**SPOKE and NASA GeneLab Integration – Notes and Guide**

The SPOKE database, hosted by Neo4j, can be used as an analytical reference for operational datasets containing experimental data. The Neo4j platform is used to interface with the vast collection of connected SPOKE reference databases with the Cypher Query Language, a language based on SQL but specific for graph databases. Cypher is designed to use descriptive text to represent **–[:relationships]->** between **(nodes)**. Node types can be assigned containing different classes of data (e.g., Mission, Study, Results, Genes, Proteins) and each node defined by a label contains {properties}. Similarly, relationships (e.g., Up Regulation, Down Regulation) contain properties such as log2FC and Adjusted p value.

**Resources**

See [Getting Started](https://neo4j.com/docs/getting-started/cypher-intro/) for an introduction to Cypher language or the [Cypher Manual](https://neo4j.com/docs/cypher-manual/current/introduction/) for complete documentation.

Visit the [Neo4j Graph Academy](https://graphacademy.neo4j.com/?_gl=1*lgv80f*_ga*MTQ4MjQ0Njg1NS4xNjkwMTUzNTQ3*_ga_DL38Q8KGQC*MTY5MTE3MDEzMy4yMi4xLjE2OTExNzIwMzUuNTguMC4w&_ga=2.106043587.1557279067.1691170135-1482446855.1690153547) for foundational courses.

Additionally, the [Neo4j Sandbox](https://neo4j.com/sandbox/) provides guided tutorials working in projects querying with both in Cypher and a graphical user interface. This approach was the fastest and most intuitive way to learn Cypher and the capabilities of Neo4j, without reading the full manual.

**Cypher and Graph Database Structures**

The Cypher Query Language was designed for graph databases and to be easily adapted to for those with experience in SQL database queries. The graphical nature of the language and relationships makes it simple to learn even without database management experience. From a technical standpoint, Cypher is pitched as being a much simpler and faster form of exploring relationships in databases. The primary example from Neo4Jj is Cypher requiring significantly less lines of code to achieve the same relationship expression as an SQL query. This is achieved with nodes connected by edges, which themselves contain relational data and properties. Edges are the [:relationships] used to define queries while nodes contain specific data or results.

Highly connected data that is spread across many tables, such as in a multi-omics investigation, is well suited to graphical database representation. In such an investigation we are interested in end to end relationships of multiple datasets and need to quickly search for connections across tables. Graphical databases also enable analysis of such connections and even prediction of new ones based on existing relationships. Finally graph databases and concurrent analyses can be rapidly updated with additional data and relationships.

**Examples**

CALL {

USE compositenasa.glds

MATCH (n0:Study {identifier:'GLDS-101'})-[]-(:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(p:Gene)

WITH COLLECT(p.identifier) AS glds\_pids

RETURN glds\_pids

}

UNWIND glds\_pids AS glds\_pid

CALL{

WITH glds\_pid

USE compositenasa.human

MATCH path=(:Anatomy)-[:UPREGULATES\_AuG]-(p:Gene{identifier:glds\_pid})

RETURN path

}

RETURN path LIMIT 10

CALL {

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WITH COLLECT(g.identifier) AS glds\_gids

RETURN glds\_gids

}

CALL{

USE compositenasa.glds

MATCH (n0:Study {identifier:'GLDS-99'})-[]-(:Result)-[:UPREGULATION\_RuMP]-(:MouseProtein)-[]-(p:Protein)

WITH COLLECT(p.identifier)AS glds\_pids

RETURN glds\_pids

}

UNWIND glds\_pids AS glds\_pid

UNWIND glds\_gids AS glds\_gid

CALL{

WITH glds\_pid, glds\_gid

USE compositenasa.human

MATCH path=(:Gene{identifier:glds\_gid})-[ENCODES\_GeP]->(:Protein{identifier:glds\_pid})-[]->(:Disease)

RETURN path

}

RETURN path LIMIT 75

Making Graph Data Sciences projection graph:

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WITH gds.graph.project('gldsG1','source','target')as g

RETURN g.graphName as graph, g.nodeCount AS nodes, g.relationshipCount AS rels

PageRank algorithm

Working out the right way to limit nodes to just genes

USE compositenasa.glds

CALL gds.pageRank.stream('gldsG2', {nodeLabels:[labels(Gene)]})

