Module 1 - Assignment 1

RESPAI VT2022-Group 3

>>Shirdi Manjunath Adigarla

>>Sundara Lakshmi Paramasivam

>>Muthupriya Shankaran

Part 1

1. **cypher codes for creating the database**

   Node creation cypher

phd Student label and properties

create(peter:PhdStudent {name:'Peter Tesla', grantPerMonth:2500}), (nina:PhdStudent {name:'Nina Bashir', grantPerMonth:2500}), (azaria:PhdStudent {name:'Azaria Cohen', grantPerMonth:1500}), (elephteria:PhdStudent {name:'Elephteria Thomas', grantPerMonth:4260}), (abebe:PhdStudent {name:'Abebe Zakaria', grantPerMonth:2600})

Teacher label and properties

create(erik:Teacher {name:'Erik Tesla'}), (anderson:Teacher {name:'Anderson Emo'}), (sofia:Teacher {name:'Sofia Mike'}), (susan:Teacher {name:'Susan Mohammed'}), (janis:Teacher {name:'Janis Solomon'})

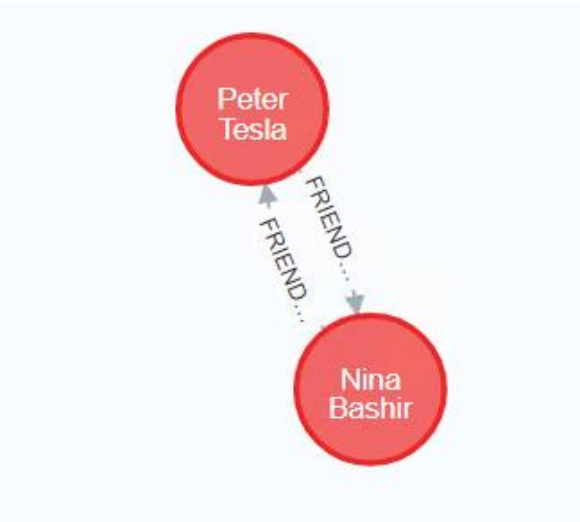
Department label and properties

create(dsv:Department {departmentName:'DSV', location:'Kista'}), (law:Department {departmentName:'Law', location:'Frescaty'}), (physics:Department {departmentName:'Physics', location:'Albano'}), (computational_sci:Department {departmentName:'Computational Science', location:'Albano'}), (mathematics:Department {departmentName:'Mathematics', location:'Albano'}), (psychology:Department {departmentName:'Psychology', location:'Frescaty'})

Relationship Creation Cypher code

PhdStudent FRIEND_OF Relationship

match(a:PhdStudent{name:'Peter Tesla'}),(b:PhdStudent{name:'Nina Bashir'}) create(a)-[:FRIEND_OF]->(b)-[:FRIEND_OF]->(a) return a,b



match(a:PhdStudent{name:'Peter Tesla'}),(b:PhdStudent{name:'Azaria Cohen'}) create(a)-[:FRIEND_OF]->(b)

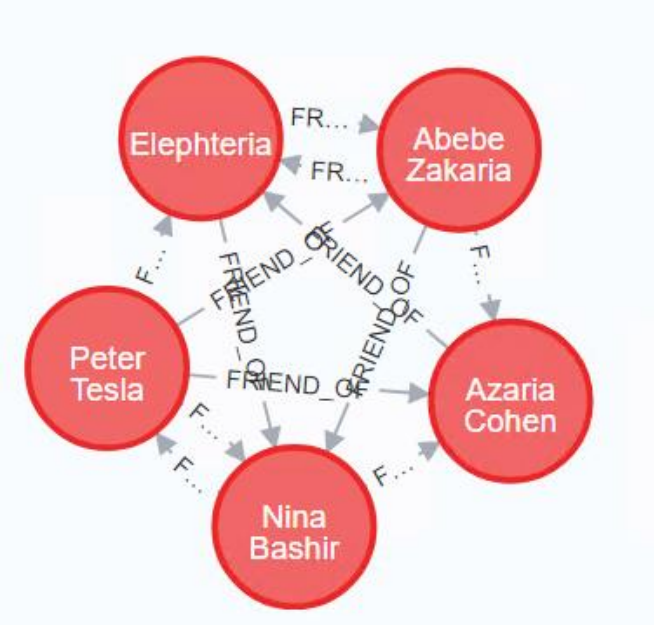
match(a:PhdStudent{name:'Peter Tesla'}),(b:PhdStudent{name:'Elephteria Thomas'}) merge(a)-[:FRIEND_OF]->(b)

match(a:PhdStudent{name:'Peter Tesla'}),(b:PhdStudent{name:'Abebe Zakaria'}) merge(a)-[:FRIEND_OF]->(b)

match(a:PhdStudent{name:'Nina Bashir'}),(b:PhdStudent{name:'Azaria Cohen'}),(c:PhdStudent{name:'Elephteria Thomas'}) merge(a)-[:FRIEND_OF]->(b)-[:FRIEND_OF]->(c)-[:FRIEND_OF]->(a)

match(a:PhdStudent{name:'Nina Bashir'}),(b:PhdStudent{name:'Azaria Cohen'}),(c:PhdStudent{name:'Abebe Zakaria'}) merge(b)<-[:FRIEND_OF]-(c)-[:FRIEND_OF]->(a)

match(a:PhdStudent{name:'Abebe Zakaria'}),(b:PhdStudent{name:'Elephteria Thomas'}) merge(b)-[:FRIEND_OF]->(a)-[:FRIEND_OF]->(b)



Teachers FOLLOWS Relationship

match(a:Teacher{name:'Erik Tesla'}),(b:Teacher{name:'Anderson Emo'}) create(a)-[:FOLLOWS]->(b)

match(a:Teacher{name:'Erik Tesla'}),(b:Teacher{name:'Sofia Mike'}) create(a)-[:FOLLOWS]->(b)

match(a:Teacher{name:'Anderson Emo'}),(b:Teacher{name:'Sofia Mike'}) create(b)<-[:FOLLOWS]-(b)-[:FOLLOWS]->(a)

match(a:Teacher{name:'Susan Mohammed'}),(b:Teacher{name:'Erik Tesla'}) merge(a)-[:FOLLOWS]->(b)

match(a:Teacher{name:'Susan Mohammed'}),(b:Teacher{name:'Anderson Emo'}) merge(a)-[:FOLLOWS]->(b)

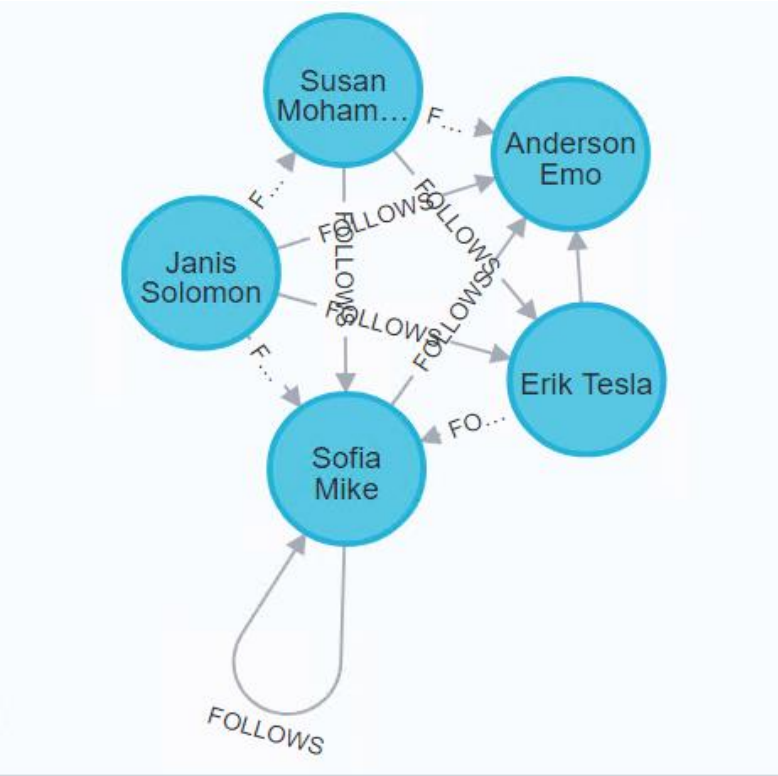
match(a:Teacher{name:'Susan Mohammed'}),(b:Teacher{name:'Sofia Mike'}) merge(a)-[:FOLLOWS]->(b)

match(a:Teacher{name:'Janis Solomon'}),(b:Teacher{name:'Erik Tesla'}) merge(a)-[:FOLLOWS]->(b)

match(a:Teacher{name:'Janis Solomon'}),(b:Teacher{name:'Anderson Emo'}) merge(a)-[:FOLLOWS]->(b)

match(a:Teacher{name:'Janis Solomon'}),(b:Teacher{name:'Sofia Mike'}) merge(a)-[:FOLLOWS]->(b)

match(a:Teacher{name:'Janis Solomon'}),(b:Teacher{name:'Susan Mohammed'}) merge(a)-[:FOLLOWS]->(b)



