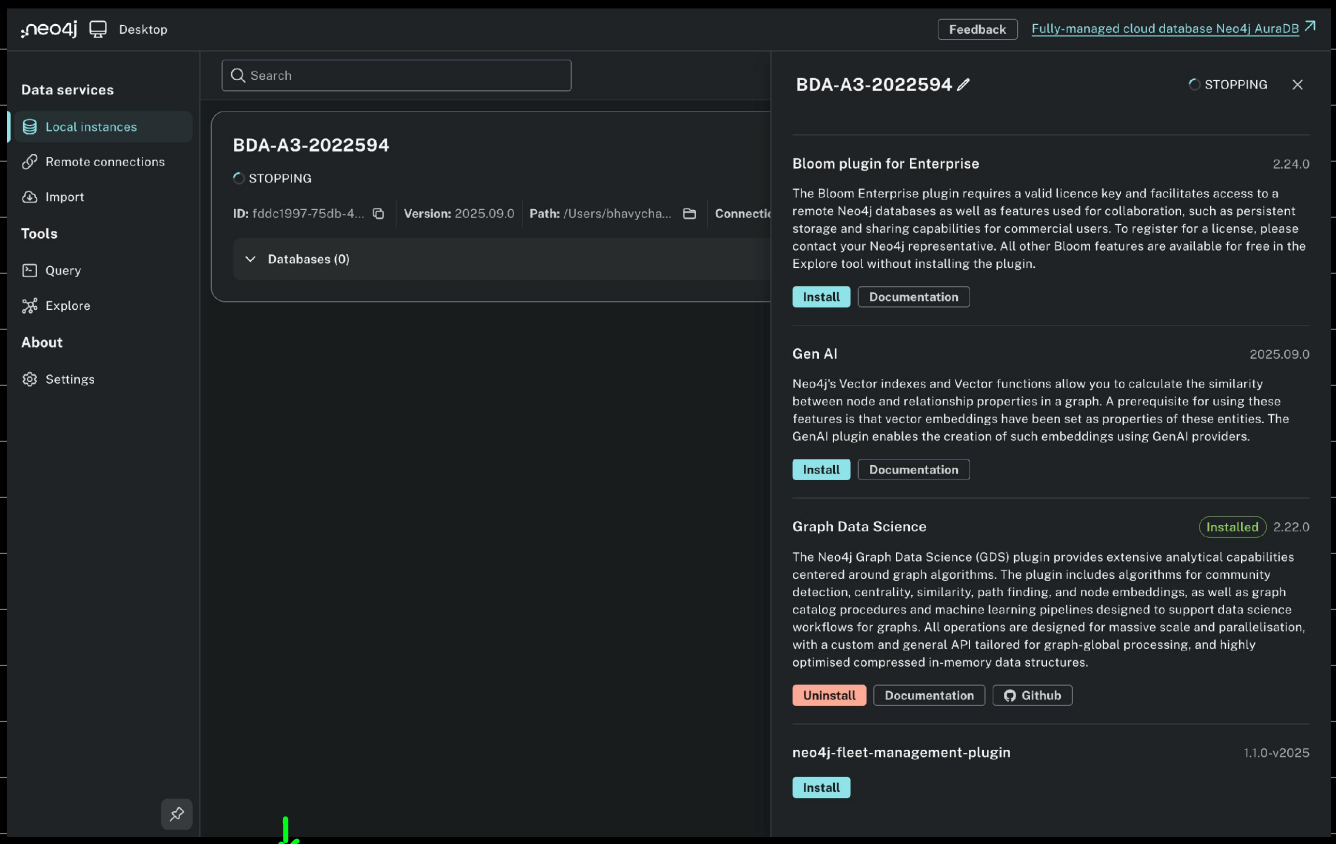
