Hands-On: Connectivity Analytics in Neo4j with Cypher

In this activity, we're going to go perform Connectivity Analytics using Cypher. Before starting, complete the following steps:

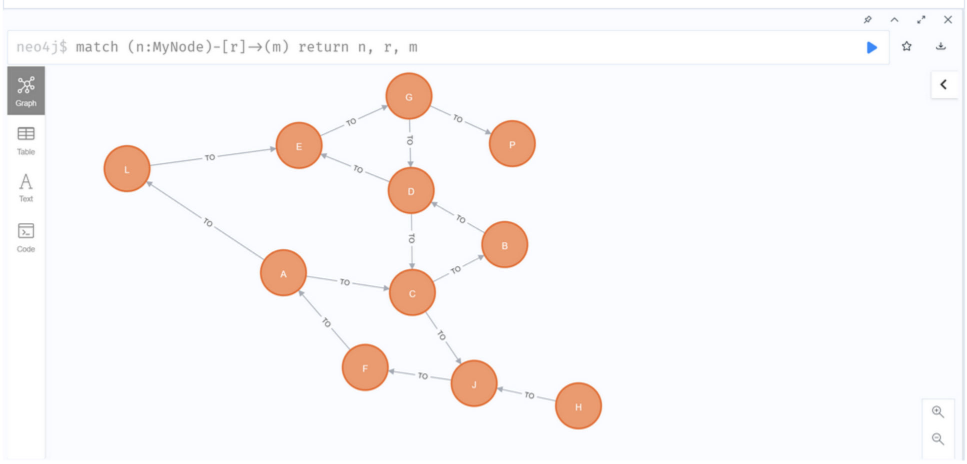
1. Start your Neo4j container and open it in https://localhost:7474/browser/.

2. Load the test.csv data

```
LOAD CSV WITH HEADERS FROM "file:///datasets/test.csv" AS line
MERGE (n:MyNode {Name: line.Source})
MERGE (m:MyNode {Name: line.Target})
MERGE (n) -[:TO {dist: toInteger(line.distance)}]-> (m)
```

Query 1. Viewing the graph. Let's start by displaying *MyNode* again.

```
match (n:MyNode)-[r]->(m)
return n, r, m
```



Query 2. Find the outdegree of all nodes.

```
match (n:MyNode)-[r]->()
return n.Name as Node, count(r) as Outdegree
order by Outdegree
union
match (a:MyNode)-[r]->(leaf)
where not((leaf)-->())
return leaf.Name as Node, 0 as Outdegree
```

Our first match statement finds all nodes with outgoing edges. We then return the names of the nodes and the number as the variable Outdegree. For convenience, we order by outdegree. Then we need to combine this with a specific query that deals with leaf nodes. So we match all leaf nodes and return the name and the value zero for its outdegree.


neo4j$ match (n:MyNode)-[r]→() return n.Name as Node, count(r) as Outdegree order by Outdegree union match (a:MyNode)-[r]→(l…

"Node"	"Outdegree"
"B"	1
"E"	1
"F"	1
"H"	1
"L"	1
"J"	1
"A"	2
"C"	2
"D"	2
"G"	2
"P"	0

Node *P* has an outdegree of 0, and all the other nodes are as you'd expect, ordered by their outdegree value.

Query 3. Find the indegree of all nodes.

```
match (n:MyNode)<-[r]-()
return n.Name as Node, count(r) as Indegree
order by Indegree
union
match (a:MyNode)<-[r]-(root)
where not((root)<--())
return root.Name as Node, 0 as Indegree
```

This new query is very similar to the previous one, but this time we're going to consider root nodes instead of leaf nodes.


neo4j$ match (n:MyNode)←[r]-() return n.Name as Node, count(r) as Indegree order by Indegree union match (a:MyNode)←[r]-(roo…