Phd Student MEMBER_OF Department Relationship

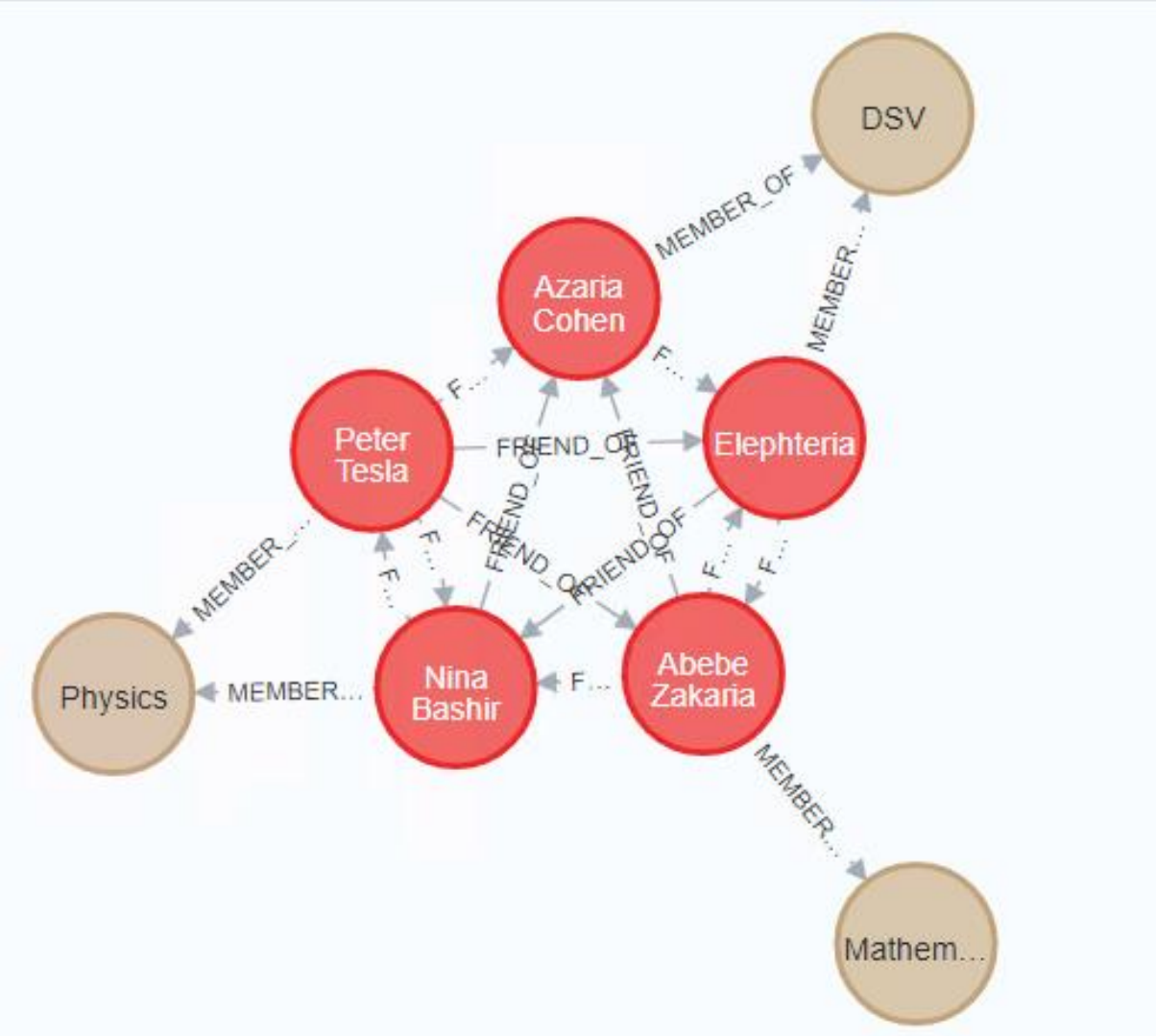
match(a:PhdStudent{name:'Peter Tesla'}),(b:Department{departmentName:'Physics'}) merge (a)-[:MEMBER_OF]->(b)

match(a:PhdStudent{name:'Nina Bashir'}),(b:Department{departmentName:'Physics'}) merg e(a)-[:MEMBER_OF]->(b)

match(a:PhdStudent{name:'Azaria Cohen'}),(b:Department{departmentName:'DSV'}) merge(a)-[:MEMBER_OF]->(b)

match(a:PhdStudent{name:'Elephteria Thomas'}),(b:Department{departmentName:'DSV'}) m erge(a)-[:MEMBER_OF]->(b)

match(a:PhdStudent{name:'Abebe Zakaria'}),(b:Department{departmentName:'Mathematics' }) merge(a)-[:MEMBER_OF]->(b)



Teacher MEMBER_OF Department Relationship

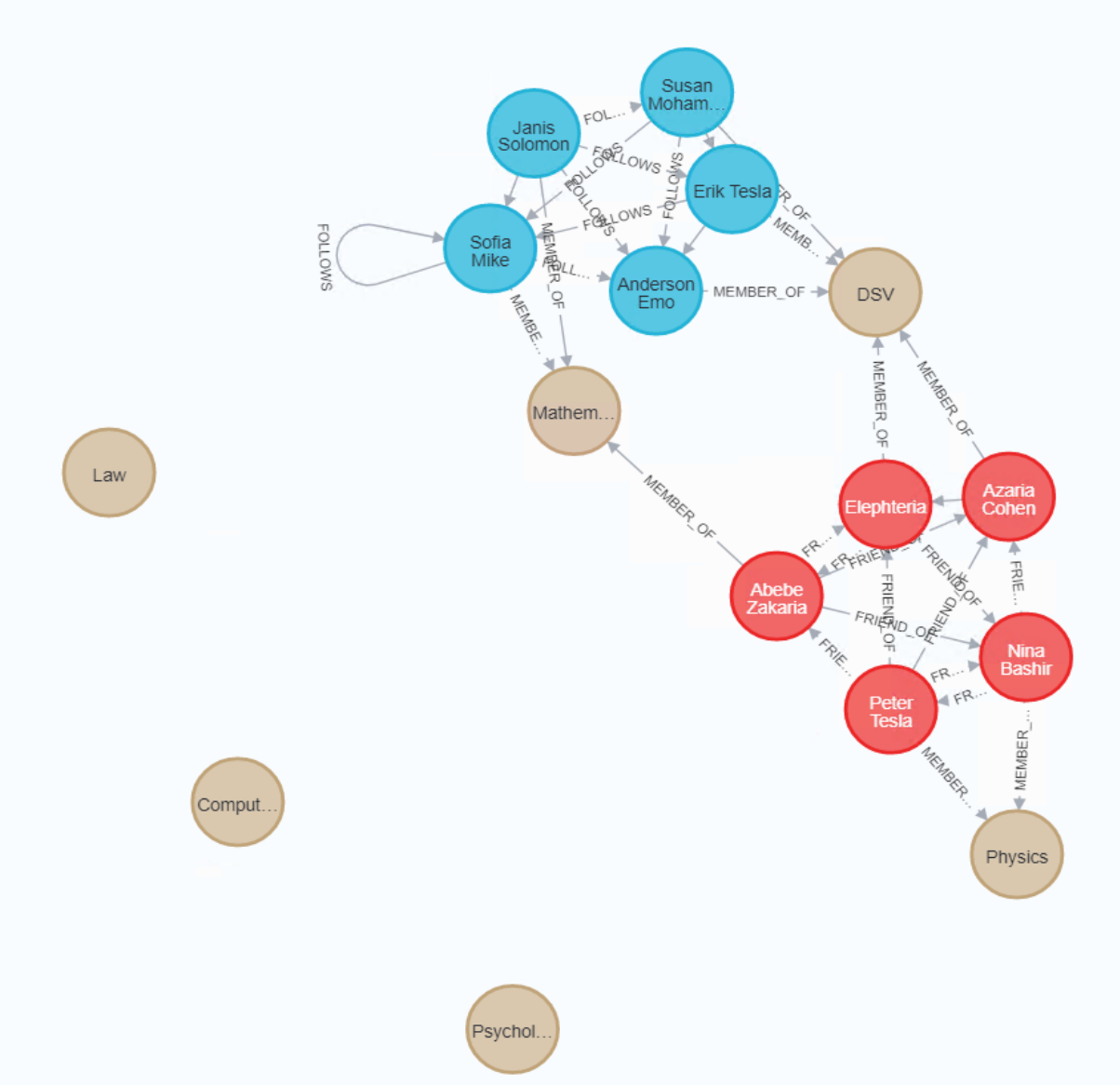
match(a:Teacher{name:'Erik Tesla'}),(b:Department{departmentName:'DSV'}) merge(a)-[:MEMBER_OF]->(b)