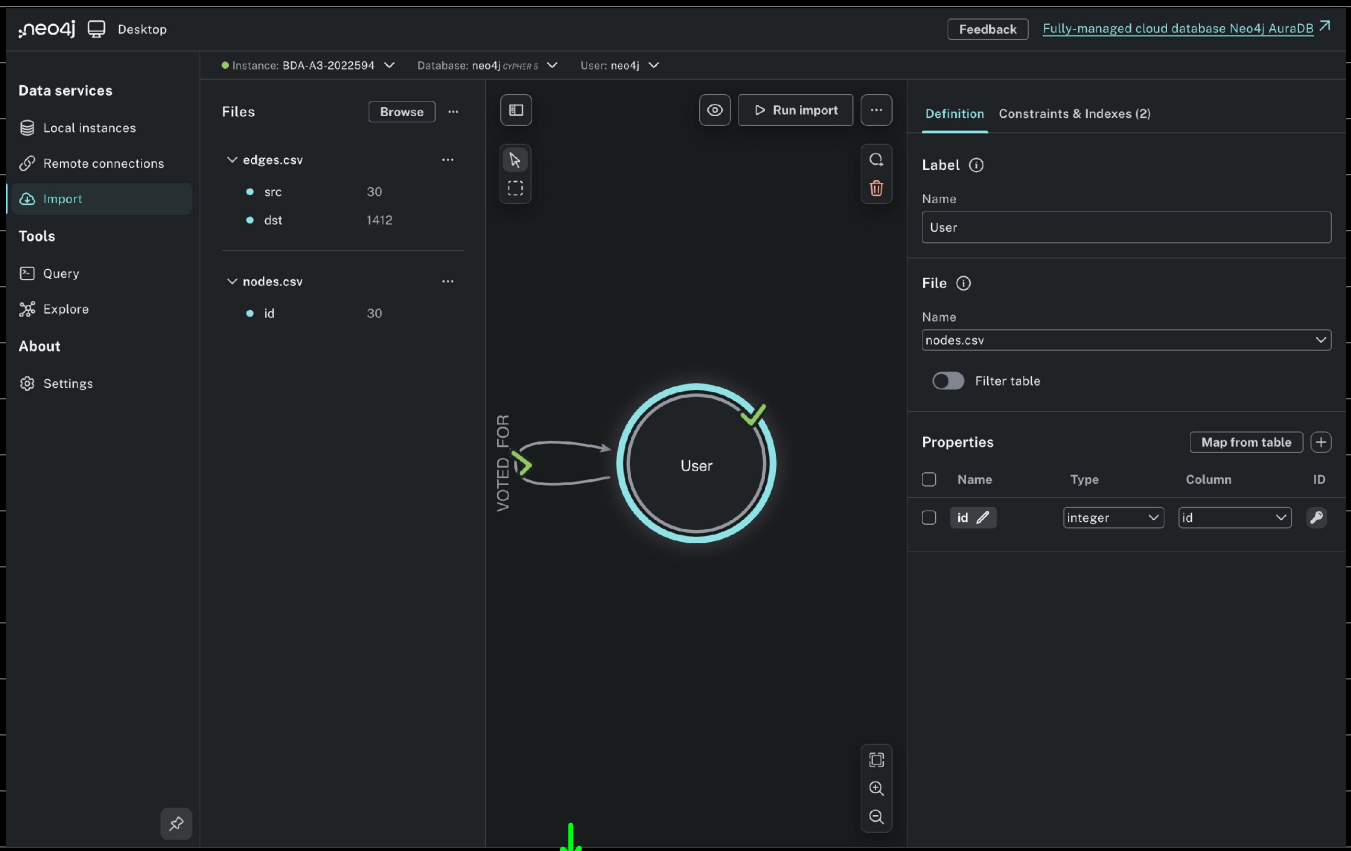
Assignment-3, Neo4J:



↓

install GDS Plugin and restart session.



↓

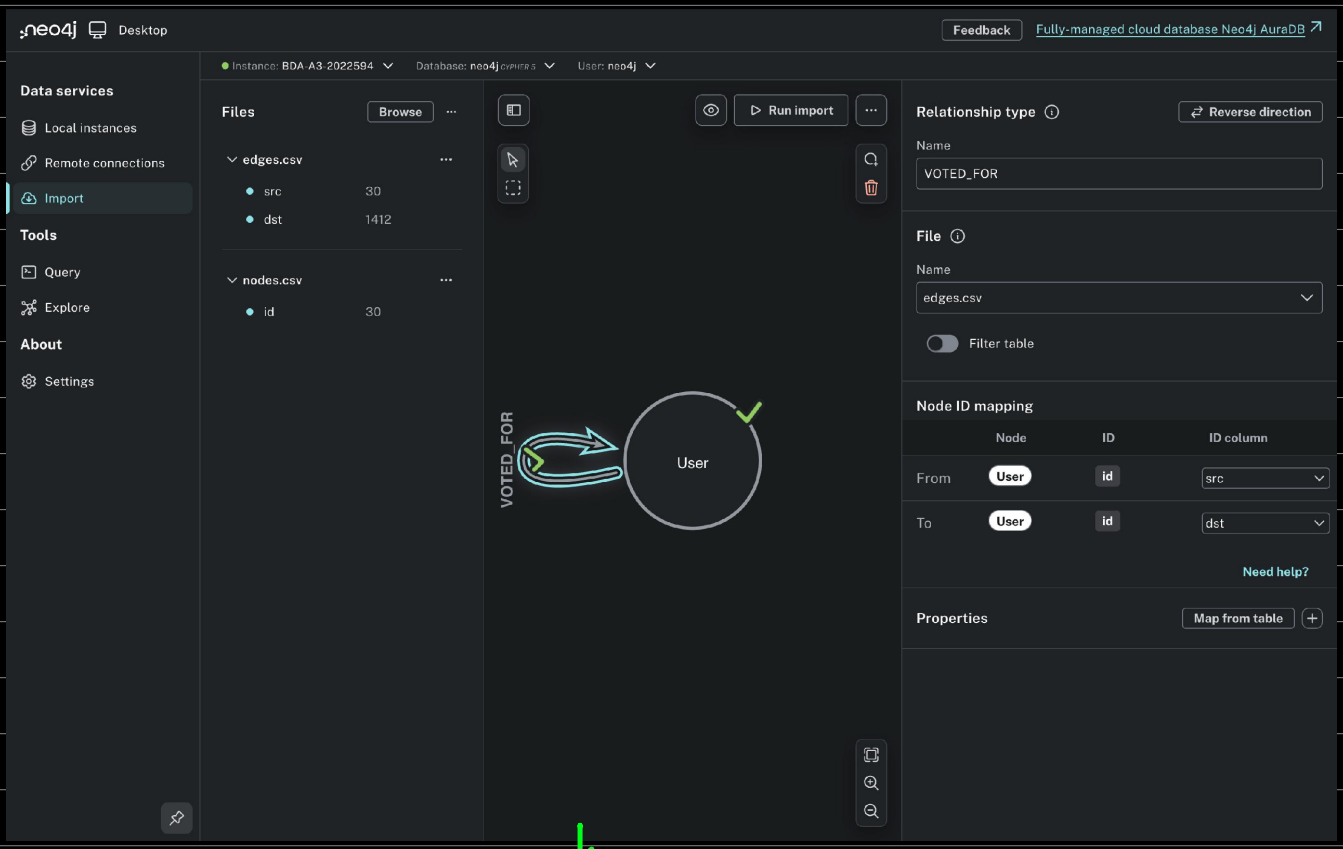
import all csv files after conversion from txt file as below.

