

# Assignment - 3, Neo4J :

neo4j Desktop

Data services

- Local instances
- Remote connections
- Import
- Query
- Explore
- About
- Settings

BDA-A3-2022594

STOPPING

ID: fddc1997-75db-4... Version: 2025.09.0 Path: /Users/bhavyach... Connect

Databases (0)

Bloom plugin for Enterprise 2.24.0

The Bloom Enterprise plugin requires a valid licence key and facilitates access to a remote Neo4j databases as well as features used for collaboration, such as persistent storage and sharing capabilities for commercial users. To register for a license, please contact your Neo4j representative. All other Bloom features are available for free in the Explore tool without installing the plugin.

Install Documentation

Gen AI 2025.09.0

Neo4j's Vector indexes and Vector functions allow you to calculate the similarity between node and relationship properties in a graph. A prerequisite for using these features is that vector embeddings have been set as properties of these entities. The GenAI plugin enables the creation of such embeddings using GenAI providers.

Install Documentation

Graph Data Science (Installed) 2.22.0

The Neo4j Graph Data Science (GDS) plugin provides extensive analytical capabilities centered around graph algorithms. The plugin includes algorithms for community detection, centrality, similarity, path finding, and node embeddings, as well as graph catalog procedures and machine learning pipelines designed to support data science workflows for graphs. All operations are designed for massive scale and parallelisation, with a custom and general API tailored for graph-global processing, and highly optimised compressed in-memory data structures.

Uninstall Documentation Github

neo4j-fleet-management-plugin 1.1.0-v2025

Install

install GDS plugin and restart session.

neo4j Desktop

Data services

- Local instances
- Remote connections
- Import
- Query
- Explore
- About
- Settings

Instance: BDA-A3-2022594 Database: neo4j CYCLES User: neo4j

Files

- edges.csv
- nodes.csv

... Run import ...

VOTED FOR

User

Definition Constraints & Indexes (2)