match(a:Teacher{name:'Anderson Emo'}),(b:Department{departmentName:'DSV'}) merge(a)-[:MEMBER_OF]->(b)

match(a:Teacher{name:'Sofia Mike'}),(b:Department{departmentName:'Mathematics'}) merge(a)-[:MEMBER_OF]->(b)

match(a:Teacher{name:'Susan Mohammed'}),(b:Department{departmentName:'DSV'}) merge(a)-[:MEMBER_OF]->(b)

match(a:Teacher{name:'Janis Solomon'}),(b:Department{departmentName:'Mathematics'}) merge(a)-[:MEMBER_OF]->(b)



2. cypher codes to display or query Teachers who are MEMBER_OF of ‘DSV.’

match(n:Teacher)-[:MEMBER_OF]->({departmentName:'DSV'}) return n

The teachers **Erik Tesla, Anderson Emo and Susan Mohammed** are the member_of DSV.

Output:

1. {

"identity": 5,
"labels": [
"Teacher"
],
"properties": {
"name": "Erik Tesla"
}
}

2. {

"identity": 6,
"labels": [
"Teacher"
],
"properties": {
"name": "Anderson Emo"
}
}

3. {

"identity": 8,
"labels": [
"Teacher"
],
"properties": {
"name": "Susan Mohammed"
}

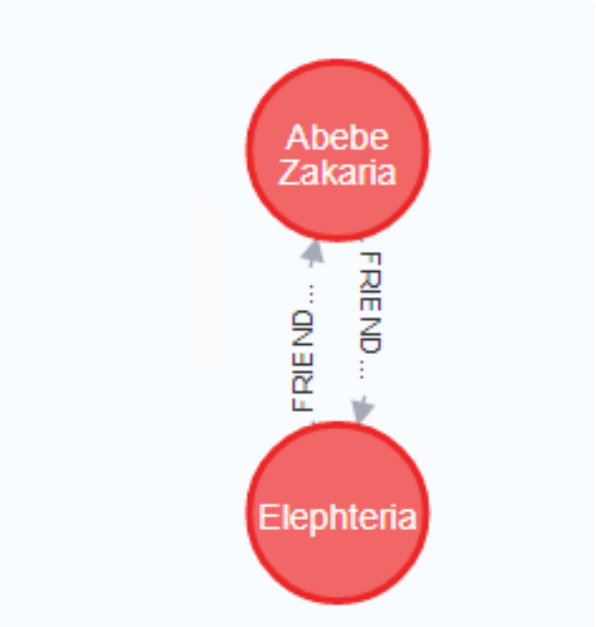
}

3. cypher code to display or query all nodes with a label PhdStudent whose grantPerMonth is greater than 2550.

match(x:PhdStudent) where x.grantPerMonth>2550 return x

Elephteria Thomas and Abebe Zakaria are PhdStudents who earns more than 2550

output:

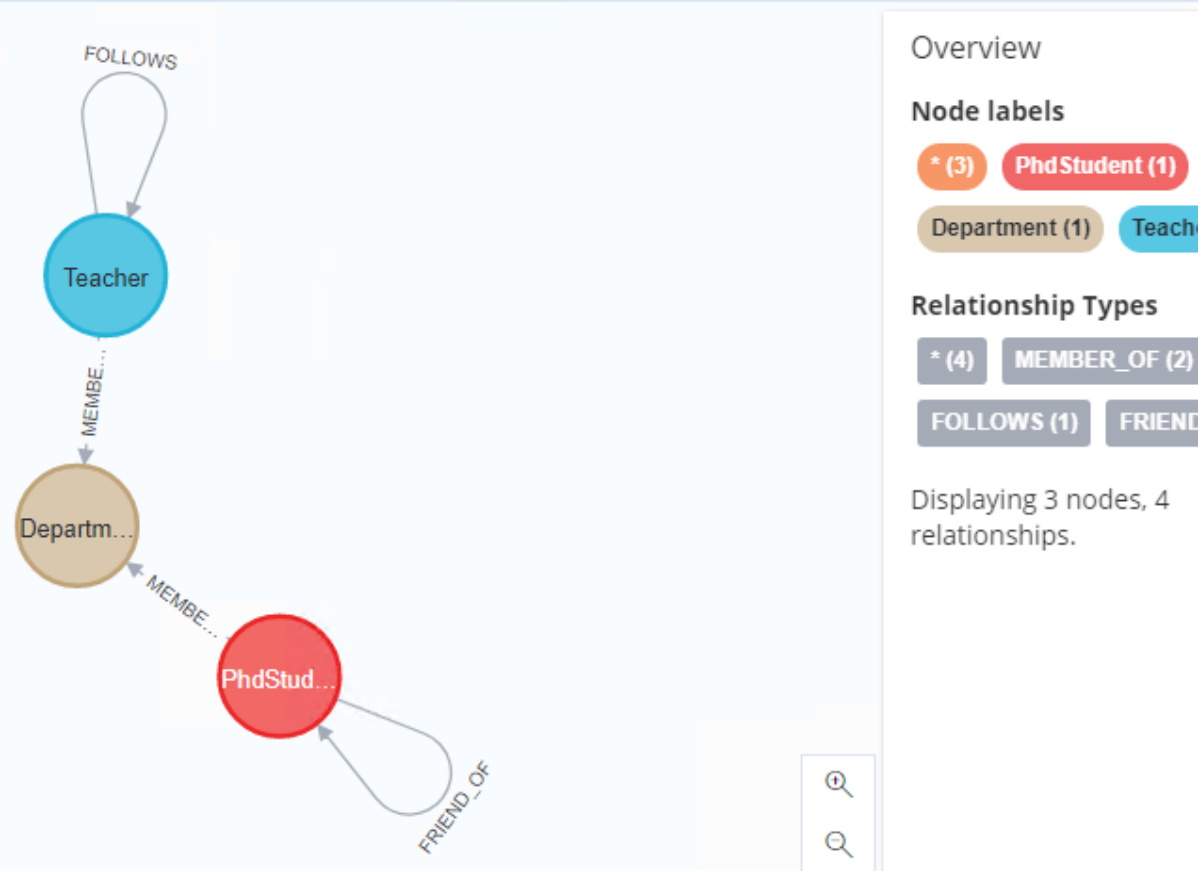


1.{
"identity": 3,
"labels": [
"PhdStudent"
],
"properties": {
"grantPerMonth": 4260,
"name": "Elephteria Thomas"
}
}
2.{
"identity": 4,
"labels": [
"PhdStudent"