YIELD nodeId, score AS pageRank

RETURN gds.util.asNode(nodeId).identifier AS personId, pageRank

Inclusion of node labels in graph projection:

CALL{

USE compositenasa.glds

MATCH (r0:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WITH gds.graph.project('gldsG2',r0,g,{sourceNodeLabels:labels(r0),targetNodeLabels:labels(g)}) AS a

RETURN a

}

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

Specific Graph projection

CALL{

USE compositenasa.glds

MATCH (r0:Result)-[:UPREGULATION\_RuMG]-(mg:MouseGene)-[]-(g:Gene)

WITH gds.graph.project('gldsG3',r0,mg,{sourceNodeLabels:labels(r0),targetNodeLabels:labels(mg)}) AS a

RETURN a

}

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

PageRank algorithm

USE compositenasa.glds

CALL gds.pageRank.stream('gldsG3', {nodeLabels:["MouseGene"]})

YIELD nodeId, score AS pageRank

RETURN gds.util.asNode(nodeId).name AS GeneName, pageRank

Graph projection:

CALL {

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[r:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WHERE r.pvalue <.00000001

WITH COLLECT(g.identifier) AS glds\_gids

RETURN glds\_gids

}

UNWIND glds\_gids AS glds\_gid

CALL{

WITH glds\_gid

USE compositenasa.human

MATCH (g0:Gene {identifier:glds\_gid})-[m:MARKER\_POS\_GmpD]->(d:Disease)

RETURN g0, d

}

WITH gds.graph.project('gldsG5',g0,d,{sourceNodeLabels:labels(g0),targetNodeLabels:labels(d)}) AS a

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

Graph projection:

CALL {

USE compositenasa.human

MATCH (g0:Gene)-[ENCODES\_GeP]->(:Protein)-[]->(d:Disease)

WITH gds.graph.project('SPOKEgeneDisease',g0,d,{sourceNodeLabels:labels(g0),targetNodeLabels:labels(d)})AS a

RETURN a

}

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

Centrality algorithm:

CALL {

USE compositenasa.human

CALL gds.degree.stream('SPOKEgeneDisease')

YIELD nodeId, score AS score

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

}

RETURN GeneID, score

ORDER BY score DESC

Graph:

CALL {

USE compositenasa.human

MATCH (g0:Gene)-[:ENCODES\_GeP]->(p:Protein)-[]->(d:Disease)

WITH gds.graph.project('SPOKEgeneDisease2',g0,d,p,{sourceNodeLabels:labels(g0),targetNodeLabels:labels(d)})AS a

RETURN a

}

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

USE compositenasa.human

MATCH (g0:Gene)--(d0:Disease)--(p0:Protein)--(g0:Gene)

WITH gds.graph.project('geneProtDisease7',g0,d0,{sourceNodeLabels:labels(g0),targetNodeLabels:labels(d0)})AS a

RETURN a

CALL {

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WITH COLLECT(g.identifier) AS glds\_gids

RETURN glds\_gids

}

CALL{

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMP]-(:MouseProtein)-[]-(p:Protein)

WITH COLLECT(p.identifier)AS glds\_pids

RETURN glds\_pids

}

UNWIND glds\_pids AS glds\_pid

UNWIND glds\_gids AS glds\_gid

CALL{

WITH glds\_pid, glds\_gid

USE compositenasa.human

MATCH path=(g0:Gene{identifier:glds\_gid})-[:ENCODES\_GeP]->(p:Protein{identifier:glds\_pid})-[]->(d:Disease)--(:Gene)

RETURN path}

RETURN path LIMIT 100

**Graph project of SPOKE:**

USE compositenasa.human

MATCH (g0:Gene)--(d0:Disease)--(p0:Protein)--(g0:Gene)

WITH gds.graph.project('geneProtDisease7',g0,d0,{sourceNodeLabels:labels(g0),targetNodeLabels:labels(d0)})AS a

RETURN a

**And degree centrality scoring:**

USE compositenasa.human

CALL gds.degree.stream('geneProtDisease7')