and start defining relationships.

```python
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

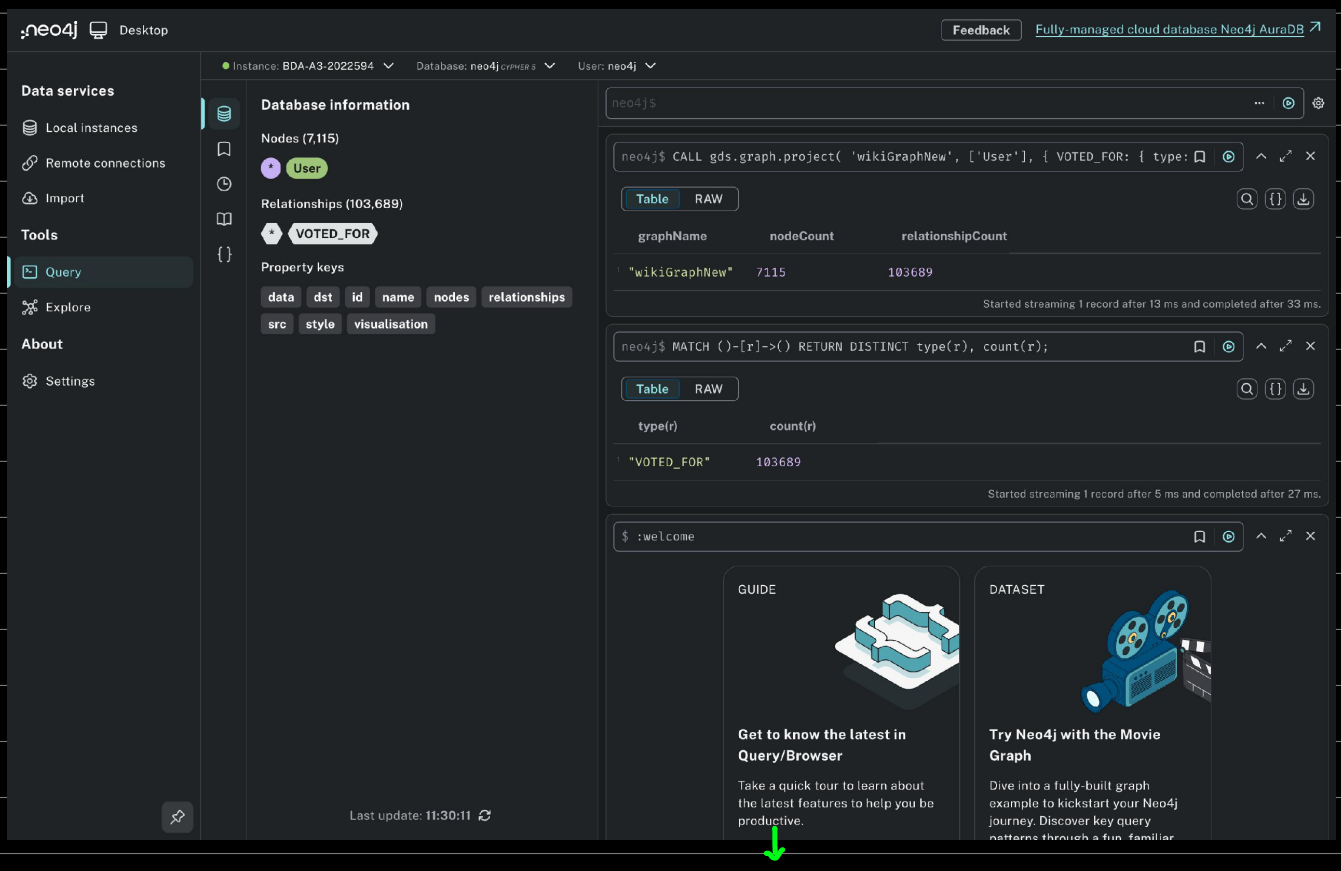
↑ convert to csv files, one for edges other for nodes



↓

Define relationship from user to user (user is our nodes files) that has a from, to relationship.

also set primary keys, as src to dst.



project grap properties and connect session...

```
// Drop old graph if it exists
CALL gds.graph.drop('wikiGraphDirected', false)
YIELD graphName
RETURN
  graphName AS graph,
  0 AS nodes,
  0 AS edges,
  0 AS total

UNION ALL

// Project and print node + edge counts
CALL gds.graph.project(
  'wikiGraphDirected',
  'User',
  { VOTED_FOR: { orientation: 'NATURAL' } }
)
YIELD graphName, nodeCount, relationshipCount
RETURN
  graphName AS graph,
  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 yeild /return node count, edges and total relationships.

```
YIELD graphName
RETURN
  graphName AS graph,
  0 AS nodes,
  0 AS edges,
  0 AS total

UNION ALL

// Project and print node + edge counts
CALL gds.graph.project(
  'wikiGraphDirected',
  'User',
  { VOTED_FOR: { orientation: 'NATURAL' } }
)
YIELD graphName, nodeCount, relationshipCount
RETURN
  graphName AS graph,
  nodeCount AS nodes,
  relationshipCount AS edges,
  (nodeCount + relationshipCount) AS total;
```

