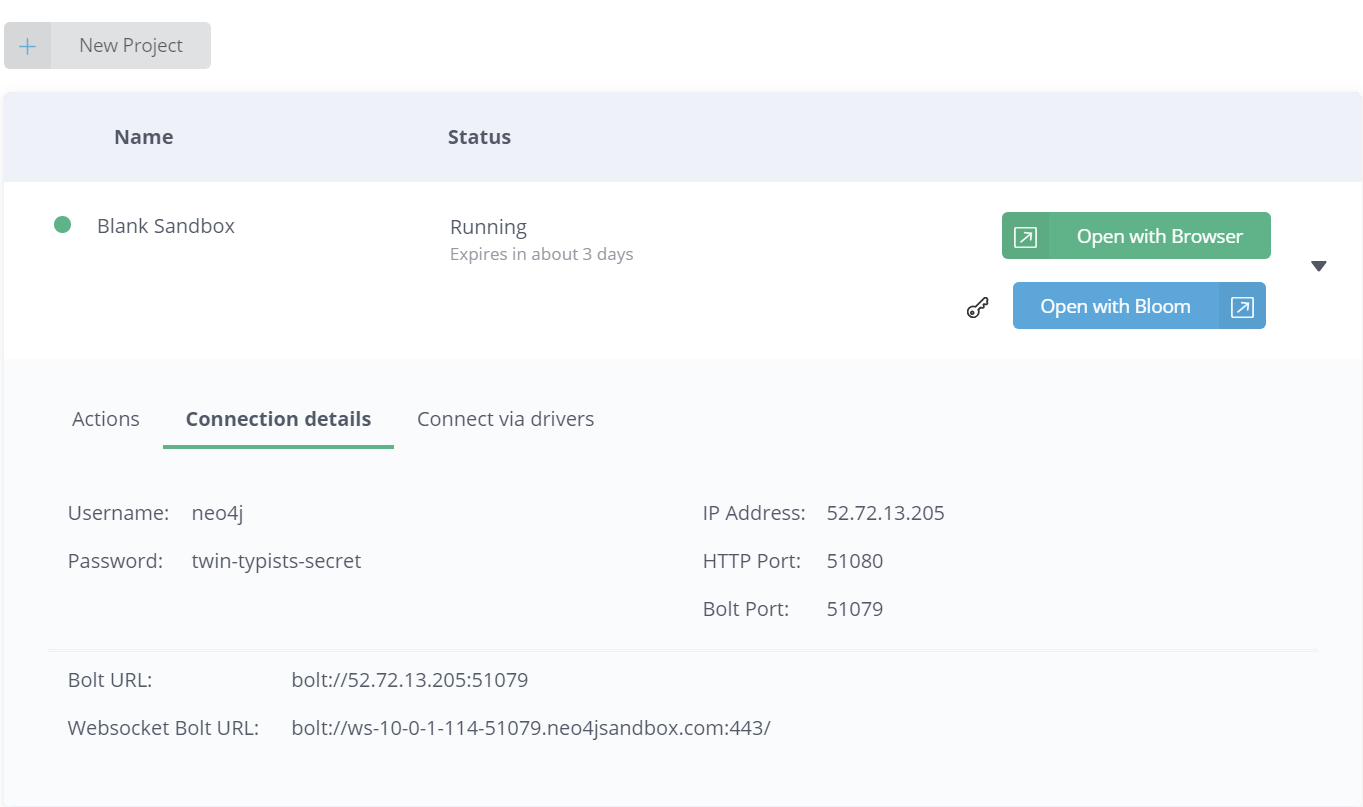
使用neo4j实现louvain代码:



//Examples

//create the example graph in the Neo4j database

CREATE

(nAlice:User {name: 'Alice', seed: 42}),

(nBridget:User {name: 'Bridget', seed: 42}),

(nCharles:User {name: 'Charles', seed: 42}),

(nDoug:User {name: 'Doug'}),

(nMark:User {name: 'Mark'}),

(nMichael:User {name: 'Michael'}),

(nAlice)-[:LINK {weight: 1}]->(nBridget),

(nAlice)-[:LINK {weight: 1}]->(nCharles),

(nCharles)-[:LINK {weight: 1}]->(nBridget),

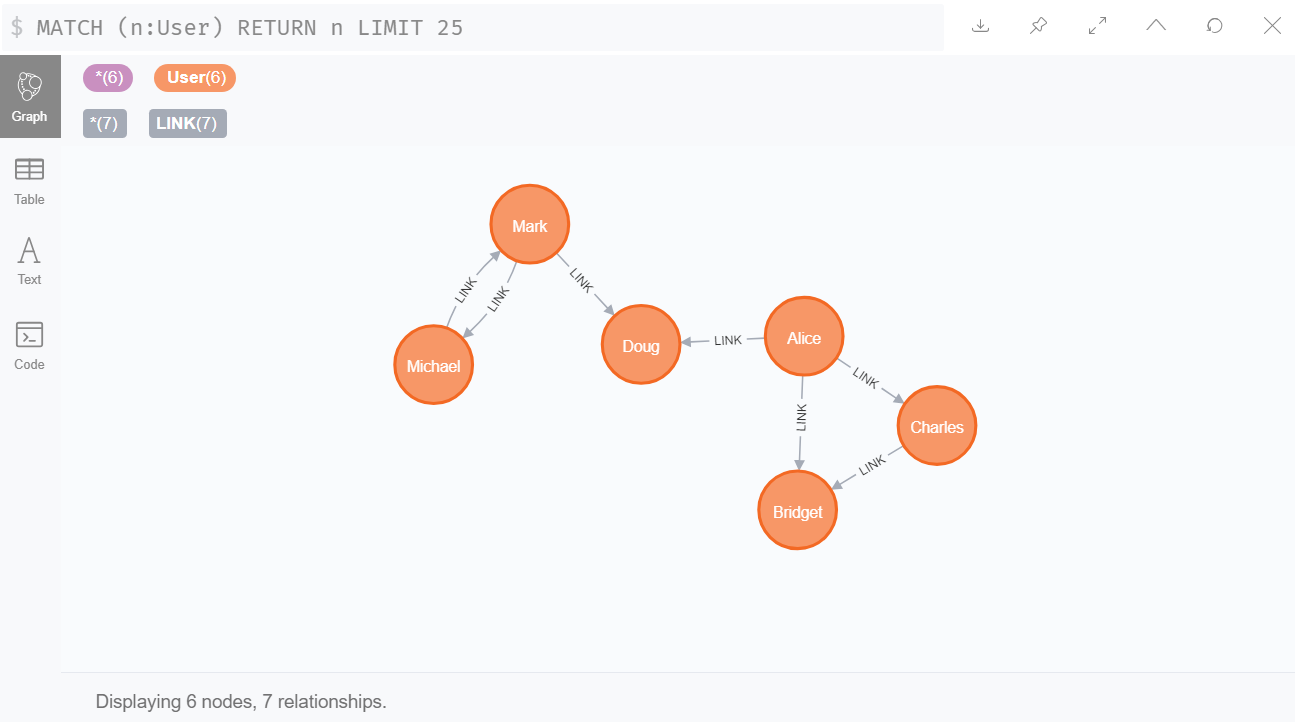
(nAlice)-[:LINK {weight: 5}]->(nDoug),

(nMark)-[:LINK {weight: 1}]->(nDoug),

(nMark)-[:LINK {weight: 1}]->(nMichael),

(nMichael)-[:LINK {weight: 1}]->(nMark);





//create the graph and store it in the graph catalog

CALL gds.graph.create(

'myGraph',

'User',

{

LINK: {

orientation: 'UNDIRECTED'

}

},

{

nodeProperties: 'seed',

relationshipProperties: 'weight'

}

)



//Memory Estimation

//estimate the memory requirements for running the algorithm

CALL gds.louvain.write.estimate('myGraph', { writeProperty: 'community' })

YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory



//Stream

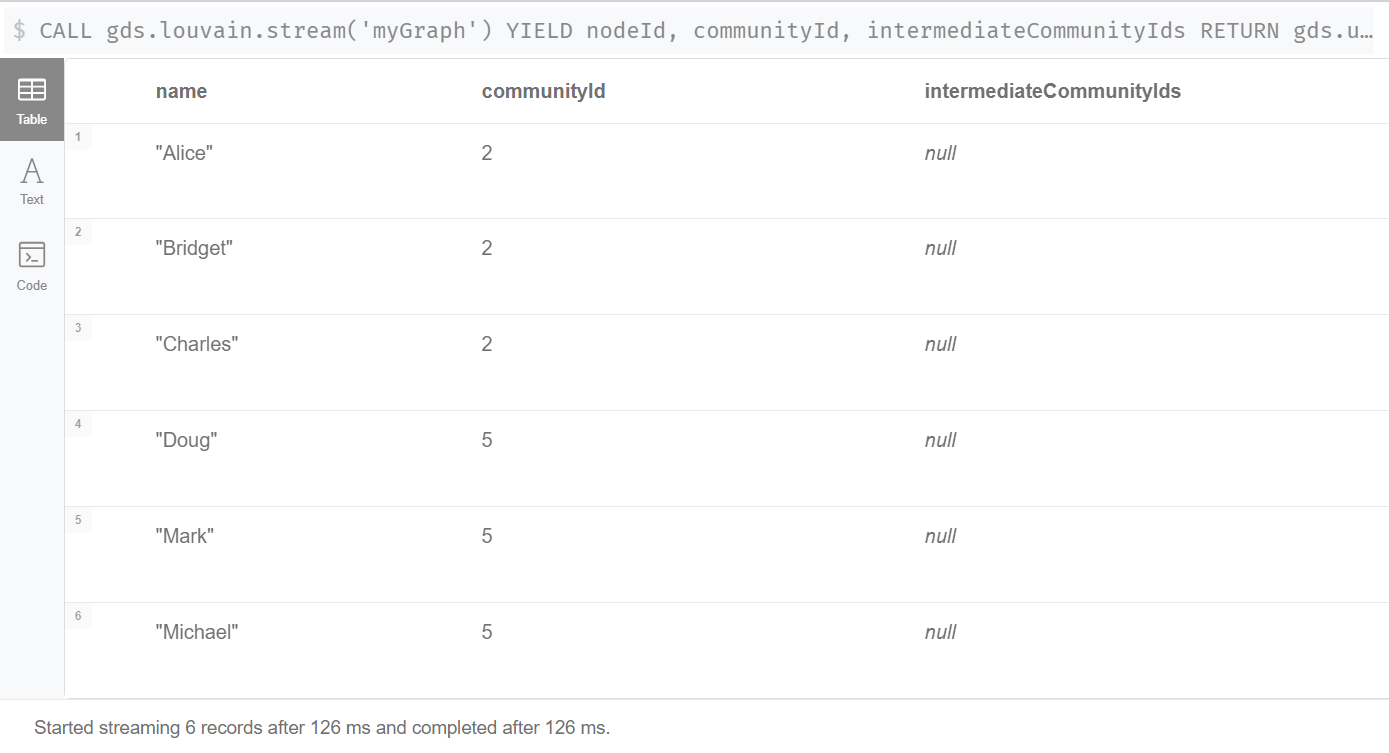
//run the algorithm and stream results

CALL gds.louvain.stream('myGraph')