YIELD nodeId, score

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

ORDER BY score DESC

**And results:**

|  |  |
| --- | --- |
| id | score |
| "CD4" | 14 |
| "CD8A" | 13 |
| "CDH1" | 9 |
| "VEGFA" | 6 |
| "KIT" | 6 |

**Graph NASA data mapped to spoke:**

CALL {

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WITH COLLECT(g.identifier) AS glds\_gids

RETURN glds\_gids

}

CALL{

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMP]-(:MouseProtein)-[]-(p:Protein)

WITH COLLECT(p.identifier)AS glds\_pids

RETURN glds\_pids

}

UNWIND glds\_pids AS glds\_pid

UNWIND glds\_gids AS glds\_gid

CALL{

WITH glds\_pid, glds\_gid

USE compositenasa.human

MATCH path=(g0:Gene{identifier:glds\_gid})--(p:Protein{identifier:glds\_pid})--(d:Disease)--(:Gene)

RETURN path

}

RETURN path

**Projection:**

CALL {

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WITH COLLECT(g.identifier) AS glds\_gids

RETURN glds\_gids

}

CALL{

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMP]-(:MouseProtein)-[]-(p:Protein)

WITH COLLECT(p.identifier)AS glds\_pids

RETURN glds\_pids

}

UNWIND glds\_pids AS glds\_pid

UNWIND glds\_gids AS glds\_gid

CALL{

WITH glds\_pid, glds\_gid

USE compositenasa.human

MATCH (g0:Gene{identifier:glds\_gid})--(p:Protein{identifier:glds\_pid})--(d:Disease)--(:Gene)

WITH gds.graph.project('geneProtDisease9',g0,d,{sourceNodeLabels:labels(g0),targetNodeLabels:labels(d)})AS a

RETURN a

}

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

**A diagram of a network

Description automatically generated**

**Backing up to just genelab data on NASA server… generating graph, and including relationship properties:**

USE compositenasa.glds

MATCH (r:Result)-[u:UPREGULATION\_RuMP|DOWNREGULATION\_RdMP]-(g:MouseProtein)--(g0:Protein)

WITH gds.graph.project('gldsProteins5',r,g,

{

sourceNodeLabels:labels(r),

targetNodeLabels:labels(g),

relationshipProperties:u{.log2fc}

}

)AS a

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

**What this graph looks like:**

**A screenshot of a computer screen

Description automatically generated**

**Algortihm on it:**

USE compositenasa.glds

CALL gds.louvain.stream('gldsProteins5',{ includeIntermediateCommunities: true })

YIELD nodeId, communityId, intermediateCommunityIds

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

ORDER BY communityId ASC

**Results:**

|  |  |  |
| --- | --- | --- |
| name | communityId | intermediateCommunityIds |
| null | 6 | [9, 14, 12, 11, 1, 2, 3, 4, 5, 6] |
| "HMGN1\_MOUSE" | 6 | [1, 1, 1, 1, 1, 2, 3, 4, 5, 6] |
| "OTUD4\_MOUSE" | 6 | [2, 2, 2, 2, 2, 2, 3, 4, 5, 6] |
| "NCALD\_MOUSE" | 6 | [3, 3, 3, 3, 3, 3, 3, 4, 5, 6] |
| "PPA5\_MOUSE" | 6 | [4, 4, 4, 4, 4, 4, 4, 4, 5, 6] |
| "ZG16\_MOUSE" | 6 | [5, 5, 5, 5, 5, 5, 5, 5, 5, 6] |
| "CBPA3\_MOUSE" | 6 | [6, 6, 6, 6, 6, 6, 6, 6, 6, 6] |
| "CTRB1\_MOUSE" | 6 | [9, 14, 12, 11, 1, 2, 3, 4, 5, 6] |
| "CEL2A\_MOUSE" | 6 | [11, 11, 11, 11, 1, 2, 3, 4, 5, 6] |
| "A1AT1\_MOUSE" | 6 | [12, 12, 12, 11, 1, 2, 3, 4, 5, 6] |
| "VGF\_MOUSE" | 6 | [14, 14, 12, 11, 1, 2, 3, 4, 5, 6] |

**Algorithm finds communities of genes clustered on each study, study names don’t come up… node id label for study is different than MouseProtein label. Also for some reason couldn’t get Protein names to come up, only showed as null.**

