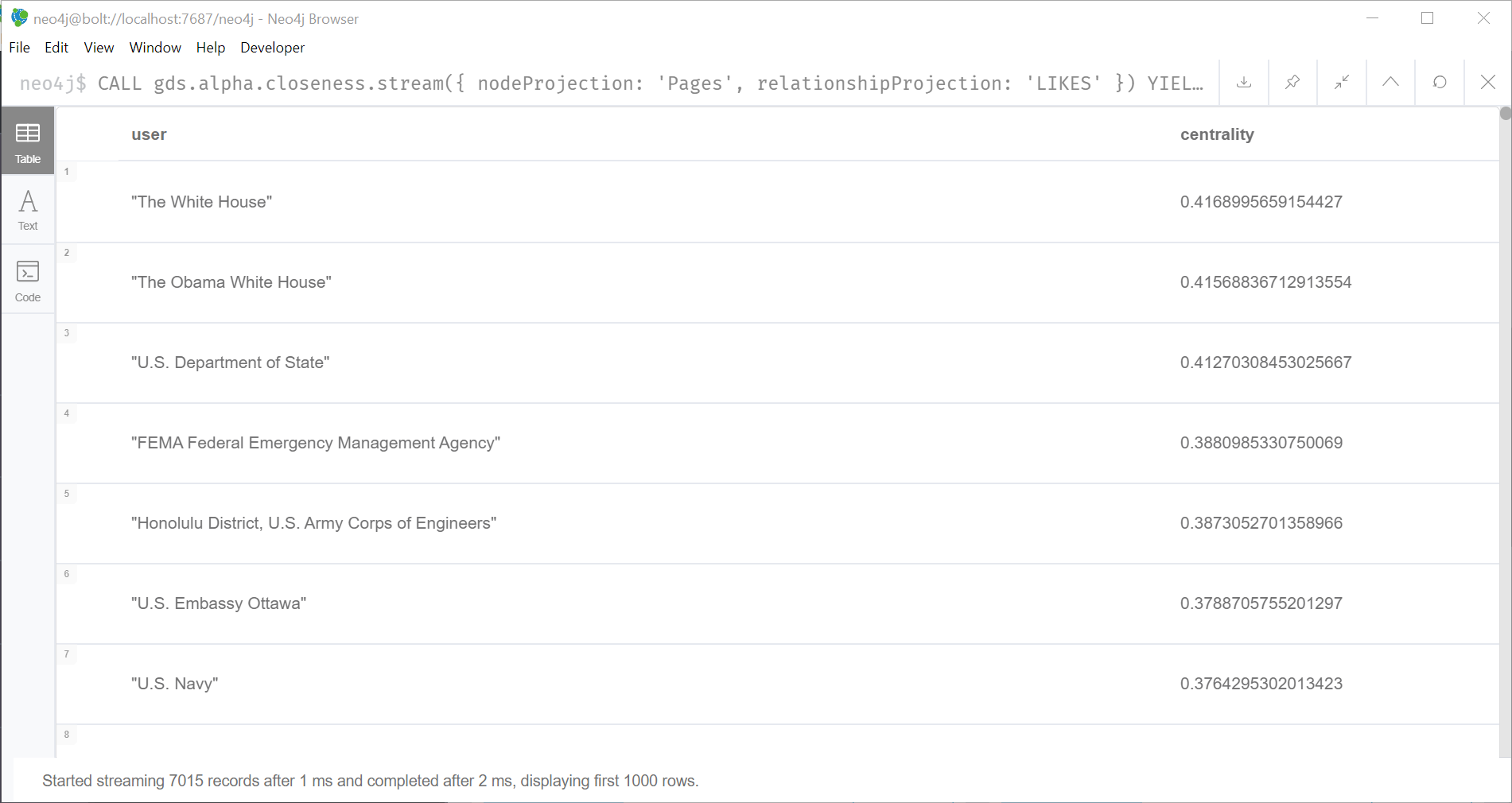
nodeProjection: 'Pages',

relationshipProjection: 'LIKES',

writeProperty: 'centrality'

}) YIELD nodes, minCentrality, maxCentrality, sumCentrality

I decided to go ahead and run a few other centrality algorithms on the data. Here is the closeness centrality analysis.



**Analysis:**

Here we see that the White House again has the highest closeness centrality weight. We also see a new one come up called, the Obama White House.