Label

Name: User

File

Name: nodes.csv

Properties

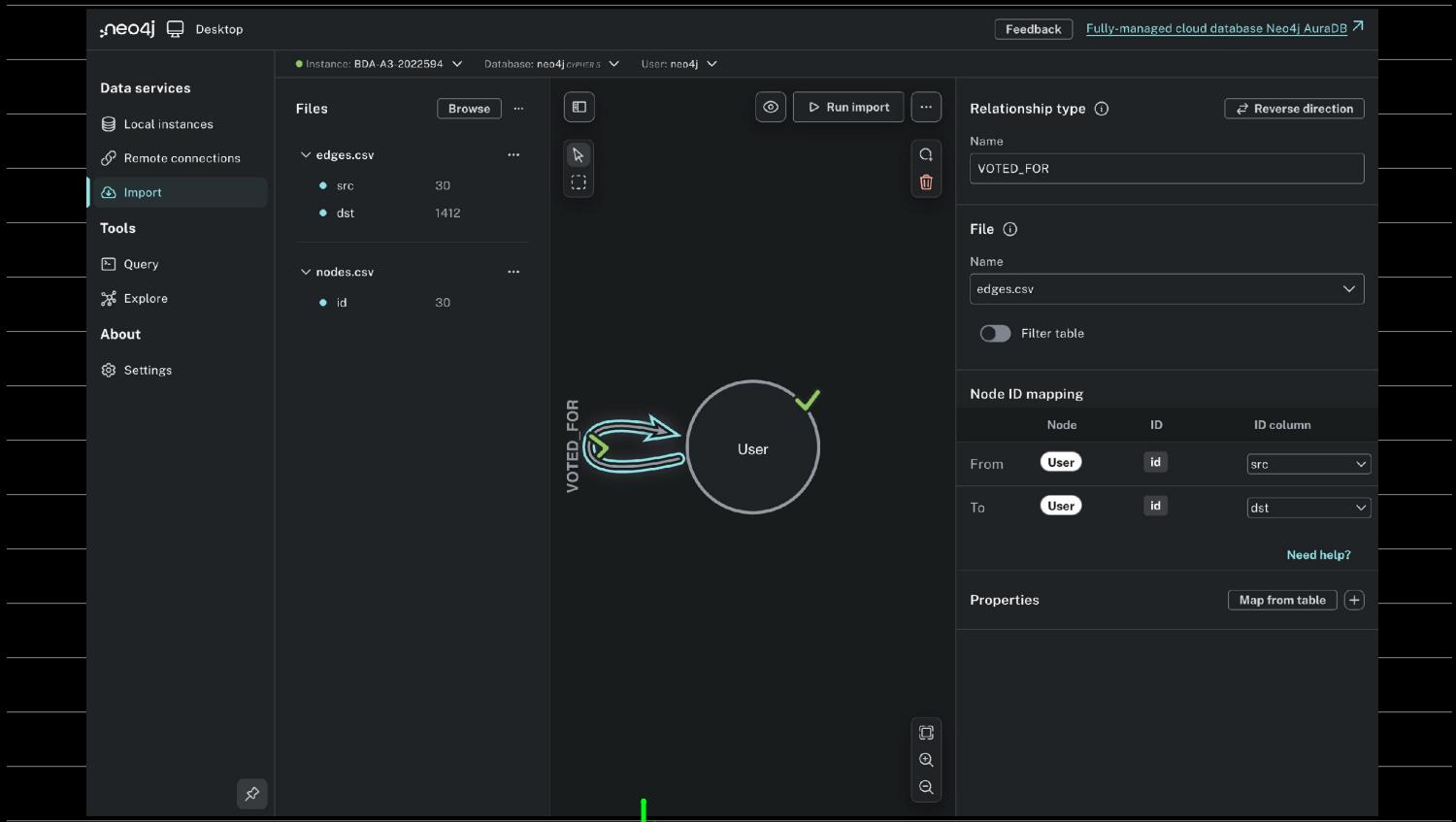
Name	Type	Column	ID
id	integer	id	

import all csv files after conversion  
and from txt file as below.  
and then run the following command

and start defining relationships.

```
import pandas as pd  
# Load edges  
edges = pd.read_csv("/Users/bhavychawla/Downloads/wiki-Vote.txt", sep="\t", comment="#", names=["src", "dst"])  
  
# --- Neo4j files ---  
edges.to_csv("/Users/bhavychawla/Downloads/edges.csv", index=False)  
nodes = pd.DataFrame(pd.unique(edges[['src', 'dst']].values.ravel()), columns=["id"])  
nodes.to_csv("/Users/bhavychawla/Downloads/nodes.csv", index=False)  
  
# --- Giraph file ---  
adj = edges.groupby("src")["dst"].apply(lambda x: " ".join(f"{i}:0" for i in x))  
adj.to_csv("/Users/bhavychawla/Downloads/giraph_input.txt", header=False)  
print("Generated: edges.csv, nodes.csv, giraph_input.txt")
```

↑ read the txt file  
↑ convert to csv files, one for edges other for nodes



↓

Define relationship from User to user (User is our nodes files) that has a relationship from, to also set primary keys, as src to dst.

neo4j Desktop

Instance: BDA-A3-2022594 Database: neo4j User: neo4j

Feedback Fully-managed cloud database Neo4j AuraDB ↗

Data services

- Local instances
- Remote connections
- Import

Tools

- Query
- Explore
- About
- Settings

Database information

Nodes (7,115)

User

Relationships (103,689)

VOTED\_FOR

Property keys

graphName nodeCount relationshipCount

"wikiGraphNew"	7115	103689
----------------	------	--------

Started streaming 1 record after 13 ms and completed after 33 ms.

neo4j\$ CALL gds.graph.project( 'wikiGraphNew', ['User'], { VOTED\_FOR: { type: 'VOTED\_FOR' } } )

Table RAW

type(r)	count(r)
"VOTED_FOR"	103689

Started streaming 1 record after 5 ms and completed after 27 ms.

Last update: 11:30:11

:welcome

GUIDE

Get to know the latest in Query/Browser

Take a quick tour to learn about the latest features to help you be productive.

DATA SET

Try Neo4j with the Movie Graph

Dive into a fully-built graph example to kickstart your Neo4j journey. Discover key query patterns through a fun familiar

The screenshot shows the Neo4j Desktop interface. The left sidebar has 'Query' selected. The main area shows 'Database information' with nodes (7,115) and relationships (103,689). A GDS project 'wikiGraphNew' is listed under 'Relationships'. Below it, a 'GUIDE' section is open with a 'Welcome' message and a 'DATA SET' section titled 'Try Neo4j with the Movie Graph'. At the bottom, there's a note about the last update.

project graph properties and  
connect session... ↴

```

1 // Drop old graph if it exists
2 CALL gds.graph.drop('wikiGraphDirected', false)
3 YIELD graphName
4 RETURN
5   graphName AS graph,
6   0 AS nodes,
7   0 AS edges,
8   0 AS total
9
10 UNION ALL
11
12 // Project and print node + edge counts
13 CALL gds.graph.project(
14   'wikiGraphDirected',
15   'User',
16   { VOTED_FOR: { orientation: 'NATURAL' } }
17 )
18 YIELD graphName, nodeCount, relationshipCount
19 RETURN
20   graphName AS graph,
21   nodeCount AS nodes,

```

**Table**

**RAW**

graph	nodes	edges	total
1 "wikiGraphDirected"	0	0	0
2 "wikiGraphDirected"	7115	103689	110804

Started streaming 2 records after 1 ms and completed after 9 ms.