**After adding node identifier as another return column in the algorithm: we can see the study labels too.**

|  |  |  |  |
| --- | --- | --- | --- |
| name | study | communityId | intermediateCommunityIds |
| null | "GLDS-98\_Protein\_DEA\_FLT\_v\_GC." | 6 | [9, 14, 12, 11, 1, 2, 3, 4, 5, 6] |
| "HMGN1\_MOUSE" | "P18608" | 6 | [1, 1, 1, 1, 1, 2, 3, 4, 5, 6] |
| "OTUD4\_MOUSE" | "B2RRE7" | 6 | [2, 2, 2, 2, 2, 2, 3, 4, 5, 6] |
| "NCALD\_MOUSE" | "Q91X97" | 6 | [3, 3, 3, 3, 3, 3, 3, 4, 5, 6] |
| "PPA5\_MOUSE" | "Q05117" | 6 | [4, 4, 4, 4, 4, 4, 4, 4, 5, 6] |
| "ZG16\_MOUSE" | "Q8K0C5" | 6 | [5, 5, 5, 5, 5, 5, 5, 5, 5, 6] |
| "CBPA3\_MOUSE" | "P15089" | 6 | [6, 6, 6, 6, 6, 6, 6, 6, 6, 6] |
| "CTRB1\_MOUSE" | "Q9CR35" | 6 | [9, 14, 12, 11, 1, 2, 3, 4, 5, 6] |
| "CEL2A\_MOUSE" | "P05208" | 6 | [11, 11, 11, 11, 1, 2, 3, 4, 5, 6] |
| "A1AT1\_MOUSE" | "P07758" | 6 | [12, 12, 12, 11, 1, 2, 3, 4, 5, 6] |
| "VGF\_MOUSE" | "Q0VGU4" | 6 | [14, 14, 12, 11, 1, 2, 3, 4, 5, 6] |
| "HKDC1\_MOUSE" | "Q91W97" | 7 | [7, 7, 7, 7, 7, 7, 7, 7, 7, 7] |
| "S10AD\_MOUSE" | "P97352" | 8 | [8, 8, 8, 8, 8, 8, 8, 8, 8, 8] |
| "CIAO1\_MOUSE" | "Q99KN2" | 10 | [10, 10, 10, 10, 10, 10, 10, 10, 10, 10] |
| "CBPA1\_MOUSE" | "Q7TPZ8" | 13 | [13, 13, 13, 13, 13, 13, 13, 13, 13, 13] |
| "MED28\_MOUSE" | "Q920D3" | 15 | [15, 15, 15, 15, 15, 15, 15, 15, 15, 15] |
| "CELA1\_MOUSE" | "Q91X79" | 16 | [16, 16, 16, 16, 16, 16, 16, 16, 16, 16] |
| null | "GLDS-101\_Protein\_DEA\_FLT\_v\_GC." | 25 | [24, 18, 20, 21, 22, 23, 25, 25, 25, 25] |
| "ELOB\_MOUSE" | "P62869" | 25 | [18, 18, 20, 21, 22, 23, 25, 25, 25, 25] |
| "SNAA\_MOUSE" | "Q9DB05" | 25 | [20, 20, 20, 21, 22, 23, 25, 25, 25, 25] |
| "IF4A1\_MOUSE" | "P60843" | 25 | [21, 21, 21, 21, 22, 23, 25, 25, 25, 25] |
| "RMXL1\_MOUSE" | "Q91VM5" | 25 | [22, 22, 22, 22, 22, 23, 25, 25, 25, 25] |
| "TBB5\_MOUSE" | "P99024" | 25 | [23, 23, 23, 23, 23, 23, 25, 25, 25, 25] |
| "TBA1B\_MOUSE" | "P05213" | 25 | [24, 18, 20, 21, 22, 23, 25, 25, 25, 25] |
| "DPYL2\_MOUSE" | "O08553" | 25 | [25, 25, 25, 25, 25, 25, 25, 25, 25, 25] |
| null | "GLDS-102\_Protein\_DEA\_FLT\_v\_GC." | 36 | [27, 28, 29, 30, 31, 32, 33, 34, 35, 36] |