**Code:**

CALL gds.alpha.closeness.stream({

nodeProjection: 'Pages',

relationshipProjection: 'LIKES'

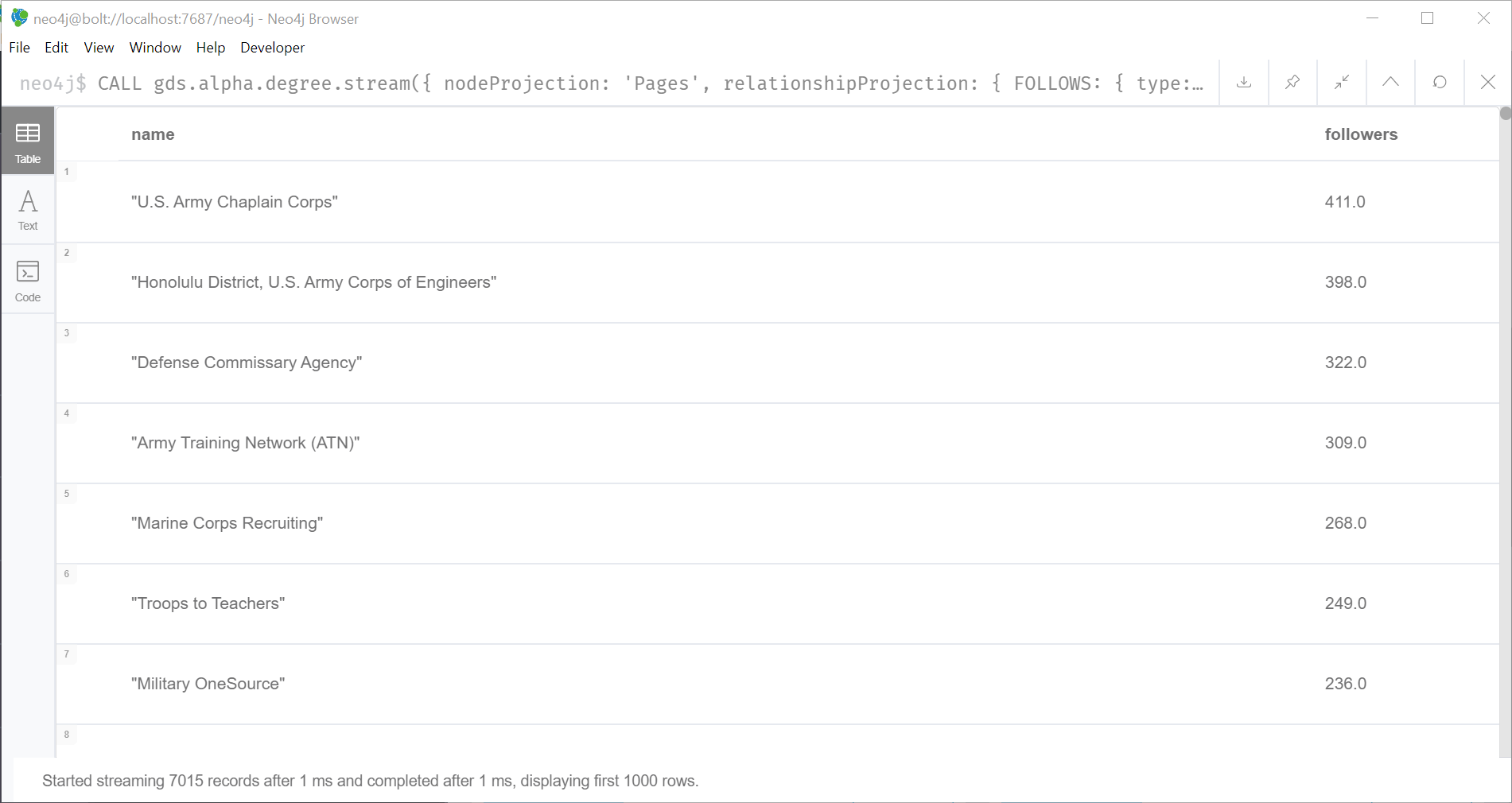
})

YIELD nodeId, centrality

RETURN gds.util.asNode(nodeId).name AS user, centrality

ORDER BY centrality DESC

I wanted to see if I could find the most popular node (and I’m assuming it will be the White House again) so now I’ll run the degree centrality.



**Analysis:**

I was incorrect! The most popular node is the U.S. Army Chaplain Corps. The next most popular was the Honolulu District, U.S. Army Corps of Engineers. Very interesting and something I did not expect.

**Code:**

CALL gds.alpha.degree.stream({

nodeProjection: 'Pages',

relationshipProjection: {

FOLLOWS: {

type: 'LIKES',

projection: 'REVERSE'