↑ Actual value matches GT.

project graph in a directed fashion,  
 'NATURAL' stands for  
 directed graph...

also yield /return node count, edges and  
 total relationships.

```

3 YIELD graphName
4 RETURN
5   graphName AS graph,
6   0 AS nodes,
7   0 AS edges,
8   0 AS total
9
10 UNION ALL
11
12 // Project and print node + edge counts
13 CALL gds.graph.project(
14   'wikiGraphDirected',
15   'User',
16   { VOTED_FOR: { orientation: 'NATURAL' } }
17 )
18 YIELD graphName, nodeCount, relationshipCount
19 RETURN
20   graphName AS graph,
21   nodeCount AS nodes,
22   relationshipCount AS edges,
23   (nodeCount + relationshipCount) AS total;

```

return graphname, nodes and total edges.

```

1 CALL gds.wcc.stream('wikiGraphDirected')
2 YIELD nodeId, componentId
3 WITH componentId, count(*) AS size
4 ORDER BY size DESC LIMIT 1
5 RETURN size AS nodes_in_largest_WCC,
6       roundtoFloat(size) / 7115, 3 AS fraction_of_total;

```

Table RAW

Q {} ↴

nodes_in_largest_	fraction_of_total
7066	0.993

↑ matches GT values.

Started streaming 1 record after 18 ms and completed after 23 ms.

↓  
run wcc on <sup>in-memory</sup> directed graph.  
and return nodeId and component ID of  
each node. Group nodes based on Component  
ID's, and count their size, return the  
largest wcc., also compute fraction of total

```

1 CALL gds.wcc.stream('wikiGraphDirected')
2 YIELD nodeId, componentId → return nodeIDs and componentIDs.
3 WITH componentId, collect(nodeId) AS nodes
4 ORDER BY size(nodes) DESC LIMIT 1
5 WITH nodes
6 MATCH (u)-[r:VOTED_FOR]->(v)
7 WHERE id(u) IN nodes AND id(v) IN nodes
8 RETURN count(r) AS edges_in_largest_WCC,
9       roundtoFloat(count(r)) / 103689, 3 AS fraction_of_total;

```

Table RAW

Q {} ↴

edges_in_largest_	fraction_of_total
103663	1.0

↑ matches GT values

Started streaming 1 record after 45 ms and completed after 60 ms.

> ! 2 warnings

↓  
group by componentID and create a list of  
node ID's in that component,  
and find out the largest list,  
then using that list  
return count of number of  
relationships in the largest list,  
also calculate fraction of total.

```

1 CALL gds.scc.stream('wikiGraphDirected')
2 YIELD nodeId, componentId
3 WITH componentId, count(*) AS size
4 ORDER BY size DESC LIMIT 1
5 RETURN size AS nodes_in_largest_SCC,
6     roundtoFloat(size) / 7115, 3 AS fraction_of_total;

```

Table RAW



nodes_in_largest_	fraction_of_total
1300	0.183

↑ matches GT

Started streaming 1 record after 15 ms and completed after 21 ms.

calculate nodes in largest SCC,  
group by component ID's and  
count of components...  
return largest component and also  
return fraction of total.

```

1 CALL gds.scc.stream('wikiGraphDirected')
2 YIELD nodeId, componentId
3 WITH componentId, collect(nodeId) AS nodes
4 ORDER BY size(nodes) DESC LIMIT 1
5 WITH nodes
6 MATCH (u)-[r:VOTED_FOR]->(v)
7 WHERE id(u) IN nodes AND id(v) IN nodes
8 RETURN count(r) AS edges_in_largest_SCC,
9     roundtoFloat(count(r)) / 103689, 3 AS fraction_of_total;

```

Table RAW



edges_in_largest_	fraction_of_total
39456	0.381

↑ matches GT.

Started streaming 1 record after 26 ms and completed after 39 ms.