],
"properties": {
"grantPerMonth": 2600,
"name": "Abebe Zakaria"
}
}

3. **A snapshot of the visualization of the schema (Use the cypher code: CALL db.schema.visualization).**

call db.schema.visualization



nodes	relationships	

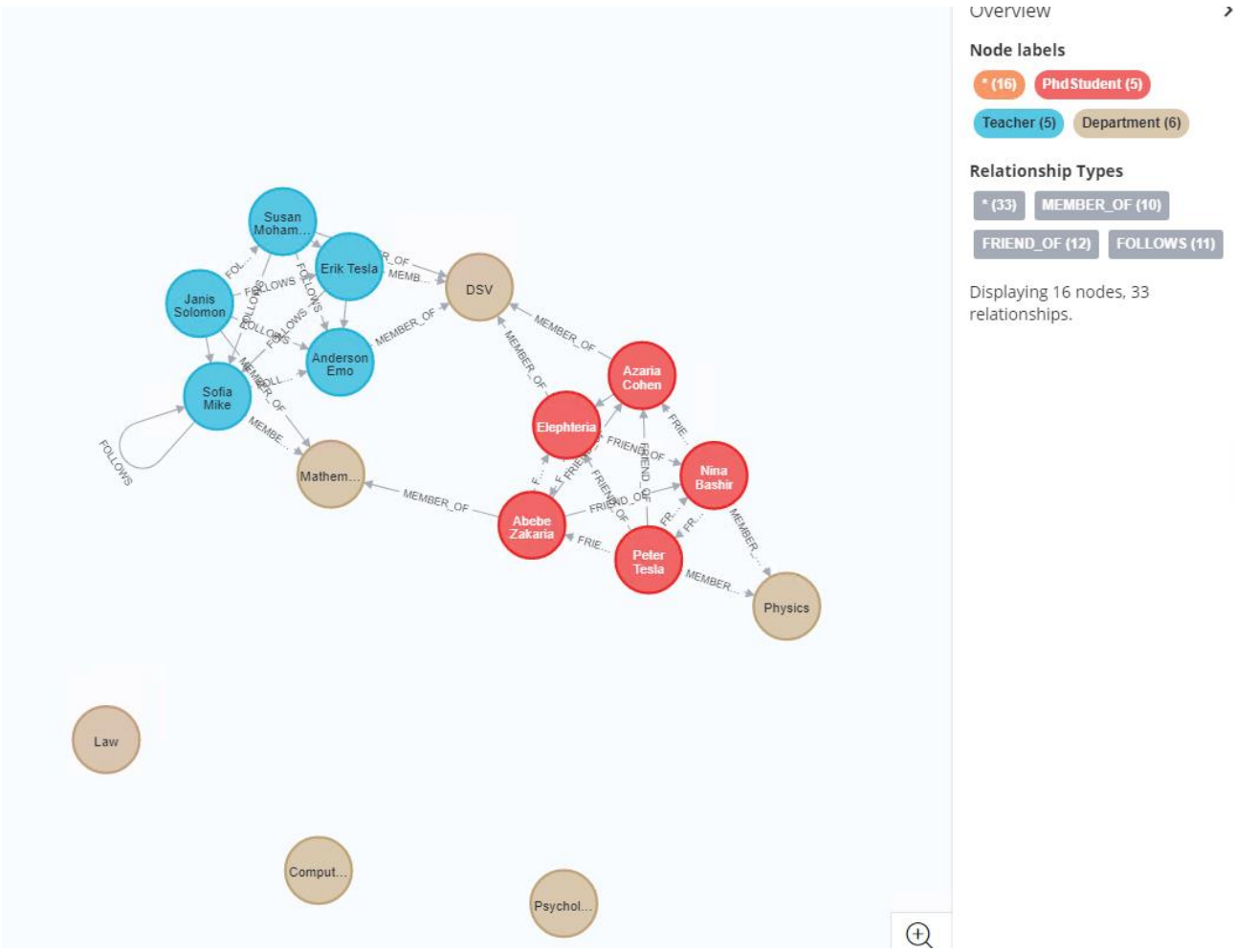
1	[
	{
	"identity": -1,
	"labels": [
	"PhdStudent"
	],
	"properties": {
	"name": "PhdStudent",
	"indexes": [],
	"constraints": []
	}
	}
	,
	{
	"identity": -3,
	"labels": [
	"Department"
	],
	"properties": {
	"name": "Department",
	"indexes": [],
	"constraints": []
	}
	}
	,
	{
	"identity": -2,
	"labels": [
	"Teacher"
	],
	"properties": {
	"name": "Teacher",
	"indexes": [],
	"constraints": []
	}
	}
	]
		[
		{
		"identity": -3,
		"start": -1,
		"end": -3,
		"type": "MEMBER_OF",
		"properties": {
		
		}
		}
		,
		{
		"identity": -4,
		"start": -2,
		"end": -3,
		"type": "MEMBER_OF",
		"properties": {
		
		}
		}
		,
		{
		"identity": -2,
		"start": -2,
		"end": -2,
		"type": "FOLLOWS",
		"properties": {
		
		}
		}
		,
		{
		"identity": -1,
		"start": -1,
		"end": -1,
		"type": "FRIEND_OF",
		"properties": {
		
		}
		}
		]

nodes	relationships	

1	[
	{
	"identity": -1,
	"labels": [
	"PhdStudent"
	],
	"properties": {
	"name": "PhdStudent",
	"indexes": [],
	"constraints": []
	}
	}
	,
	{
	"identity": -3,
	"labels": [
	"Department"
	],
	"properties": {
	"name": "Department",
	"indexes": [],
	"constraints": []
	}
	}
	,
	{
	"identity": -2,
	"labels": [
	"Teacher"
	],
	"properties": {
	"name": "Teacher",
	"indexes": [],
	"constraints": []
	}
	}
	]
		[
		{
		"identity": -3,
		"start": -1,
		"end": -3,
		"type": "MEMBER_OF",
		"properties": {
		
		}
		}
		,
		{
		"identity": -4,
		"start": -2,
		"end": -3,
		"type": "MEMBER_OF",
		"properties": {
		
		}
		}
		,
		{
		"identity": -2,
		"start": -2,
		"end": -2,
		"type": "FOLLOWS",
		"properties": {
		
		}
		}
		,
		{
		"identity": -1,
		"start": -1,
		"end": -1,
		"type": "FRIEND_OF",
		"properties": {
		
		}
		}
		]

4. **A snapshot of the visualization of the whole graph data (Hint: use MATCH clause)**

match(n) return n



Part2:

Task 1: - Pathfinding algorithms

Cypher code to import data

WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/transport-nodes.csv"

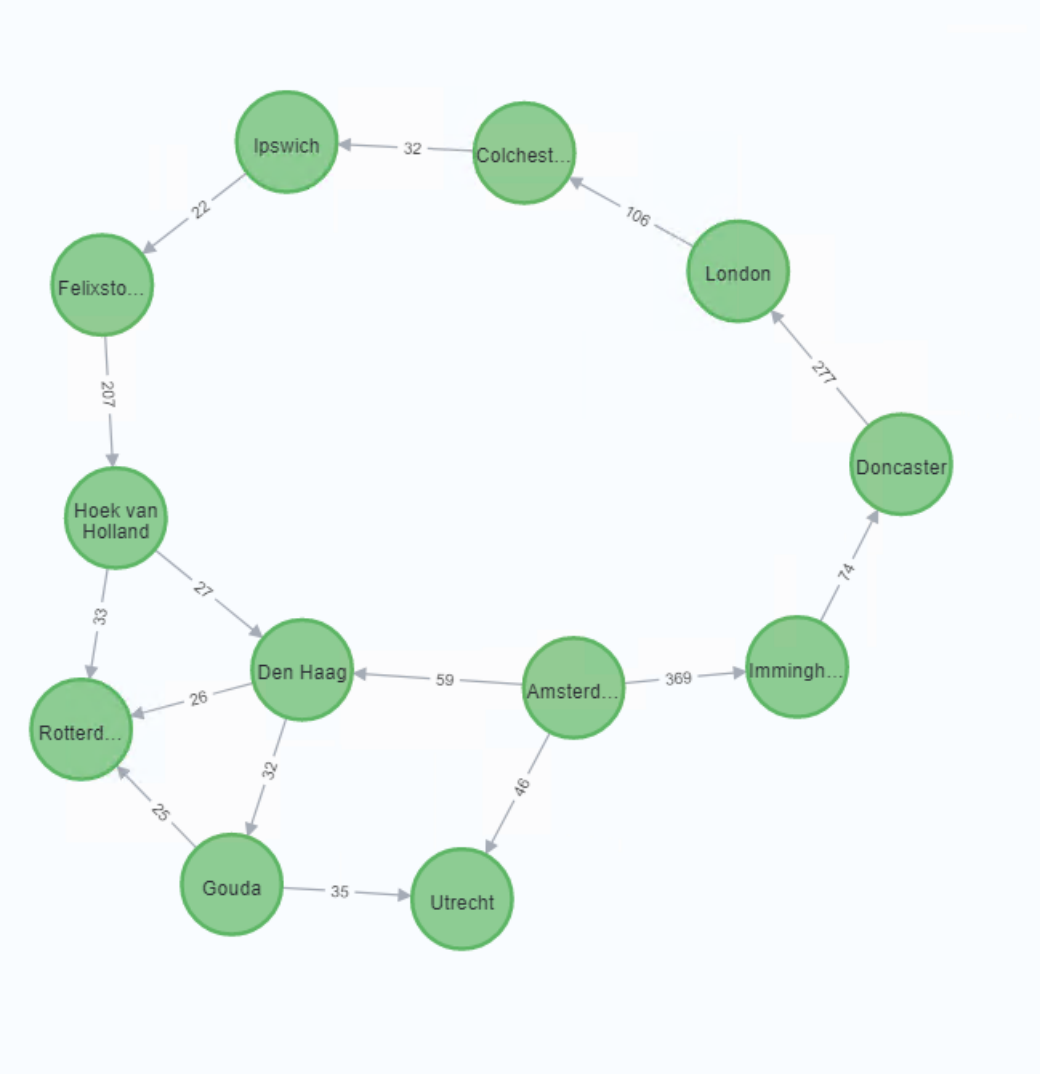
AS uri

LOAD CSV WITH HEADERS FROM uri AS row

MERGE (place:Place {id:row.id})