}

}

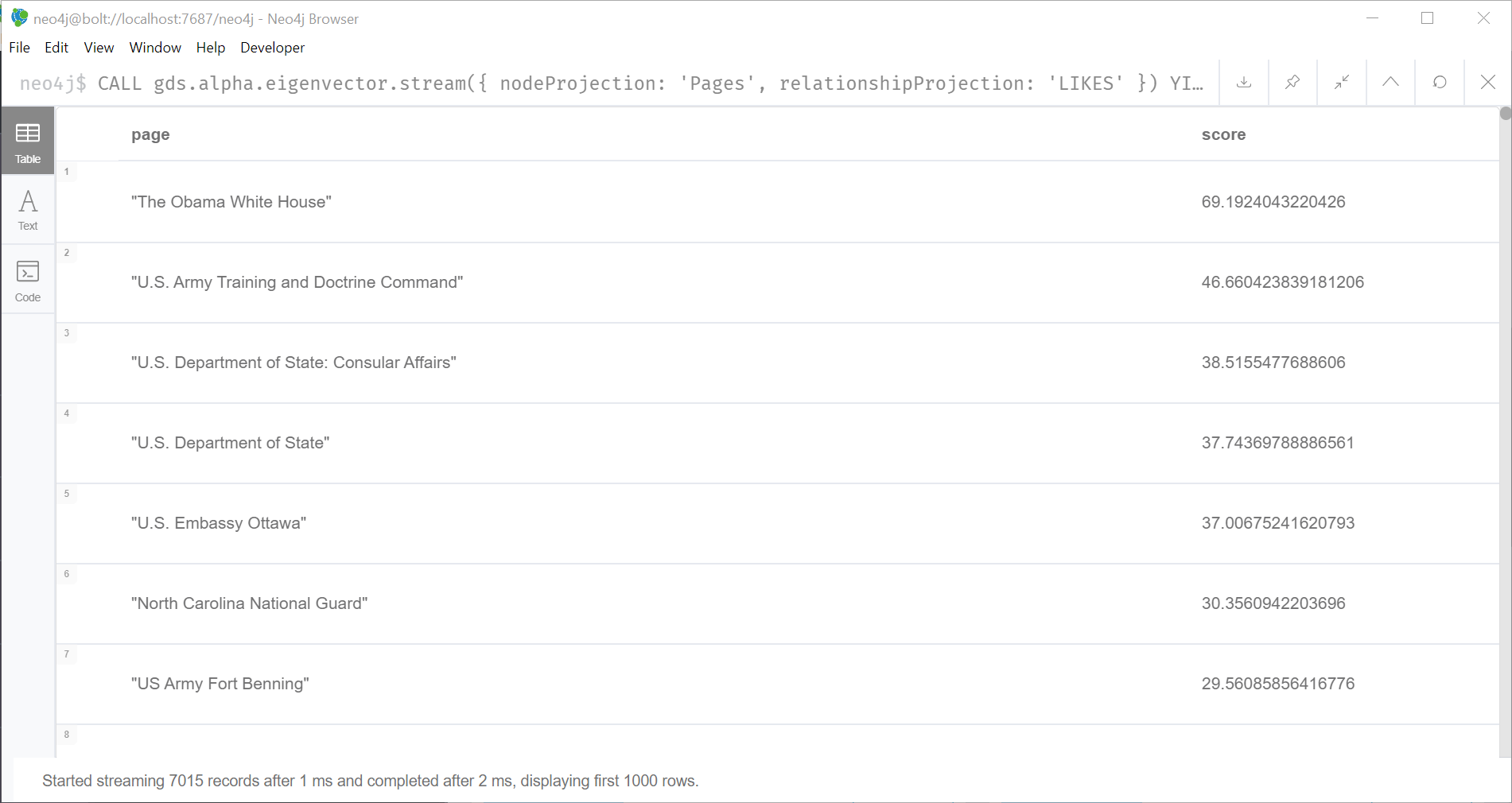
})

YIELD nodeId, score

RETURN gds.util.asNode(nodeId).name AS name, score AS followers

ORDER BY followers DESC

To finish off they analysis I decided to run the Eigenvector Centrality to see if there are any relationships between high-scoring nodes.



**Analysis:**

The Obama White House shows up again here with the highest Eigenvector score.

**Code:**

CALL gds.alpha.eigenvector.stream({

nodeProjection: 'Pages',

relationshipProjection: 'LIKES'

})