↓ return graphname, nodes and total edges.

```
CALL gds.wcc.stream('wikiGraphDirected')
YIELD nodeId, componentId
WITH componentId, count(*) AS size
ORDER BY size DESC LIMIT 1
RETURN size AS nodes_in_largest_WCC,
       round(toFloat(size) / 7115, 3) AS fraction_of_total;
```

Table | RAW

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 in-memory directed graph. and return nodeID and componentID of each node. Group nodes based on Component ID's, and count their size, return the largest wcc., also compute fraction of total

```
CALL gds.wcc.stream('wikiGraphDirected')
YIELD nodeId, componentId  — return nodeIDs and componentIDs!
WITH componentId, collect(nodeId) AS nodes
ORDER BY size(nodes) DESC LIMIT 1
WITH nodes
MATCH (u)-[r:VOTED_FOR]->(v)
WHERE id(u) IN nodes AND id(v) IN nodes
RETURN count(r) AS edges_in_largest_WCC,
       round(toFloat(count(r)) / 103689, 3) AS fraction_of_total;
```

Table | RAW

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.

```
CALL gds.scc.stream('wikiGraphDirected')
YIELD nodeId, componentId
WITH componentId, count(*) AS size
ORDER BY size DESC LIMIT 1
RETURN size AS nodes_in_largest_SCC,
       round(toFloat(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 of components.. return largest component and also return fraction of total.

```
CALL gds.scc.stream('wikiGraphDirected')
YIELD nodeId, componentId
WITH componentId, collect(nodeId) AS nodes
ORDER BY size(nodes) DESC LIMIT 1
WITH nodes
MATCH (u)-[r:VOTED_FOR]->(v)
WHERE id(u) IN nodes AND id(v) IN nodes
RETURN count(r) AS edges_in_largest_SCC,
       round(toFloat(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.

2 warnings

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

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

CALL gds.graph.project(
  'wikiGraphUndirected',
  'User',
  { VOTED_FOR: { orientation: 'UNDIRECTED' } }
)
YIELD graphName, nodeCount, relationshipCount
RETURN graphName, nodeCount, relationshipCount;
```

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

```
CALL gds.graph.project(
```

↓ convert to a undirected projection.

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

Table | RAW

average_clusterin
0.14089784589308788

↑ matches GT only slight error

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

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

likely FP error in calculation of average.

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

Table | RAW

number_of_triangl
608389.0

↑ matches GT value

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

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

count total Δs ↑

```
// Step 1: total number of triangles
CALL {
  CALL gds.triangleCount.stream('wikiGraphUndirected')
  YIELD nodeId, triangleCount
  RETURN sum(triangleCount) / 3.0 AS totalTriangles
}

// Step 2: total connected triples = sum(degree * (degree - 1) / 2)
CALL {
  CALL gds.degree.stream('wikiGraphUndirected')
  YIELD nodeId, score AS degree
  RETURN sum(degree * (degree - 1) / 2) AS totalTriples
}

// Step 3: compute global fraction
RETURN
  totalTriangles AS number_of_triangles,
  totalTriples AS number_of_triples,
  (totalTriangles / totalTriples) AS fraction_closed_triangles;
```

→ count total triples via formula

CALL subquery without a variable scope clause is now deprecated. Use CALL () { ... }

→ return open and closed triplets.

↗ calculate fraction of Δs.

Table | RAW

number_of_triangl	number_of_triples	fraction_closed_tr
608389.0	15890700.0	0.038285852731471864

off from GT ↑

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.

```
// Estimate diameter and 90% effective diameter more accurately
MATCH (n:User)
WITH collect(id(n)) AS allNodes
UNWIND allNodes[..200] AS sourceNode        // increase sample size to 200
CALL gds.shortestPath.dijkstra.stream('wikiGraphUndirected', {
  sourceNode: sourceNode,
  targetNodes: allNodes,
  relationshipWeightProperty: null
})
YIELD totalCost
WHERE totalCost IS NOT NULL AND totalCost > 0
RETURN
  max(totalCost) AS diameter,
  percentileCont(totalCost, 0.9) AS effective_diameter_90;
```

↑ no weights in graph.

Table | RAW

diameter	effective_diamete
6.0	4.0

↑ off from GT

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 11th GDS execution, multiple concurrent graphs and advanced procedures.