SET place.latitude = toFloat(row.latitude),

```
    place.longitude = toFloat(row.latitude),
    place.population = toInteger(row.population)
// end::neo4j-import-nodes[]
// tag::neo4j-import-relationships[]
WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/transport-relationships.csv"
AS uri
LOAD CSV WITH HEADERS FROM uri AS row
MATCH (origin:Place {id: row.src})
MATCH (destination:Place {id: row.dst})
MERGE (origin)-[:EROAD {distance: toInteger(row.cost)}]->(destination)
```



Cypher code to create projection

```
CALL gds.graph.create(
 'myProjectedGraph1',
 'Place',
 'EROAD',
 { relationshipProperties: 'distance' })
```

TODO:

1. **Calculate the cheapest route (shortest) based on cost between cities London and Rotterdam using Neo4j cypher**

```
MATCH (source:Place {id: 'London'}), (destination:Place {id: 'Rotterdam'})
CALL gds.shortestPath.dijkstra.stream('myProjectedGraph1', {
 sourceNode: source,
 targetNode: destination,
 relationshipWeightProperty: 'distance'
})
YIELD nodeIds,costs,path,totalCost
RETURN
 nodeIds,[nodeId IN nodeIds | gds.util.asNode(nodeId).id] AS nodeNames,
costs,totalCost
```

	nodeIds	nodeNames	costs	totalCost
1	[15, 14, 13, 12, 11, 16]	["London", "Colchester", "Ipswich", "Felixstowe", "Hoek van Holland", "Rotterdam"]	[0.0, 106.0, 138.0, 160.0, 367.0, 400.0]	400.0

The shortest distance between London and Rotterdam is 400 through Colchester, Ipswich, Felixstowe and Hoek Van Holland.

2. Calculate the cheapest route (shortest) based on cost between Rotterdam and all other cities.

```
MATCH (source:Place {id: 'Rotterdam'})
CALL gds.allShortestPaths.dijkstra.stream('myProjectedGraph1', {
```

```
 sourceNode: source,
 relationshipWeightProperty: 'distance'
})
YIELD index, sourceNode, targetNode, totalCost, nodeIds, costs, path
RETURN index, gds.util.asNode(sourceNode).id AS sourceNodeName,
 gds.util.asNode(targetNode).id AS targetNodeName, totalCost,
 [nodeId IN nodeIds | gds.util.asNode(nodeId).id] AS nodeNames,
 costs, nodes(path) as path ORDER BY index
```

index	sourceNodeName	targetNodeName	totalCost	nodeNames	costs	path	
	0	"Rotterdam"	"Rotterdam"	0.0	["Rotterdam"]	[0.0]	[
							{
							"identity": 16,
							"labels": [
							"Place"
							],
1							"properties": {
							"id": "Rotterdam",
							"latitude": 51.9225,
							"longitude": 51.9225,
							"population": 623652
							}
							}
							]

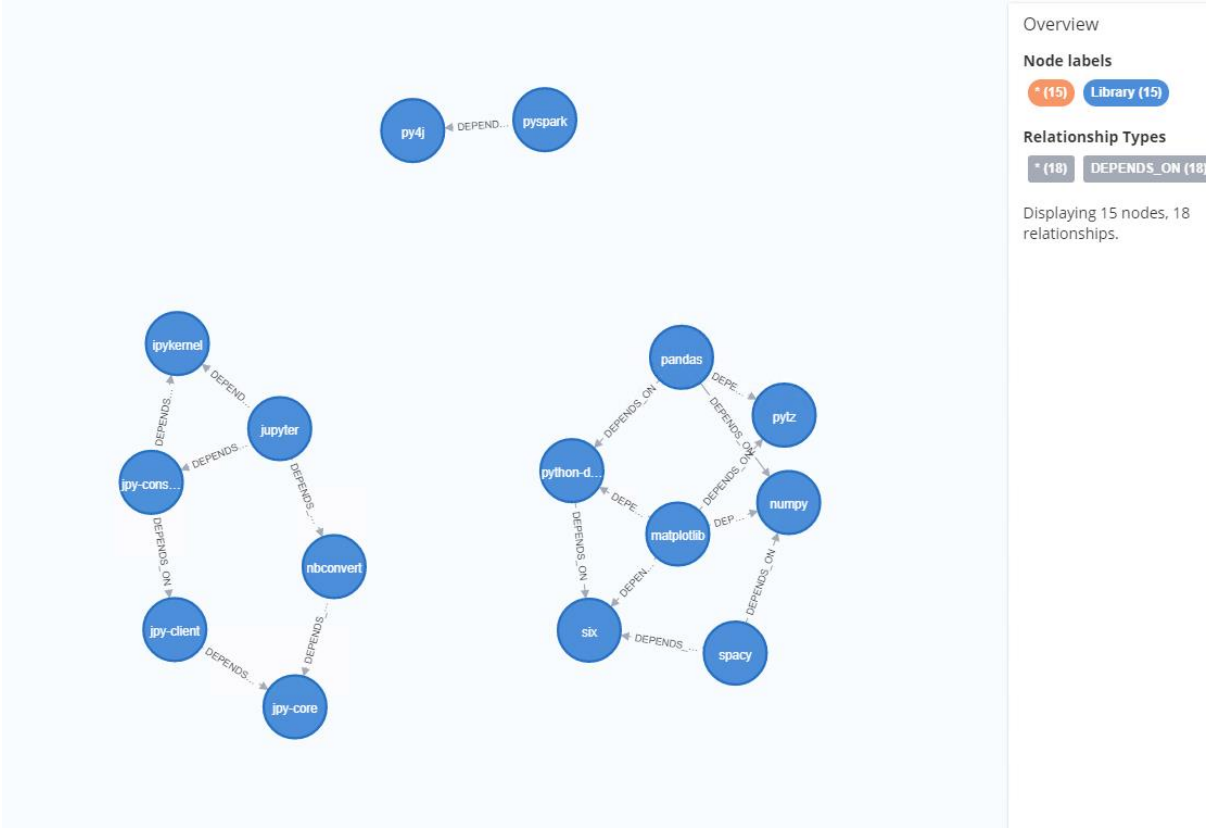
According to the graph rotterdam has no outwards arcs/ distance so we can't actually go anywhere with no arrow direction and we also see that 0 from above dijkstra ssd etc algorithms.

Task 2: - Community detection algorithms

Cypher code for importing data

```
// tag::neo4j-import-nodes[]

WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/sw-nodes.csv"

AS uri

LOAD CSV WITH HEADERS FROM uri AS row
```

```
MERGE (:Library {id: row.id})

// end::neo4j-import-nodes[]

// tag::neo4j-import-relationships[]

WITH "https://github.com/neo4j-graph-analytics/book/raw/master/data/sw-relationships.csv"

AS uri

LOAD CSV WITH HEADERS FROM uri AS row

MATCH (source:Library {id: row.src})

MATCH (destination:Library {id: row.dst})

MERGE (source)-[:DEPENDS_ON]->(destination)

// end::neo4j-import-relationships[]
```



1.Run community detection (triangle count algorithm) on the dataset using Neo4j cypher

Named cypher graph projection code for analysis

```
CALL gds.graph.create(
 'myProjectedGraph5',
 'Library',
 { DEPENDS_ON: {
 orientation: 'UNDIRECTED'
 } } )
```

Triangle prediction algorithm (Local Triangle Count)

```
CALL gds.triangleCount.stream('myProjectedGraph5')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).id AS name, triangleCount
ORDER BY triangleCount DESC
```

	name	triangleCount
1	"six"	1
2	"python-dateutil"	1
3	"matplotlib"	1
4	"jupyter"	1
5	"jpy-console"	1
6	"ipykernel"	1
7	"pandas"	0
8	"numpy"	0
9	"pytz"	0
10	"pyspark"	0
11	"spacy"	0

12	"py4j"	0
13	"nbconvert"	0
14	"jpy-client"	0
15	"jpy-core"	0

The node six, pyrhon_dateutil and matplotlib are detected to form 1 triangle through the graph and have a value of triangle count as 1(it has no connection with other triangles in the graph). The node jupyter, jpy_console and ipykernel forms 1 triangle in the graph and has a value of 1(no nodes are detected in common with 2 triangles). The other nodes have a value of 0 as no triangles are detected with those nodes.