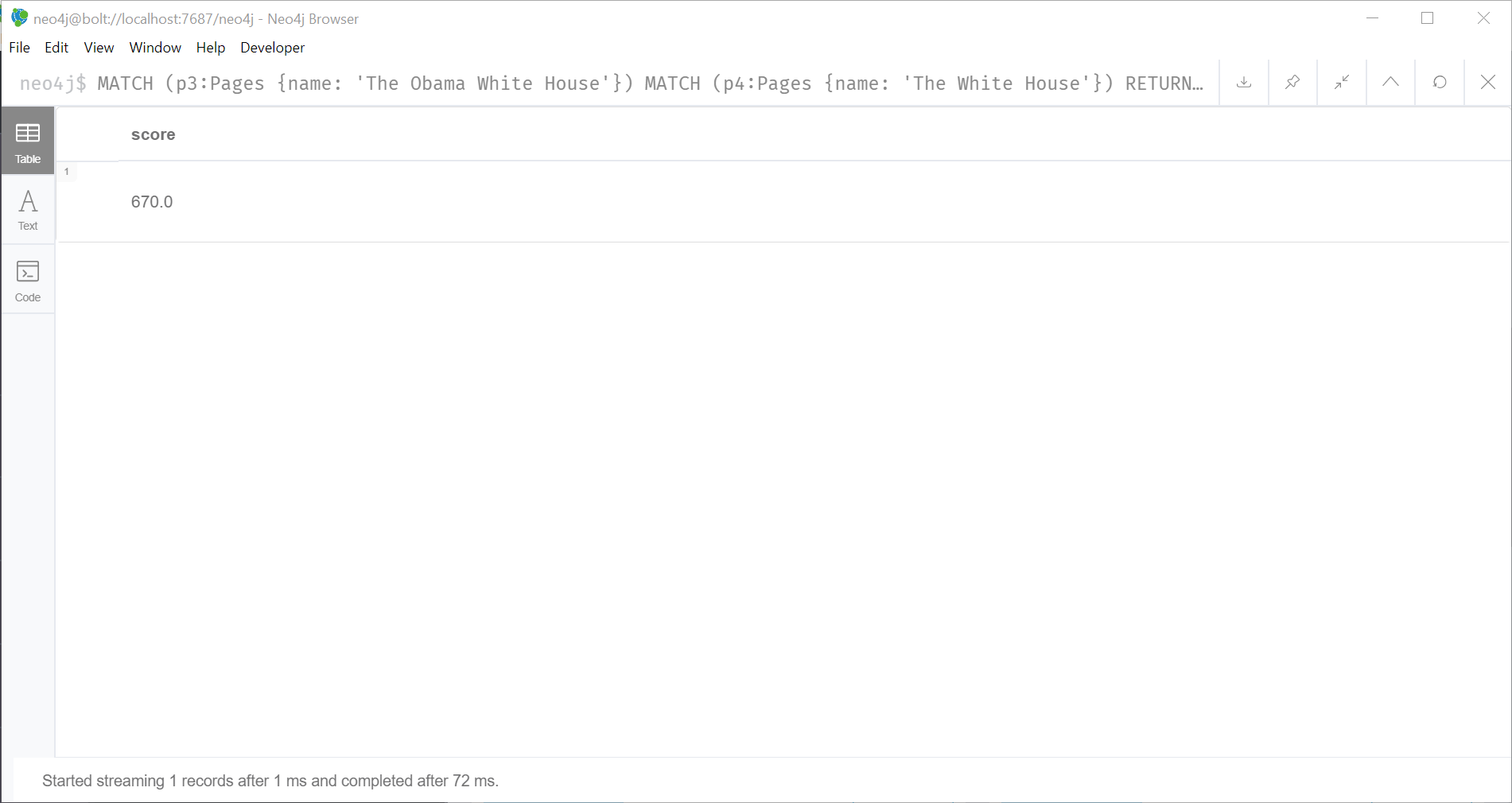
YIELD nodeId, score

RETURN gds.util.asNode(nodeId).name AS page, score

ORDER BY score DESC

**Then, choose a graph algorithm that you have NOT used in this course and run the dataset through it in Neo4j.**

I thought it might be kind of fun to look at some of the link prediction algorithms as we have not used these in any labs. The common neighbor algorithm sounded fun, so I will try that one first. I thought I’d look at The Obama White House and The White House to see if they have any common neighbors.



**Analysis:**

They have 670 common likes! That isn’t too surprising though.

**Code:**

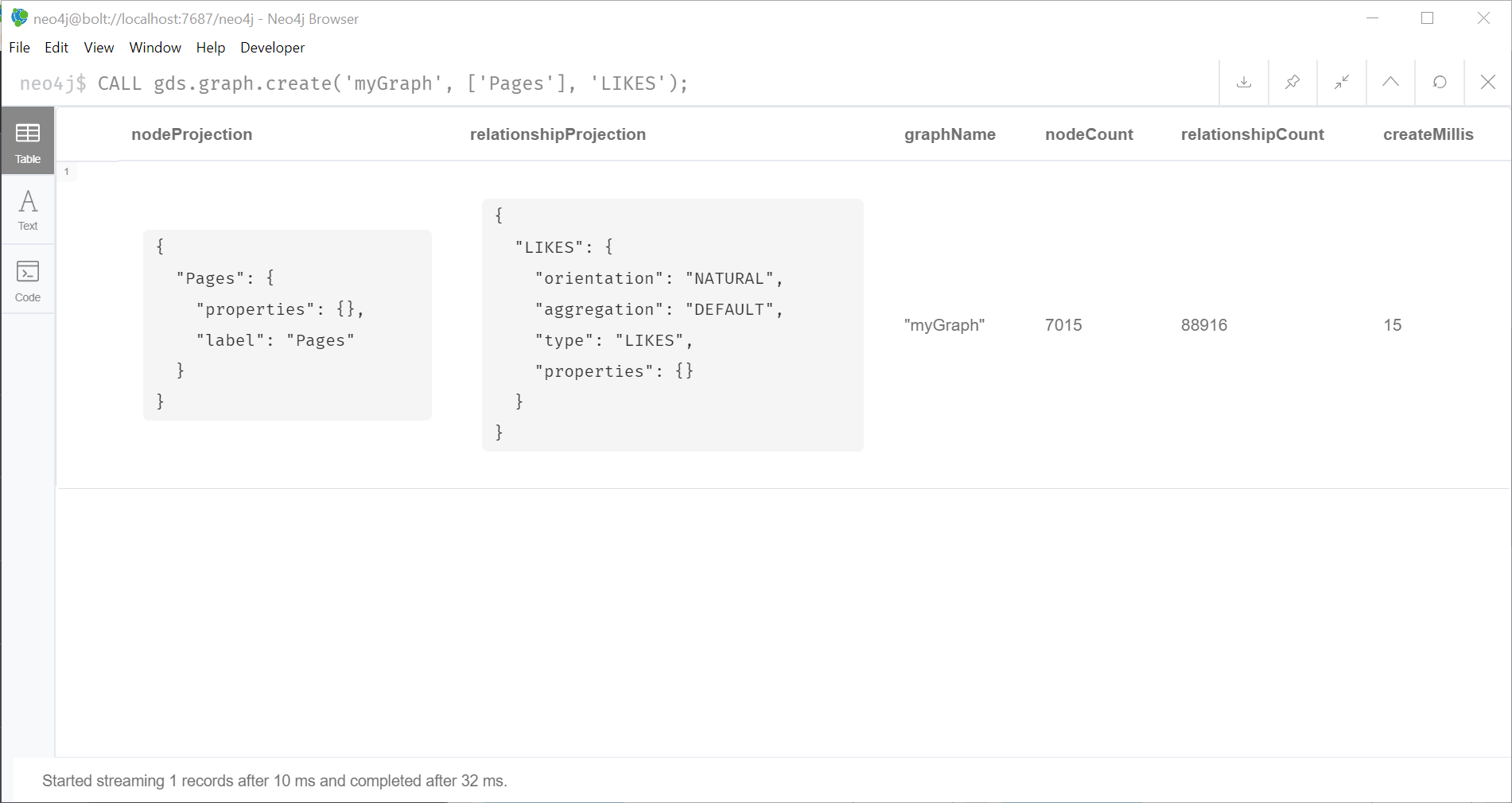
MATCH (p3:Pages {name: 'The Obama White House'})