YIELD nodeId, communityId, intermediateCommunityIds

RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds

ORDER BY name ASC

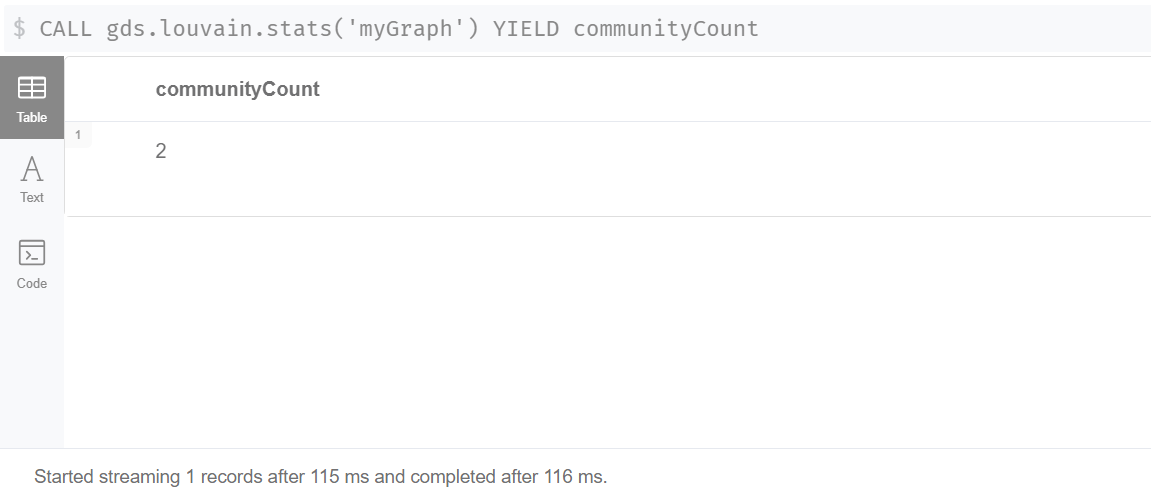


//Stats

//run the algorithm and returns the result in form of statistical and measurement values

CALL gds.louvain.stats('myGraph')

YIELD communityCount



//Mutate

//The following will run the algorithm and store the results in myGraph

CALL gds.louvain.mutate('myGraph', { mutateProperty: 'communityId' })

YIELD communityCount, modularity, modularities



//Write

//The following run the algorithm, and write back results

CALL gds.louvain.write('myGraph', { writeProperty: 'community' })

YIELD communityCount, modularity, modularities



//Weighted

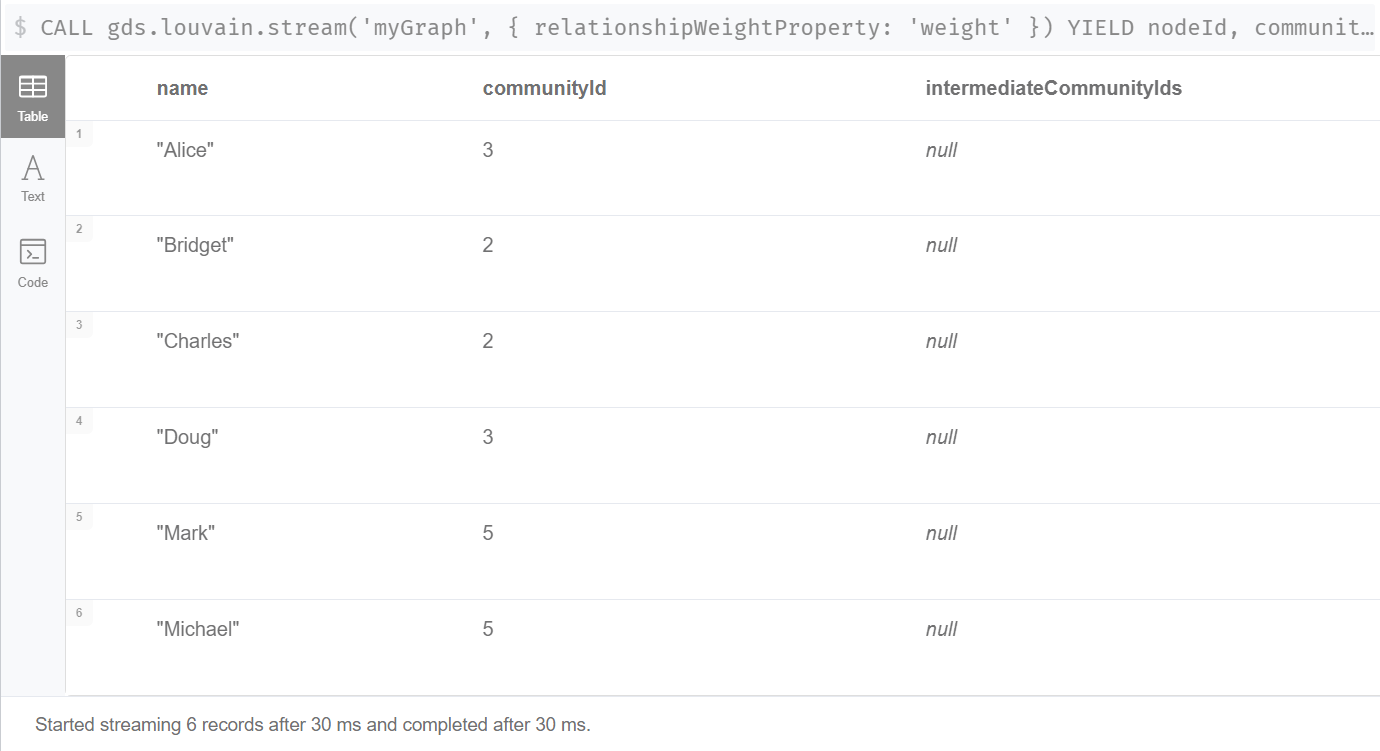
//The following will run the algorithm on a weighted graph and stream results