Statistics(Global Triangle Count)

CALL gds.triangleCount.stats('myProjectedGraph5')

YIELD globalTriangleCount, nodeCount

globalTriangleCount	nodeCount
2	15

We see from the graph and output that it has 15 total nodes with 2 global triangles(matplotlib,six,python-dateutil). and (jupyter,jpy_console,ipykernel) and the two global triangles separated are in different graphs. There is no similar node that connects 2 triangles in the same graph. The triangle prediction algorithm detects 2 triangles in total.

2. Run community detection (label propagation) on the dataset using Neo4j cypher

Cypher code to create named projected graph for analysis

```
CALL gds.graph.create(
'myProjectedGraph3',
'Library',
'DEPENDS_ON'
)
```

Label propagation algorithm on stream (community prediction)

CALL gds.labelPropagation.stream('myProjectedGraph3')

YIELD nodeId, communityId AS Community

```
RETURN gds.util.asNode(nodeId).id AS Name, Community
ORDER BY Community, Name
```

Name	Community
"matplotlib"	0
"pandas"	0
"python-dateutil"	0
"six"	0
"spacy"	0
"numpy"	2
"pytz"	4
"py4j"	8
"pyspark"	8
"ipykernel"	12
"jpy-console"	12
"jupyter"	12
"jpy-client"	14

"jpy-core"	14
"nbconvert"	14

Statistics(Global Triangle Count):

CALL gds.labelPropagation.stats('myProjectedGraph3')

YIELD communityCount, ranIterations, didConverge

communityCount	ranIterations	didConverge
6	3	TRUE

(0,2,4,8,12,14) are the total 6 communities and they converge globally. For example py4j,pyspark(8 id community) are farther away from other nodes and so have formed a single community. But a close packed nodes may not form the correct cluster(0,2,4: community ids) locally. But if we treat 0,2,4 as a single community it converges properly.

Task 3: Page Rank

Cypher code to create graph data(User label with name property and LOVES relationship)

MERGE (andrew:User {name: ‘Andrew’})

MERGE (martha:User {name: ‘Martha’})

MERGE (trump:User {name: ‘Trump’})

MERGE (julia:User {name: ‘Julia’})

MERGE (mesi:User {name: ‘Mesi’})

MERGE (zlatan:User {name: ‘Zlatan’})

MERGE (erik:User {name: ‘Erik’})

MERGE (bob:User {name: ‘Bob’})

MERGE (johan:User {name: ‘Johan’})

MERGE(zack:User {name: ‘Zack’})

MERGE (andrew)-[:Loves]->(martha)

MERGE (martha)-[:LOVES]->(mesi)

MERGE (martha)-[:LOVES]->(zlatan)

MERGE (trump)-[:LOVES]->(zlatan)

MERGE (trump)-[:LOVES]->(bob)

MERGE (bob)-[:LOVES]->(trump)

MERGE (julia)-[:LOVES]->(andrew)

MERGE (julia)-[:LOVES]->(johan)

MERGE (julia)-[:LOVES]->(mesi)

MERGE (mesi)-[:LOVES]->(johan)

MERGE (zlatan)-[:LOVES]->(mesi)

MERGE (zlatan)-[:LOVES]->(erik)

MERGE (erik)-[:LOVES]->(mesi)

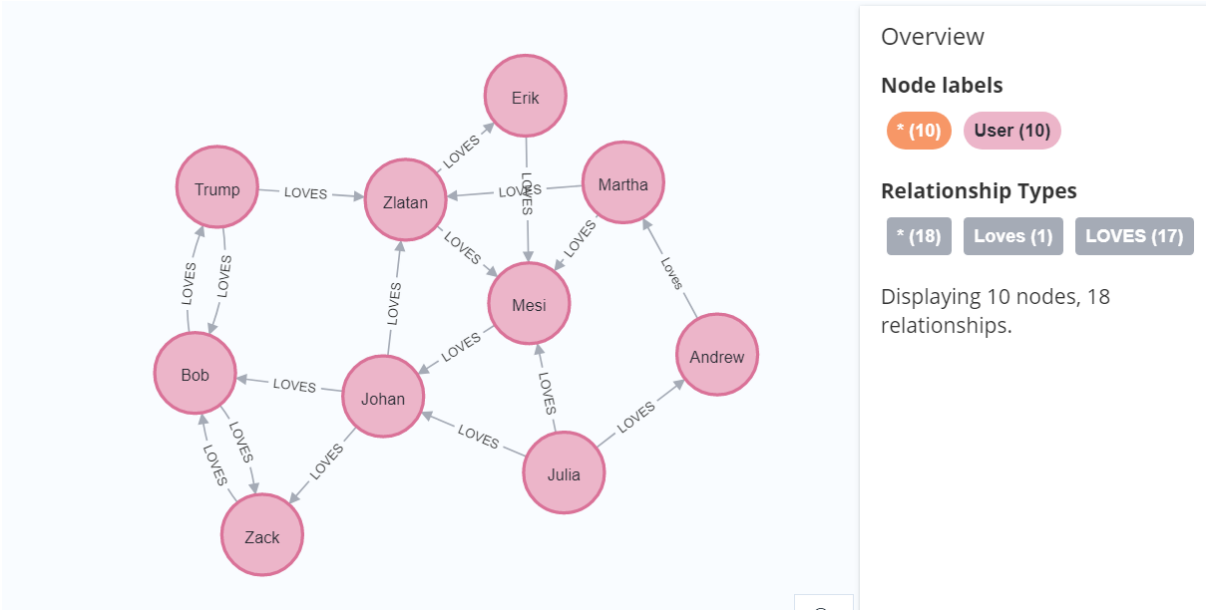
MERGE (bob)-[:LOVES]->(zack)

MERGE (zack)-[:LOVES]->(bob)

MERGE (johan)-[:LOVES]->(zlatan)

MERGE (johan)-[:LOVES]->(bob)

MERGE (johan)-[:LOVES]->(zack)



Cypher code to create named projected graph for analysis

CALL gds.graph.create('myProjectedGraph7', 'User', 'LOVES')