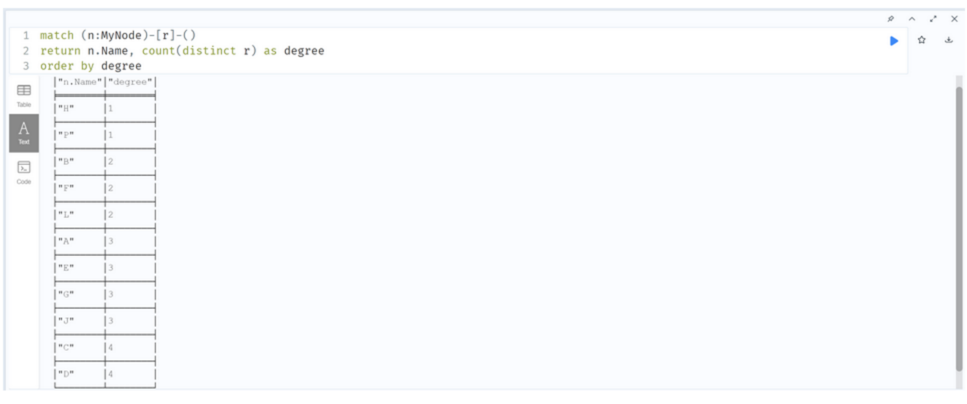
"Node"	"Indegree"
"A"	1
"B"	1
"F"	1
"G"	1
"L"	1
"P"	1
"C"	2
"D"	2
"E"	2
"J"	2
"H"	0

Node *H* has an indegree of 0, and all the other nodes are as you'd expect, ordered by their indegree value.

Query 4. Find the degree of all nodes.

```
match (n:MyNode)-[r]-()
return n.Name, count(distinct r) as degree
order by degree
```

In this case, we don't include a specific direction in our match statement, and we return the name and count for all of our edges. However, we do use the *distinct* statement, otherwise we'd be counting some nodes twice. Finally, for convenience, we order our result by the value of *degree.*


```
match (n:MyNode)-[r]-()
return n.Name, count(distinct r) as degree
order by degree
```

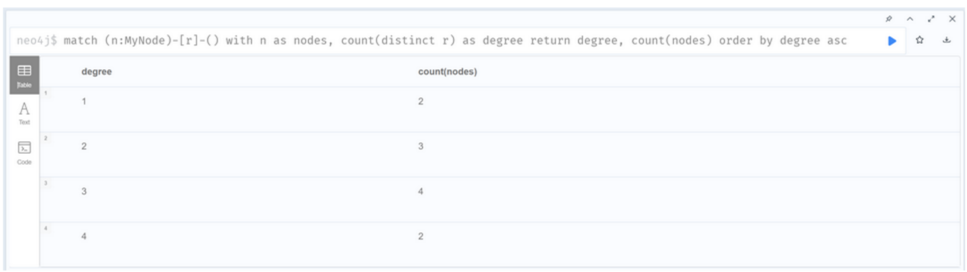
"n.Name"	"degree"
"H"	1
"P"	1
"B"	2
"F"	2
"L"	2
"A"	3
"E"	3
"G"	3
"J"	3
"C"	4
"D"	4

The values are as we would expect. We have a leaf node *P* with the degree of 1, and a root node *H* with a degree of 1.

Query 5. Find degree histogram of the graph.

```
match (n:MyNode)-[r]-()
with n as nodes, count(distinct r) as degree
return degree, count(nodes) order by degree asc
```

This time, instead of returning the number of degrees for each single node, we are going to group the count of nodes with the same degree.

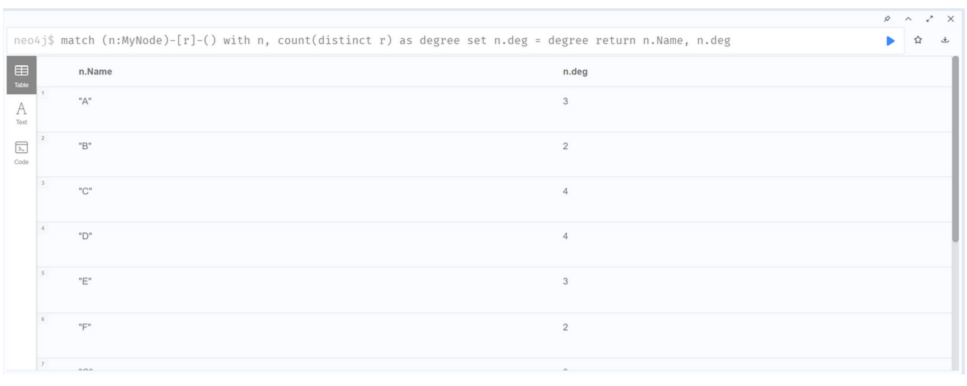

neo4j$ match (n:MyNode)-[r]-() with n as nodes, count(distinct r) as degree return degree, count(nodes) order by degree asc

degree	count(nodes)
1	2
2	3
3	4
4	2

Query 6. Save the degree of the node as a new node property.

```
match (n:MyNode)-[r]-()
with n, count(distinct r) as degree
set n.deg = degree
return n.Name, n.deg
```