CALL gds.louvain.stream('myGraph', { relationshipWeightProperty: 'weight' })

YIELD nodeId, communityId, intermediateCommunityIds

RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds

ORDER BY name ASC



//Seeded

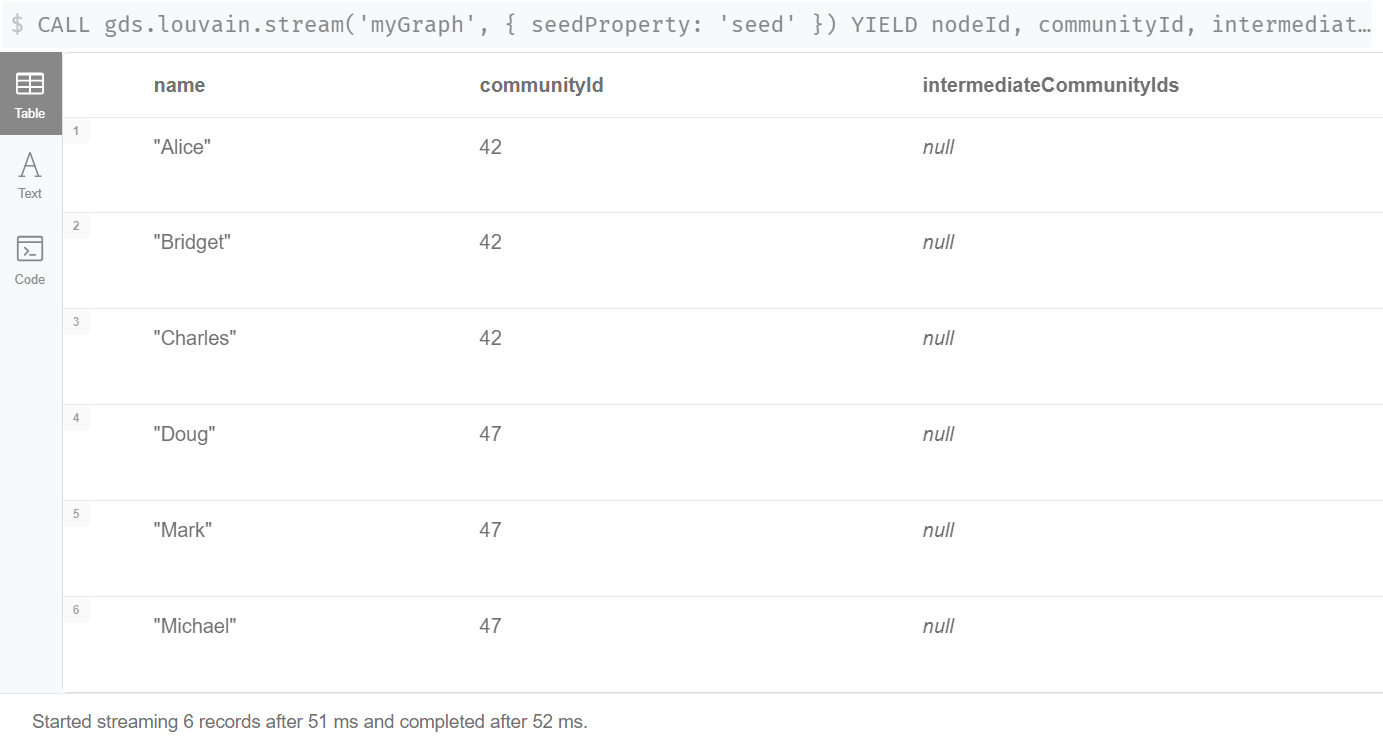
//The following will run the algorithm and stream results