MATCH (p4:Pages {name: 'The White House'})

RETURN gds.alpha.linkprediction.commonNeighbors(p3, p4) AS score

The last algorithm I wanted to try was the similarity algorithm. I decided to start with the node similarity algorithm. According the Neo4j manual this algorithm “compares a set of nodes based on the nodes they are connected to”.

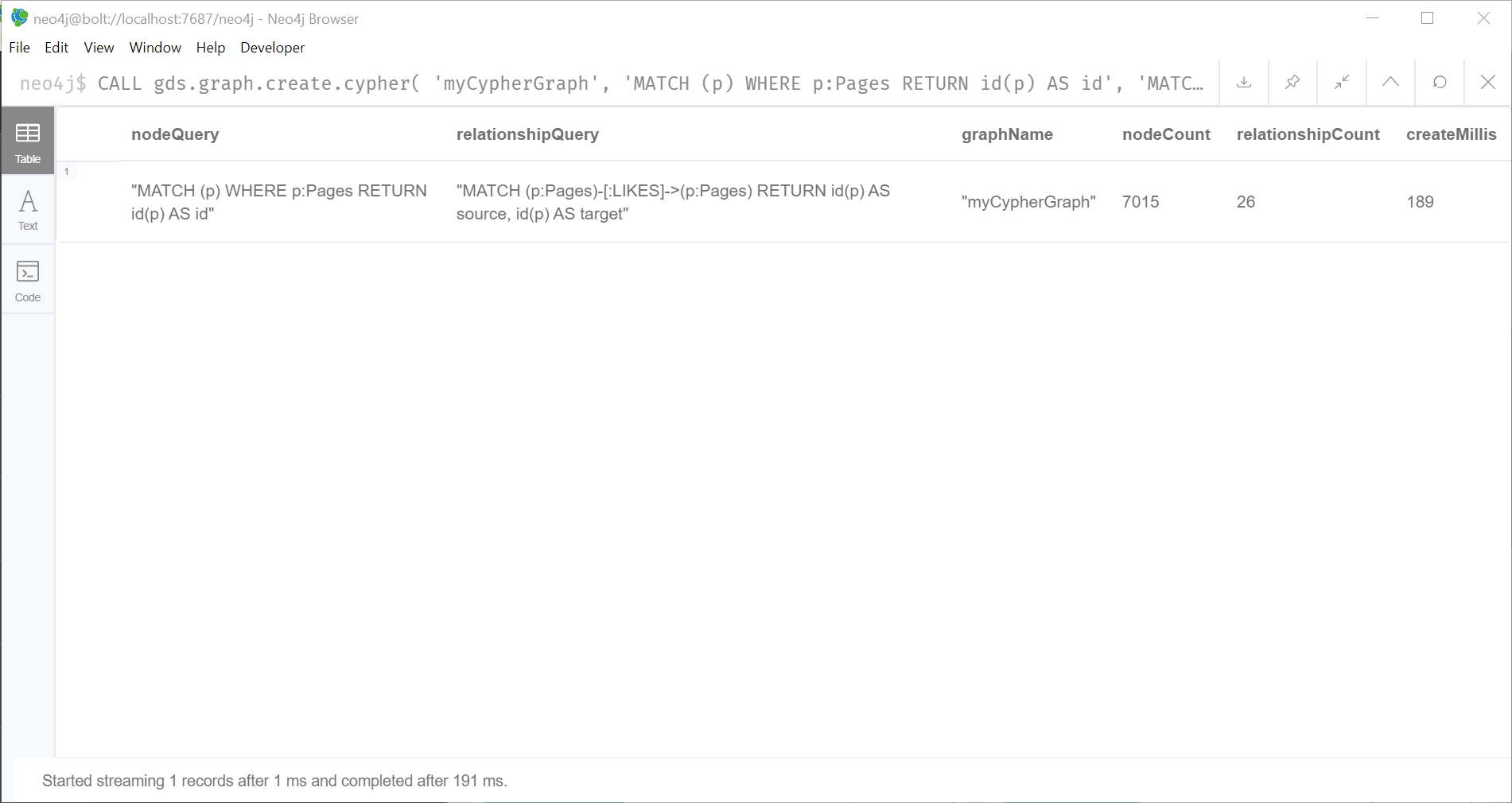
First, I needed to create a graph:



**Code:**

CALL gds.graph.create('myGraph', ['Pages'], 'LIKES');

Then, I created a graph and stored it in the catalog



Code:

CALL gds.graph.create.cypher(

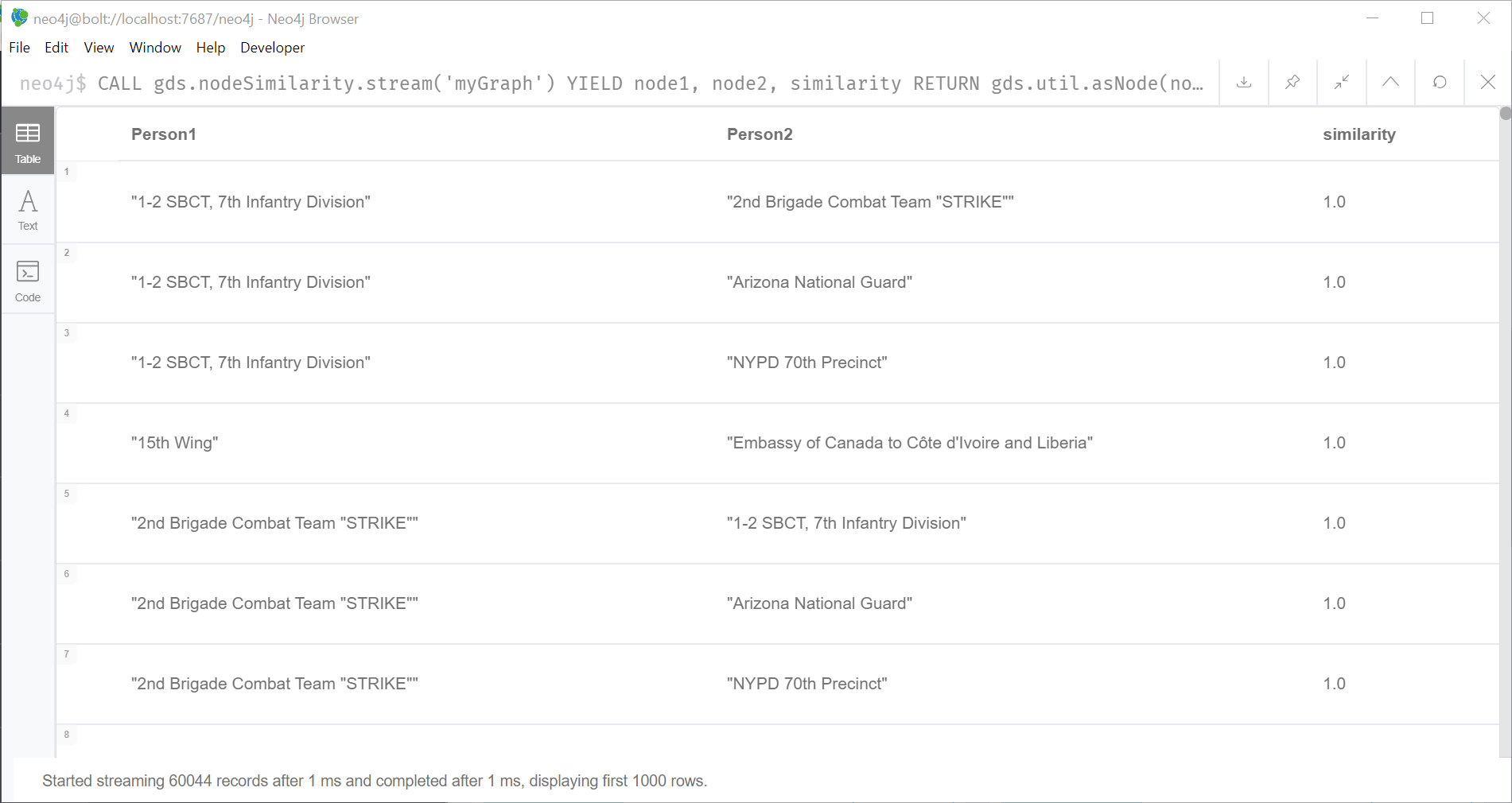
'myCypherGraph',

'MATCH (p) WHERE p:Pages RETURN id(p) AS id',

'MATCH (p:Pages)-[:LIKES]->(p:Pages) RETURN id(p) AS source, id(p) AS target'