Our first two lines of code are the same as in the two previous queries. However, this time we use our variable *degree* to set a new property to the nodes called *deg.* Finally, we return each node and its new property.


neo4j$ match (n:MyNode)-[r]-() with n, count(distinct r) as degree set n.deg = degree return n.Name, n.deg

n.Name	n.deg
"A"	3
"B"	2
"C"	4
"D"	4
"E"	3
"F"	2

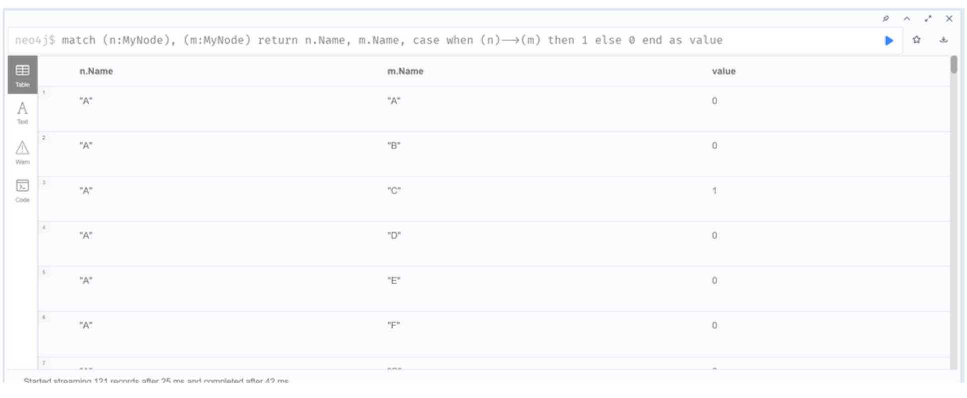
Philosophical Issue: Computational limitations of all databases.

Before we get to the last two examples, there's a philosophical issue that we need to remember with all databases. Any database will allow you to do some analytical computations, and the rest of the analytical computations will have to be done outside the database. However, it is always a good idea to get the database to produce an intermediate result that is formatted in a way that you need for the next computation. And then use that intermediate result as input for the next computation. We've seen that a number of computations in graph analytics start with the adjacency matrix. So we should be able to force Cypher to produce an adjacency matrix.

Query 7. Construct the Adjacency Matrix of the graph.

```
match (n:MyNode), (m:MyNode)
return n.Name, m.Name,
case
when (n)-->(m) then 1
else 0
end as value
```

Think of a matrix as a three-column table, where there's one column (node *n*), there's another column (node *m*), and the third column will be the values that we calculate when we determine whether two nodes have an edge between them. And we're introducing a new construct in Cypher called *case*. This allows us to evaluate conditions and return one result or a different result depending on the condition. We'll specify that if there is an edge between nodes *n* and *m*, then we'll return a value of 1, otherwise we'll return a value of 0. And we'll output those results as a value.


neo4j$ match (n:MyNode), (m:MyNode) return n.Name, m.Name, case when (n)⟶(m) then 1 else 0 end as value

n.Name	m.Name	value
"A"	"A"	0
"A"	"B"	0
"A"	"C"	1
"A"	"D"	0
"A"	"E"	0
"A"	"F"	0

Query 8. Construct the Normalized Laplacian Matrix of the graph

```
match (n:MyNode), (m:MyNode)
return n.Name, m.Name,
case
when n.Name = m.Name then 1
when (n)-->(m) then -1/(sqrt(toInteger(n.deg))*sqrt(toInteger(m.deg)))
else 0
end as value
```

We're going to do something very similar to the previous example. We're going to match all the nodes for the first column and all the nodes for the second column. We'll return the names of those nodes, and then we'll use the case structure again to compare the names of each node and determine if we have the same node. If we have the same node, then this is a diagonal of the matrix and should get a value of 1. If they are different nodes and contain an edge between them, then we compute the normalized Laplacian. You'll notice that we're using the actual degree property that we assigned to the nodes in a previous example. Finally, we if they are different nodes and there is no edge between them, we assign a value of 0, and we return that value.


neo4j$ match (n:MyNode), (m:MyNode) return n.Name, m.Name, case when n.Name = m.Name then 1 when (n)⟶(m) then -1/(sqrt(toInt…

n.Name	m.Name	value
"A"	"A"	1
"A"	"B"	0
"A"	"C"	-0.2886751345948129
"A"	"D"	0
"A"	"E"	0
"A"	"F"	0

Started streaming 121 records after 48 ms and completed after 58 ms.