> 1 2 warnings

in the directed Graph run scc algorithm,  
group by nodeID, componentID's,  
create a list of nodes in components,  
in the largest component find all relationships  
in this component. Also return fraction  
of total edged.

```
1 CALL gds.graph.drop('wikiGraphUndirected', false);
2
3 CALL gds.graph.project(
4   'wikiGraphUndirected',
5   'User',
6   { VOTED_FOR: { orientation: 'UNDIRECTED' } }
7 )
8 YIELD graphName, nodeCount, relationshipCount
9 RETURN graphName, nodeCount, relationshipCount;
```

```
✓ CALL gds.graph.drop('wikiGraphUndirected', false);
```

```
✓ CALL gds.graph.project(
```



convert to a undirected projection.

```
1 CALL gds.localClusteringCoefficient.stream('wikiGraphUndirected')
2 YIELD nodeId, localClusteringCoefficient
3 RETURN avg(localClusteringCoefficient) AS average_clustering_coefficient;
```

Table RAW

average\_clustering\_coefficient

```
0.14089784589308788
```

Started streaming 1 record after 1 ms and completed after 26 ms.

↑ matches GT only slight error

↓ using clustering coefficient API/function  
yield node wise clustering coefficient  
then average out.

likely FP error in calculation of average.

```

1 CALL gds.triangleCount.stream('wikiGraphUndirected')
2 YIELD nodeId, triangleCount
3 RETURN sum(triangleCount) / 3.0 AS number_of_triangles;

```

Table RAW

number\_of\_triangles

1 608389.0



↑ matches GT value

Started streaming 1 record after 1 ms and completed after 24 ms.

↓  
using triangleCount API figure out  
node-wise triangle counts  
ie. how many triangles a node takes  
part in, take sum and since  
each triangle was counted thrice (part of  
final value.  
} 3 nodes) we divide 3, to get

count total Δs

```

1 // Step 1: total number of triangles
2 CALL {
3   CALL gds.triangleCount.stream('wikiGraphUndirected')
4   YIELD nodeId, triangleCount
5   RETURN sum(triangleCount) / 3.0 AS totalTriangles
6 }
7
8 // Step 2: total connected triples = sum(degree * (degree - 1) / 2)
9 CALL {
10   CALL gds.degree.stream('wikiGraphUndirected')
11   YIELD nodeId, degree AS degrees
12 }
13
14
15 // Step 3: compute global fraction
16 RETURN
17   totalTriangles AS number_of_triangles,
18   totalTriples AS number_of_triples,
19   (totalTriangles / totalTriples) AS fraction_closed_triples;

```

→ count total triples via formula

return open and closed triplets.  
calculate fraction of Δs.

off from GT

Table RAW

number\_of\_triangles

1 608389.0

number\_of\_triples

1 15890700.0

fraction\_closed\_tr

0.0382858527314  
71864



Started streaming 1 record after 1 ms and completed after 29 ms.

> 2 warnings

compute total triplets, ones with degree 3,  
and divide total Δs by total triplets  
and divide to find fraction of total by

dividing.



error due to scaling or sparsity  
of graph, a large number of  
high degree nodes generate  
many triplets of which many  
might be open, also with large  
graph size clustering drops.

```
1 // Estimate diameter and 90% effective diameter more accurately
2 MATCH (n:User)
3 WITH collect(id(n)) AS allNodes
4 UNWIND allNodes[..200] AS sourceNode      // increase sample size to 200
5 CALL gds.shortestPath.dijkstra.stream('wikiGraphUndirected', {
6   sourceNode: sourceNode,
7   targetNodes: allNodes,
8   relationshipWeightProperty: null
9 })
10 YIELD totalCost
11 WHERE totalCost IS NOT NULL AND totalCost > 0
12 RETURN
13   max(totalCost) AS diameter,
14   percentileCont(totalCost, 0.9) AS effective_diameter_90;
15
```



RAW



diameter ≡

effective\_diameter

↑ off from GT

6.0

4.0

Started streaming 1 record after 27 ms and completed after 1,350 ms.

> ! 01N01: Feature deprecated with replacement

fetch all user nodes in the Graph, store  
all nodes as a list,  
choose a sample of 200 nodes to  
approximate, run Dijkstra for each of  
the source nodes to find shortest  
distances to every other node.

totalCost. shortest path distances b/w nodes



return maximum — as diameter  
and one which covers 90% of all reachable  
node pairs.

difference is likely due to approximation using lesser node samples.

Limitations: 1) Neo4J is suitable for graphs upto few 100 million nodes, beyond that suffers performance issues

2). Neo4J keeps data in RAM, so large RAM required, otherwise large queries thrash due to less availability of RAM.

3). Free edition lacks 1st GDS execution, multiple concurrent graphs and advanced procedures.