CALL gds.louvain.stream('myGraph', { seedProperty: 'seed' })

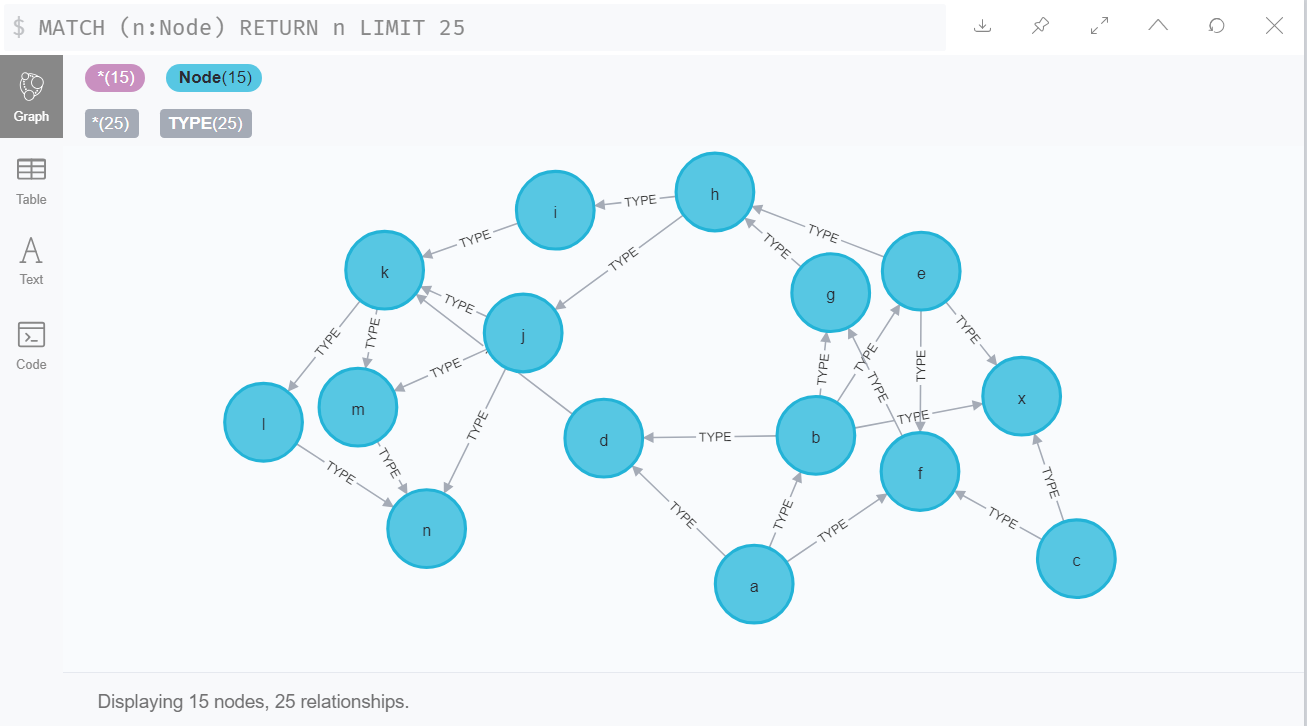
YIELD nodeId, communityId, intermediateCommunityIds

RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds

ORDER BY name ASC



//Stream intermediate communities



CREATE (a:Node {name: 'a'})

CREATE (b:Node {name: 'b'})

CREATE (c:Node {name: 'c'})

CREATE (d:Node {name: 'd'})

CREATE (e:Node {name: 'e'})

CREATE (f:Node {name: 'f'})

CREATE (g:Node {name: 'g'})

CREATE (h:Node {name: 'h'})

CREATE (i:Node {name: 'i'})

CREATE (j:Node {name: 'j'})

CREATE (k:Node {name: 'k'})

CREATE (l:Node {name: 'l'})