Cypher code for PageRank on Stream mode

```
CALL gds.pageRank.stream('myProjectedGraph7')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

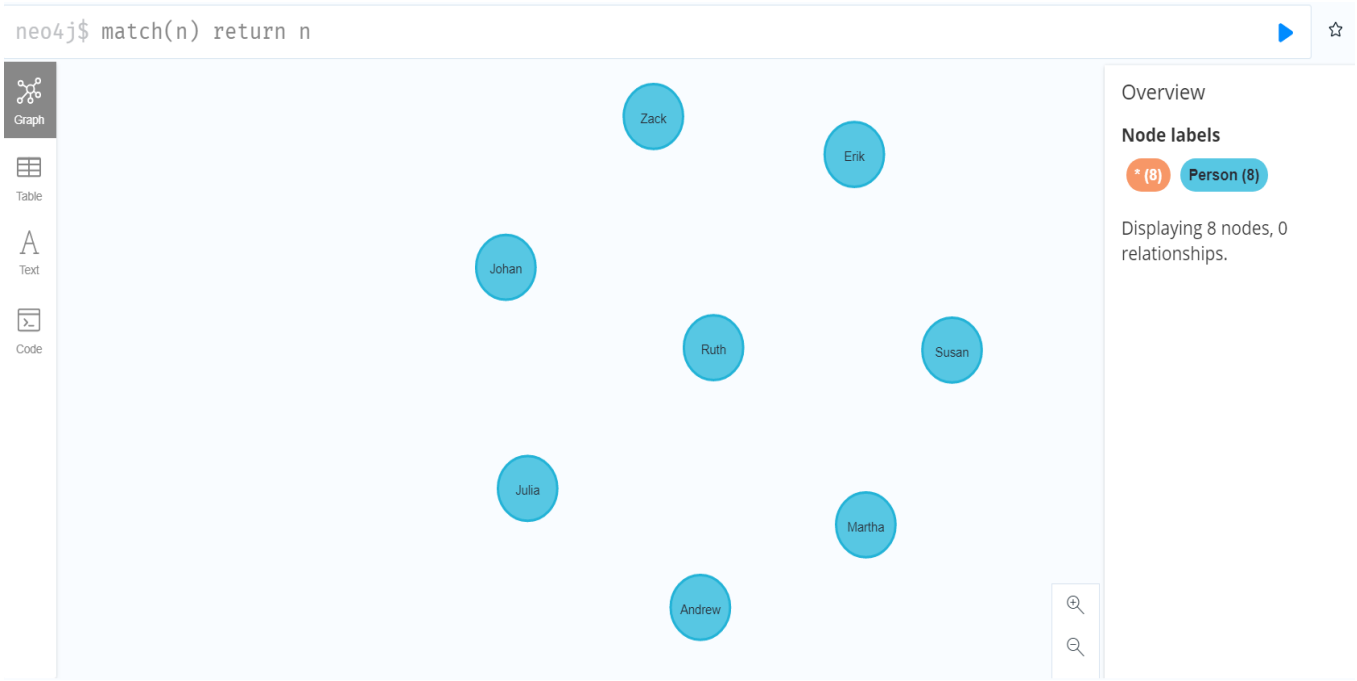
	name	score
1	"Bob"	1.9981380330338196
2	"Zack"	1.3141417939409532
3	"Johan"	1.144062847976722
4	"Mesi"	1.1259572023047637
5	"Trump"	0.9918218081396334
6	"Zlatan"	0.9539631607248121
7	"Erik"	0.5526658844801705
8	"Andrew"	0.19250000000000003
9	"Julia"	0.15000000000000002
10	"Martha"	0.15000000000000002

PageRank measures the transitive(indirect) or directional influence of nodes. *Bob* has the highest(9) transitive influences: an inward arc from Trump, Zack, Johan, Zlatan, Mesi, Erik, Martha, Andrew,Julia and an outward arc to Trump, Zack, Zlatan, Mesi, Johan, Erik. The node Bob has the *most importance and (transitive) influence* in the above pageRank graph based on value(1.998) in output and through graph. *Next highest is Zlack,* as their similarity value based on output table and transitive influence through graph is less comparatively and goes on.

Task 4: Similarity algorithm

Cypher code to create graph data(Person label with name and weight property)

```
MERGE (andrew:Person {name:'Andrew', weight:100})
MERGE (martha:Person {name:'Martha', weight:70})
MERGE (susan:Person {name:'Susan', weight:50})
MERGE (erik:Person {name:'Erik', weight:45})
MERGE (zack:Person {name:'Zack', weight:20})
MERGE (johan:Person {name:'Johan', weight:80})
MERGE (ruth:Person {name:'Ruth', weight:90})
MERGE (julia:Person {name:'Julia', weight:30})
```



Cypher code to create named native projected graph for analysis

```
CALL gds.graph.create(
 'myProjectedGraph9',
 { Person: {
 label: 'Person',
 properties: ['weight']
 } },
 '*');
```

The memory estimation run for similarity weight for knn algorithm is required between [2960 Bytes ... 5360 Bytes] . It has sufficient memory to run knn algorithm stream for weight property.

Cypher code for KNN algorithm on stream mode for weight property

```
CALL gds.beta.knn.stream('myProjectedGraph9', {
 topK: 1,
 nodeWeightProperty: 'weight',
 // The following parameters are set to produce a deterministic result
 randomSeed: 1337,
 concurrency: 1,
 sampleRate: 1.0,
 deltaThreshold: 0.0
})
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
```

	Person1	Person2	similarity
1	"Erik"	"Susan"	0.16666666666666666
2	"Susan"	"Erik"	0.16666666666666666
3	"Andrew"	"Ruth"	0.09090909090909091
4	"Johan"	"Ruth"	0.09090909090909091
5	"Martha"	"Johan"	0.09090909090909091
6	"Ruth"	"Johan"	0.09090909090909091
7	"Zack"	"Julia"	0.09090909090909091

Erik and Susan have most similarity(0.166) based on weight compared to others which we can see from the creation of cypher graph and the output of knn algorithm means that they are the nearest neighbour node(similar) to each other as both nodes have the same

similarity value from each other. Based on the property of a scalar number, the similarity between Erik and Susan is calculated as 1 / (1 + [ps – pt])= 1/(1+(50-45))=1/6= 0.1666 which is the same as similarity value in output table.

Andrew(100) and Ruth(90), Johan(80) and Ruth(90), Ruth(90) and Johan(80),Martha(70) and Johan(80) , Zack(20) and Julia(30) has similarity output of 0.909, are the next nearest neighbour nodes(i.e. Andrew is the nearest neighbour node to Ruth, Ruth and Johan are the nearest neighbour nodes, Marth and Johan are the nearest neighbour nodes, Zack is the nearest neighbour node to Julia) with similarity based on weight. Based on the property of a scalar number, the similarity is calculated as 1 / (1 + [ps – pt])= 1/(1+10)=1/11= 0.0909.

Task 5: - link prediction (supervised machine learning)

Merge (andrew:Teacher {name: 'Andrew', rating: 10})

Merge (martha:Teacher {name:'Martha', rating: 7})

Merge (susan:Teacher {name: 'Susan', rating: 5})

Merge (erik:Teacher {name:'Erik', rating: 2})

Merge (zack:Teacher {name:'Zack', rating: 1})

Merge (johan:Teacher {name:'Johan', rating: 8})

Merge (ruth:Teacher {name: 'Ruth', rating: 9})

Merge (julia:Teacher {name: 'Julia', rating: 3})

MERGE(andrew)-[:FOLLOWS]->(martha)

MERGE(martha)-[:FOLLOWS]->(susan)

MERGE(martha)-[:FOLLOWS]->(ruth)

MERGE(susan)-[:FOLLOWS]->(ruth)

MERGE(susan)-[:FOLLOWS]->(erik)

MERGE(ruth)-[:FOLLOWS]->(johan)

MERGE(erik)-[:FOLLOWS]->(ruth)

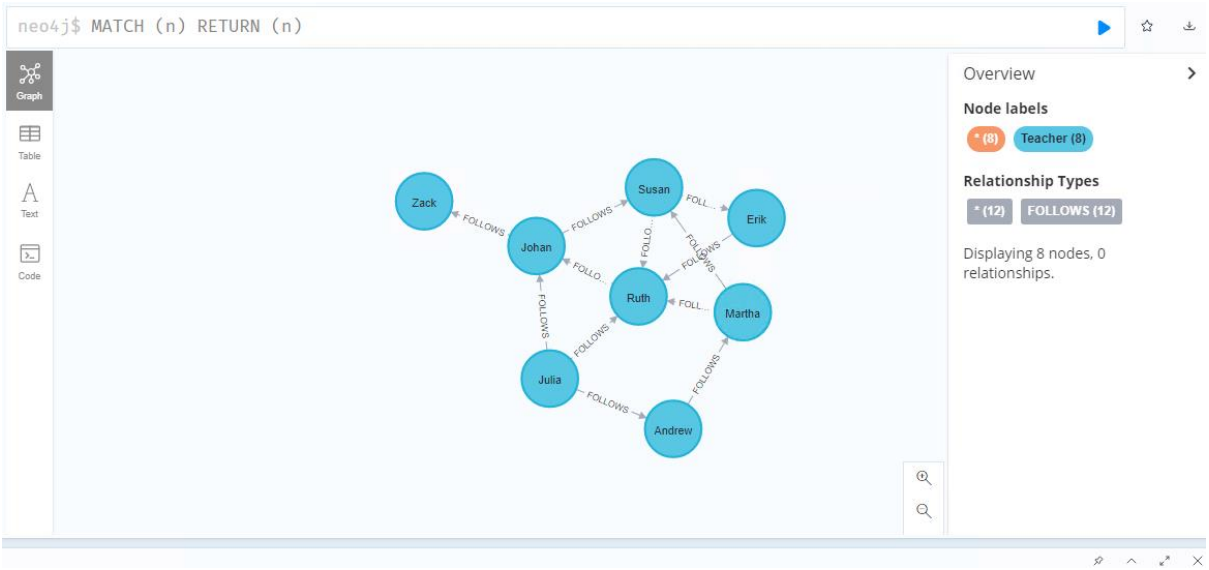
MERGE(johan)-[:FOLLOWS]->(susan)

MERGE(johan)-[:FOLLOWS]->(zack)

MERGE(julia)-[:FOLLOWS]->(andrew)

MERGE(julia)-[:FOLLOWS]->(ruth)

MERGE(julia)-[:FOLLOWS]->(johan)



Create named graph with native projection for analysis

CALL gds.graph.create('myProjectedGraph', { Teacher: { properties:'rating'} }, { FOLLOWS: { orientation: 'UNDIRECTED' } });

Run the gds.alpha.ml.splitRelationship procedure to split test and train graph data:

class ratio (q) = n(n-1)/2 = 8(8-1)/2=28

number of connections(r) = 12

negative samples ratio (q-r)/q = 28-12/12 = 1.33

Run the following for the first time to create FOLLOWS_TESTGRAPH

CALL gds.alpha.ml.splitRelationships.mutate('myProjectedGraph', {

relationshipTypes: ['FOLLOWS'],

remainingRelationshipType: 'FOLLOWS_REMAINING',

holdoutRelationshipType: 'FOLLOWS_TESTGRAPH',

holdoutFraction: 0.2,

negativeSamplingRatio: 1.33,

randomSeed: 1984

}) YIELD relationshipsWritten

The result found after running the above cypher was:- relationshipsWritten = 25

Run the following cypher to create FOLLOWS_REMAINING to create training set

CALL gds.alpha.ml.splitRelationships.mutate('myProjectedGraph', {

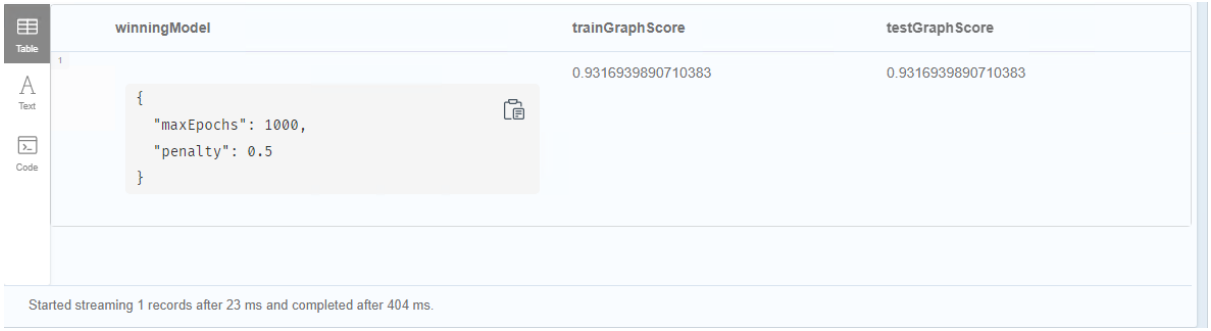
relationshipTypes: ['FOLLOWS_REMAINING'],

remainingRelationshipType: 'FOLLOWS_IGNORED_FOR_TRAINING',