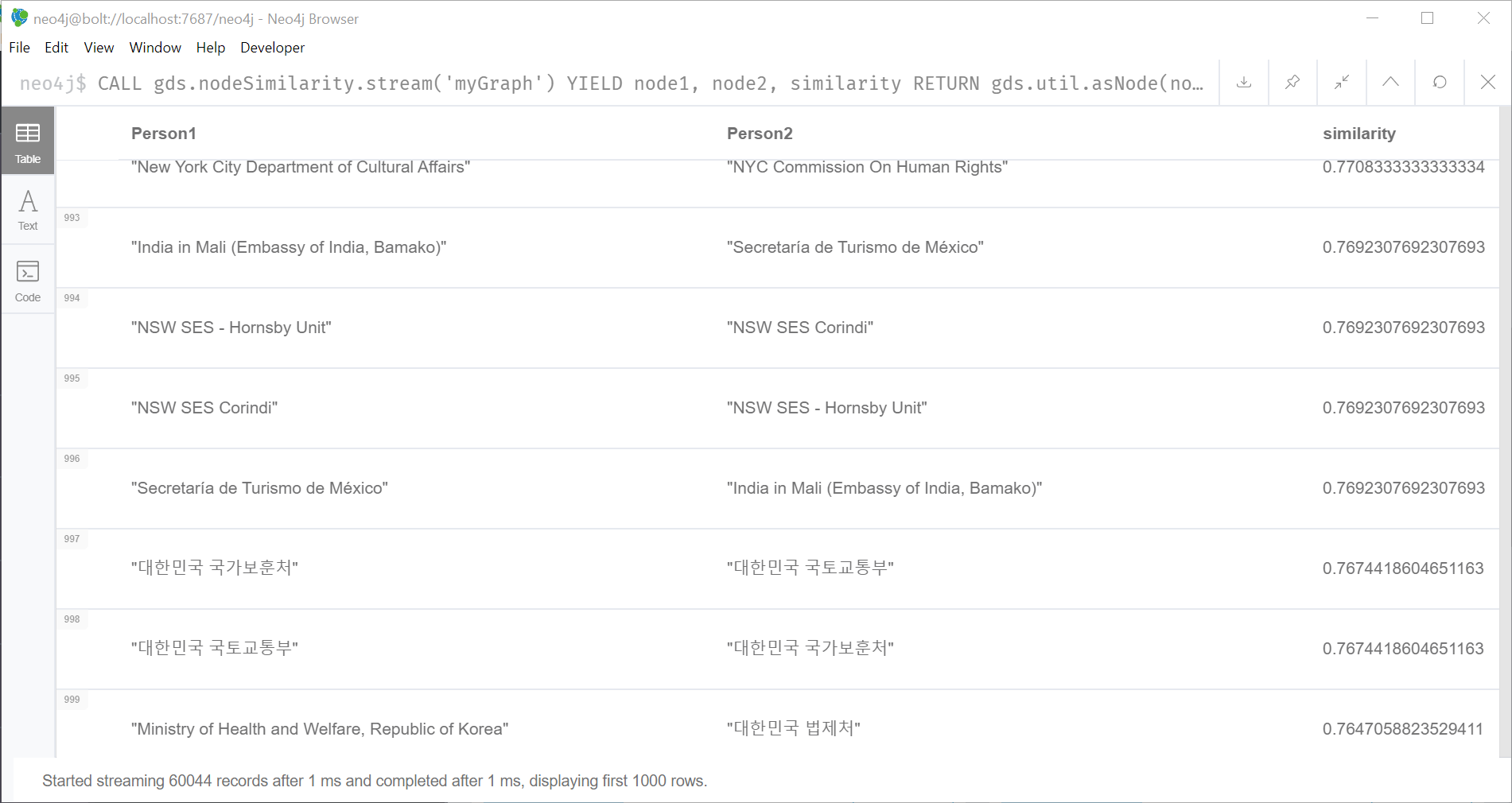
)

Lastly, I will run the algorithm and stream the results:



**Analysis:**

Based on the similarity score’s of 1.0 we can see quite a few nodes share the same neighbors. There are some nodes that do not have as high of a score.