CREATE (m:Node {name: 'm'})

CREATE (n:Node {name: 'n'})

CREATE (x:Node {name: 'x'})

CREATE (a)-[:TYPE]->(b)

CREATE (a)-[:TYPE]->(d)

CREATE (a)-[:TYPE]->(f)

CREATE (b)-[:TYPE]->(d)

CREATE (b)-[:TYPE]->(x)

CREATE (b)-[:TYPE]->(g)

CREATE (b)-[:TYPE]->(e)

CREATE (c)-[:TYPE]->(x)

CREATE (c)-[:TYPE]->(f)

CREATE (d)-[:TYPE]->(k)

CREATE (e)-[:TYPE]->(x)

CREATE (e)-[:TYPE]->(f)

CREATE (e)-[:TYPE]->(h)

CREATE (f)-[:TYPE]->(g)

CREATE (g)-[:TYPE]->(h)

CREATE (h)-[:TYPE]->(i)

CREATE (h)-[:TYPE]->(j)

CREATE (i)-[:TYPE]->(k)

CREATE (j)-[:TYPE]->(k)

CREATE (j)-[:TYPE]->(m)

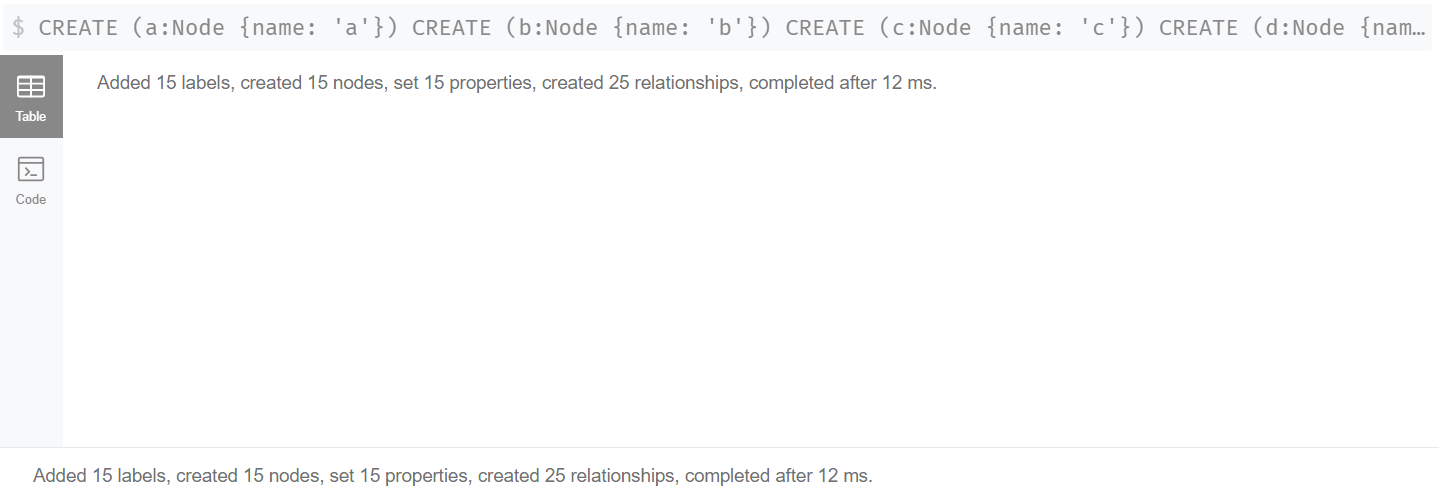
CREATE (j)-[:TYPE]->(n)

CREATE (k)-[:TYPE]->(m)

CREATE (k)-[:TYPE]->(l)

CREATE (l)-[:TYPE]->(n)

CREATE (m)-[:TYPE]->(n);



//The following will load the example graph, run the algorithm and stream results including the intermediate communities

CALL gds.louvain.stream({

nodeProjection: 'Node',

relationshipProjection: {

TYPE: {

type: 'TYPE',

orientation: 'undirected',

aggregation: 'NONE'

}

},

includeIntermediateCommunities: true

}) YIELD nodeId, communityId, intermediateCommunityIds

RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds

ORDER BY name ASC