```
holdoutRelationshipType: 'FOLLOWS_TRAINGRAPH',
holdoutFraction: 0.2,
negativeSamplingRatio: 1.33,
randomSeed: 1984
}) YIELD relationshipsWritten
```

Here, the relationshipsWritten = 20, now we have testing and training graphs so we can proceed with training models. Let us use 5 validation folds. Also, we know that the negativeSamplingRatio = 1.33, we can obtain the negativeClassWeight to 1/1.33 to balance both classes by assigning equal weight to both classes.

```
CALL gds.alpha.ml.linkPrediction.train('myProjectedGraph', {
  trainRelationshipType: 'FOLLOWS_TRAINGRAPH',
  testRelationshipType: 'FOLLOWS_TESTGRAPH',
  modelName: 'lp-numberOfRatings-model',
  featureProperties: ['rating'],
  validationFolds: 5,
  negativeClassWeight: 1.0 / 1.33,
  randomSeed: 2,
  concurrency: 1,
  params: [
    {penalty: 0.5, maxEpochs: 1000},
    {penalty: 1.0, maxEpochs: 1000},
    {penalty: 0.0, maxEpochs: 1000}
  ]
}) YIELD modelInfo
RETURN
  { maxEpochs: modelInfo.bestParameters.maxEpochs, penalty:
modelInfo.bestParameters.penalty } AS winningModel,
  modelInfo.metrics.AUCPR.outerTrain AS trainGraphScore,
  modelInfo.metrics.AUCPR.test AS testGraphScore
```



Run the algorithms on the stream mode.

CALL gds.alpha.ml.linkPrediction.predict.stream('myProjectedGraph', {

relationshipTypes: ['FOLLOWS'],

modelName: 'lp-numberOfRatings-model',

topN: 10,

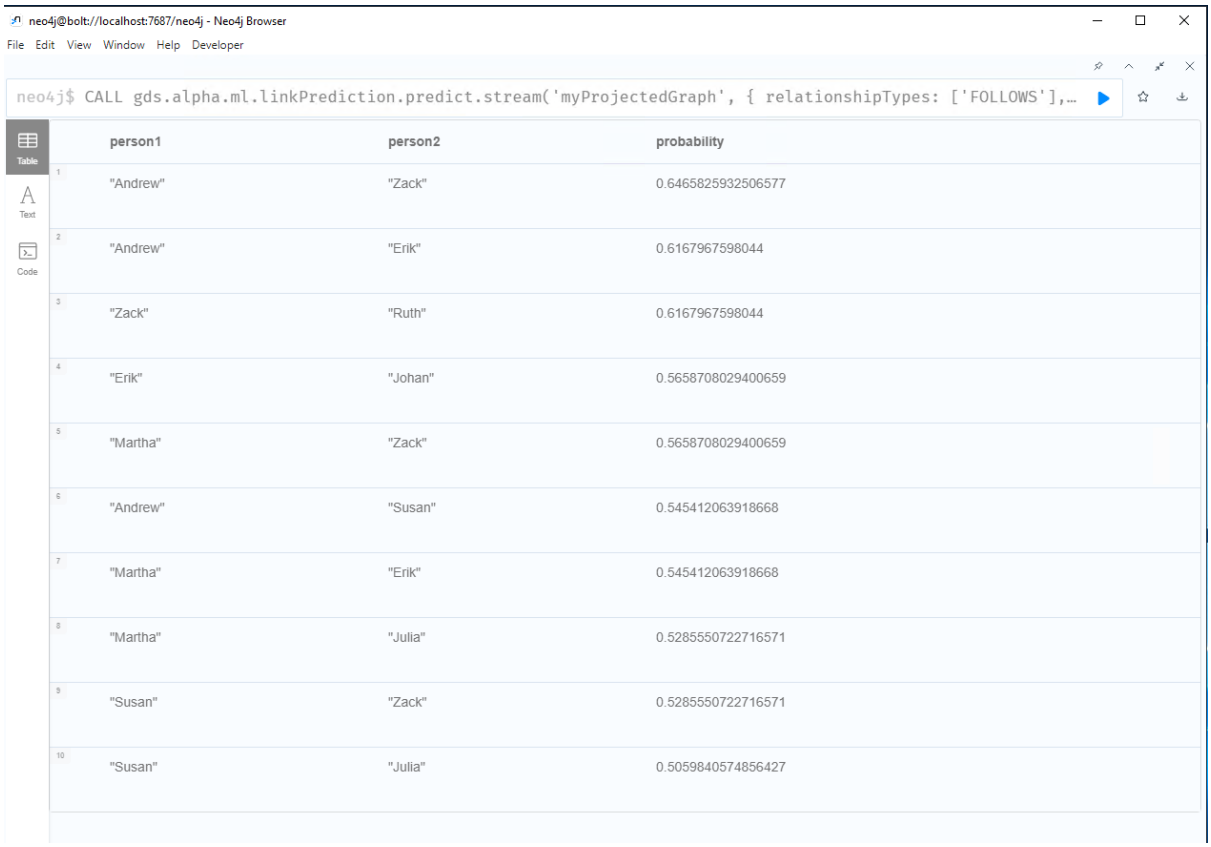
threshold: 0.45

})

YIELD node1, node2, probability

RETURN gds.util.asNode(node1).name AS person1, gds.util.asNode(node2).name AS person2, probability

ORDER BY probability DESC, person1



We specified threshold to filter out predictions with probability less than 45%, and topN to further limit output to the top 10 relationships. Note that the predicted link between the Martha and Julia,suan and jolia nodes does not reflect any particular direction between them. Julia is outcasted node with no inwards only communicator etccc.