**Code:**

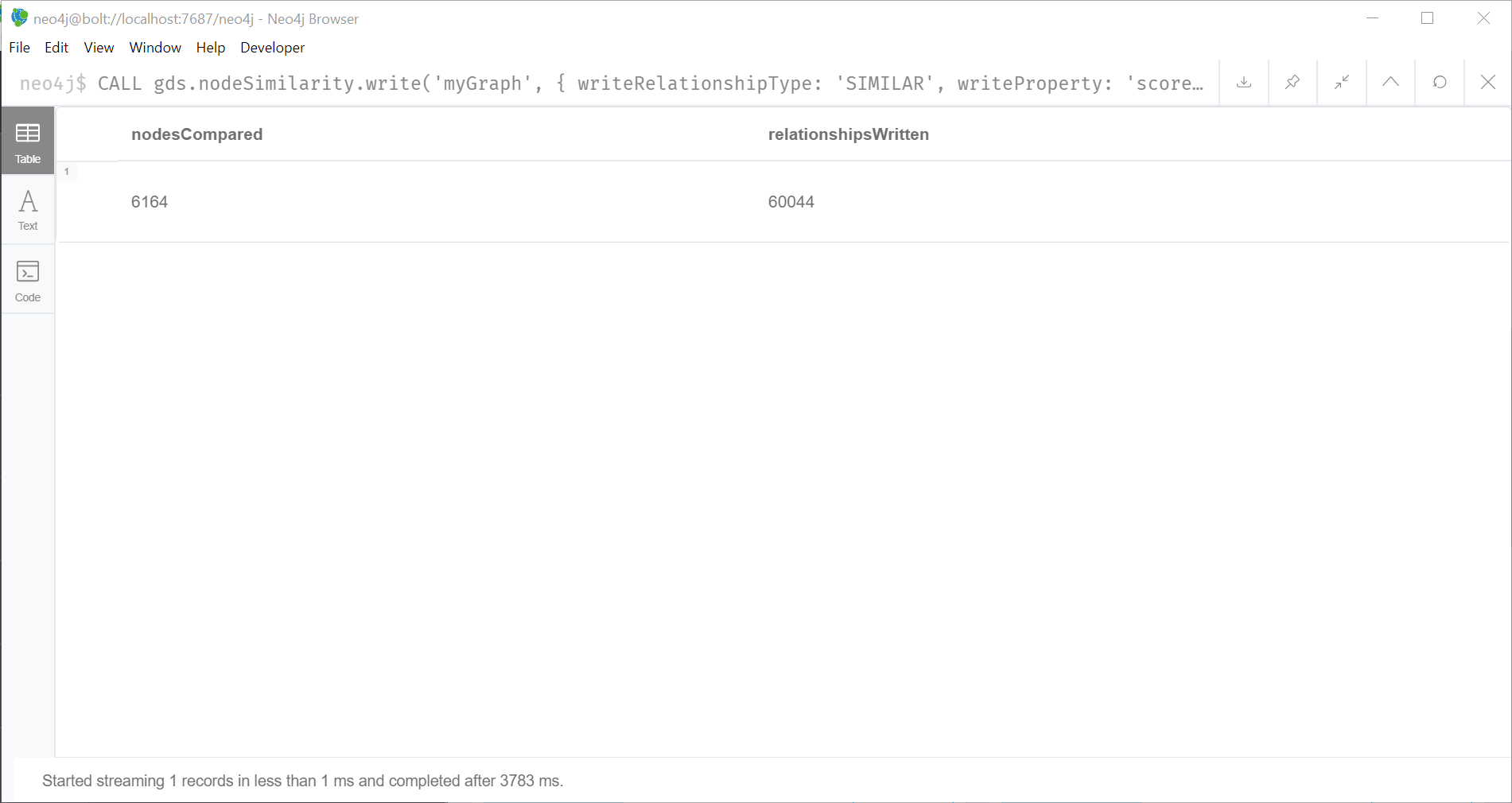
CALL gds.nodeSimilarity.stream('myGraph')

YIELD node1, node2, similarity

RETURN gds.util.asNode(node1).name AS Person1, gds.util.asNode(node2).name AS Person2, similarity

ORDER BY similarity DESCENDING, Person1, Person2

Writing the results



**Analysis:**

The algorithm compared 6164 nodes and wrote 60044 relationships.

**Code:**

CALL gds.nodeSimilarity.write('myGraph', {

writeRelationshipType: 'SIMILAR',

writeProperty: 'score'

})

YIELD nodesCompared, relationshipsWritten

Going just a little bit deeper, I wanted to ignore nodes with less than 3 likes relationships:



**Analysis:**

When a limit on the likes is set you can see the similarity scores go way down, but we do still have similarity between nodes.

**Code:**

CALL gds.nodeSimilarity.stream('myGraph', { degreeCutoff: 3 })

YIELD node1, node2, similarity

RETURN gds.util.asNode(node1).name AS Person1, gds.util.asNode(node2).name AS Person2, similarity

ORDER BY Person1

**Source and Acknowledgements:**

@inproceedings{nr,  
     title={The Network Data Repository with Interactive Graph Analytics and Visualization},  
     author={Ryan A. Rossi and Nesreen K. Ahmed},  
     booktitle={AAAI},  
     url={<http://networkrepository.com>},  
     year={2015}  
}