**Now to add relationship properties into calculation, ie log2fc:**

USE compositenasa.glds

CALL gds.louvain.stream('gldsProteins5',{ relationshipWeightProperty: 'log2fc' })

YIELD nodeId, communityId, intermediateCommunityIds

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

ORDER BY communityId ASC

**Results:**

|  |  |  |
| --- | --- | --- |
| name | study | communityId |
| "HMGN1\_MOUSE" | "P18608" | 1 |
| "OTUD4\_MOUSE" | "B2RRE7" | 2 |
| "NCALD\_MOUSE" | "Q91X97" | 3 |
| "PPA5\_MOUSE" | "Q05117" | 4 |
| "CBPA3\_MOUSE" | "P15089" | 6 |
| null | "GLDS-98\_Protein\_DEA\_FLT\_v\_GC." | 8 |
| "ZG16\_MOUSE" | "Q8K0C5" | 8 |
| "HKDC1\_MOUSE" | "Q91W97" | 8 |
| "S10AD\_MOUSE" | "P97352" | 8 |
| "CTRB1\_MOUSE" | "Q9CR35" | 8 |
| "CIAO1\_MOUSE" | "Q99KN2" | 8 |
| "CEL2A\_MOUSE" | "P05208" | 8 |
| "A1AT1\_MOUSE" | "P07758" | 8 |
| "CBPA1\_MOUSE" | "Q7TPZ8" | 8 |
| "VGF\_MOUSE" | "Q0VGU4" | 8 |
| "CELA1\_MOUSE" | "Q91X79" | 8 |
| "MED28\_MOUSE" | "Q920D3" | 15 |
| "ELOB\_MOUSE" | "P62869" | 18 |
| "TMOD1\_MOUSE" | "P49813" | 19 |
| "SNAA\_MOUSE" | "Q9DB05" | 20 |
| null | "GLDS-101\_Protein\_DEA\_FLT\_v\_GC." | 22 |

**New undirected graph including all data sets, protein, and transcript information. Allows algorithm to traverse entire graph. With directed relationships, algorithm returned betweenness score of 0 for all features.**

USE compositenasa.glds

MATCH (r:Result)-[u:UPREGULATION\_RuMP|DOWNREGULATION\_RdMP|UPREGULATION\_RuMG|DOWNREGULATION\_RdMG]-(g:MouseProtein|MouseGene)

WITH gds.graph.project('gldsProteins8',r,g,

{

sourceNodeLabels:labels(r),

targetNodeLabels:labels(g),

relationshipProperties:u{.log2fc}

},

{undirectedRelationshipTypes: ['\*']}

)AS a

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels

**Mutate: writing data to in memory graphs. Here adding betweenness**

USE compositenasa.glds

CALL gds.betweenness.mutate('gldsProteins8',{ relationshipWeightProperty: 'log2fc',mutateProperty:'betweenness' })

YIELD centralityDistribution, nodePropertiesWritten

RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore, nodePropertiesWritten

**Adding anatomy mapping:**

CALL {

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMG]-(:MouseGene)-[]-(g:Gene)

WITH COLLECT(g.identifier) AS glds\_gids

RETURN glds\_gids

}

CALL{

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMP]-(:MouseProtein)-[]-(p:Protein)

WITH COLLECT(p.identifier)AS glds\_pids

RETURN glds\_pids

}

USE compositenasa.glds

MATCH (n0:Study)-[]-(:Result)-[:UPREGULATION\_RuMP]-(:MouseProtein)-[]-(p:Protein)

WITH COLLECT(n0.Tissue)AS tissues

RETURN tissues

}

UNWIND glds\_pids AS glds\_pid

UNWIND glds\_gids AS glds\_gid

UNWIND tissues AS tissue

CALL{

WITH glds\_pid, glds\_gid,tissue

USE compositenasa.human

MATCH (g0:Gene{identifier:glds\_gid})--(p:Protein{identifier:glds\_pid})--(d:Disease)--(:Gene|Disease)--(:Anatomy{tissue:tissue})

WITH gds.graph.project('geneProtDisease9',g0,d,{sourceNodeLabels:labels(g0),targetNodeLabels:labels(d)})AS a

RETURN a

}

RETURN a.graphName AS graph, a.nodeCount AS nodes, a.relationshipCount AS rels