CSE 538 Graph Database and Graph Analytics

Project 3

Octavio Villalaz

13/7/2024

1)

1.1)

Script:

```
CALL apoc.load.json('file:///yelp_academic_dataset_business.json')
YIELD value
MERGE (b:Business{id:value.business_id})
SET b += apoc.map.clean(value,
['attributes','hours','business_id','categories'],[])
WITH b, split(value.categories, ',') as categories
UNWIND categories as category
MERGE (c:Category{id:trim(category)})
MERGE (b)-[:IN_CATEGORY]->(c)
```

Result:

```
Added 151657 labels, created 151657 nodes, set 1655117 properties, created 668549 relationships, completed after 140196 ms.
```

1.2)

First I ran this CIPHER query to create a csv and get the degrees of each business and their names.

```
CALL apoc.export.csv.query(
  "MATCH (b:Business)-[:IN_CATEGORY]-
>(c:Category) RETURN b.id AS business_id, COUNT(c) AS degree",
  "business_degrees.csv",
  {}
)
```

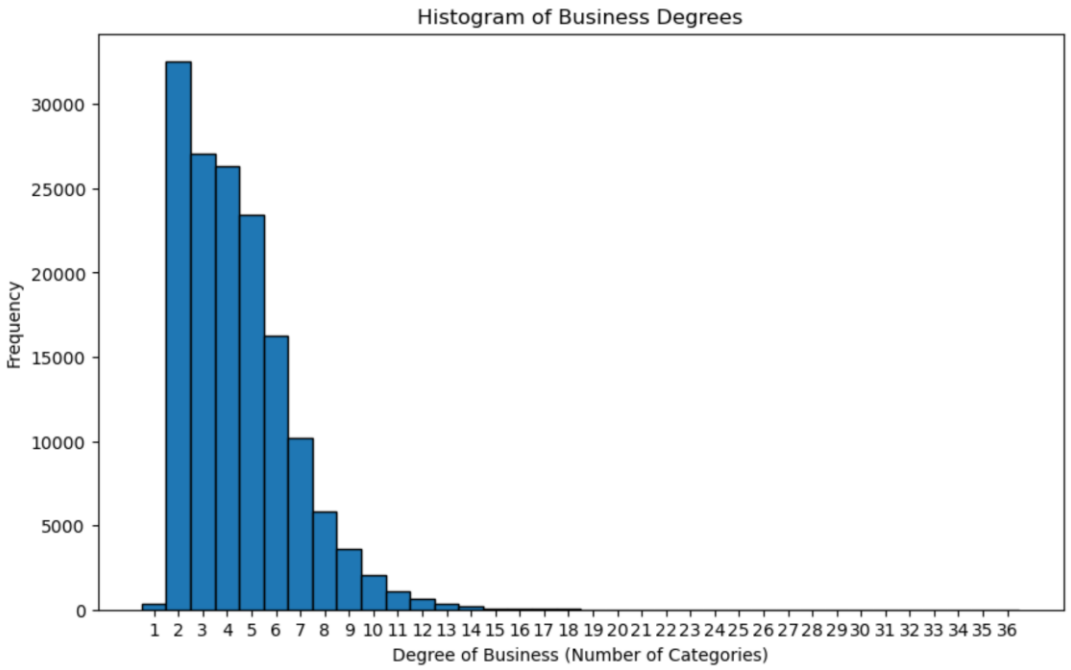
Then ran this python code in jupyter notebooks to create the histogram with the csv created.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the CSV file into a DataFrame
df = pd.read_csv('business_degrees.csv')

# Plot the histogram
plt.figure(figsize=(10, 6))
plt.hist(df['degree'], bins=range(1, df['degree'].max() + 2), edgecolor='black', align='left')
plt.xlabel('Degree of Business (Number of Categories)')
plt.ylabel('Frequency')
plt.title('Histogram of Business Degrees')
plt.xticks(range(1, df['degree'].max() + 1))
plt.show()
```

And the result of the python script was this histogram.



2)

2.1)

Number of Station nodes:

```
MATCH (s:Station)
RETURN count(s) AS numberOfStations
```

	numberOfStations
1	2593

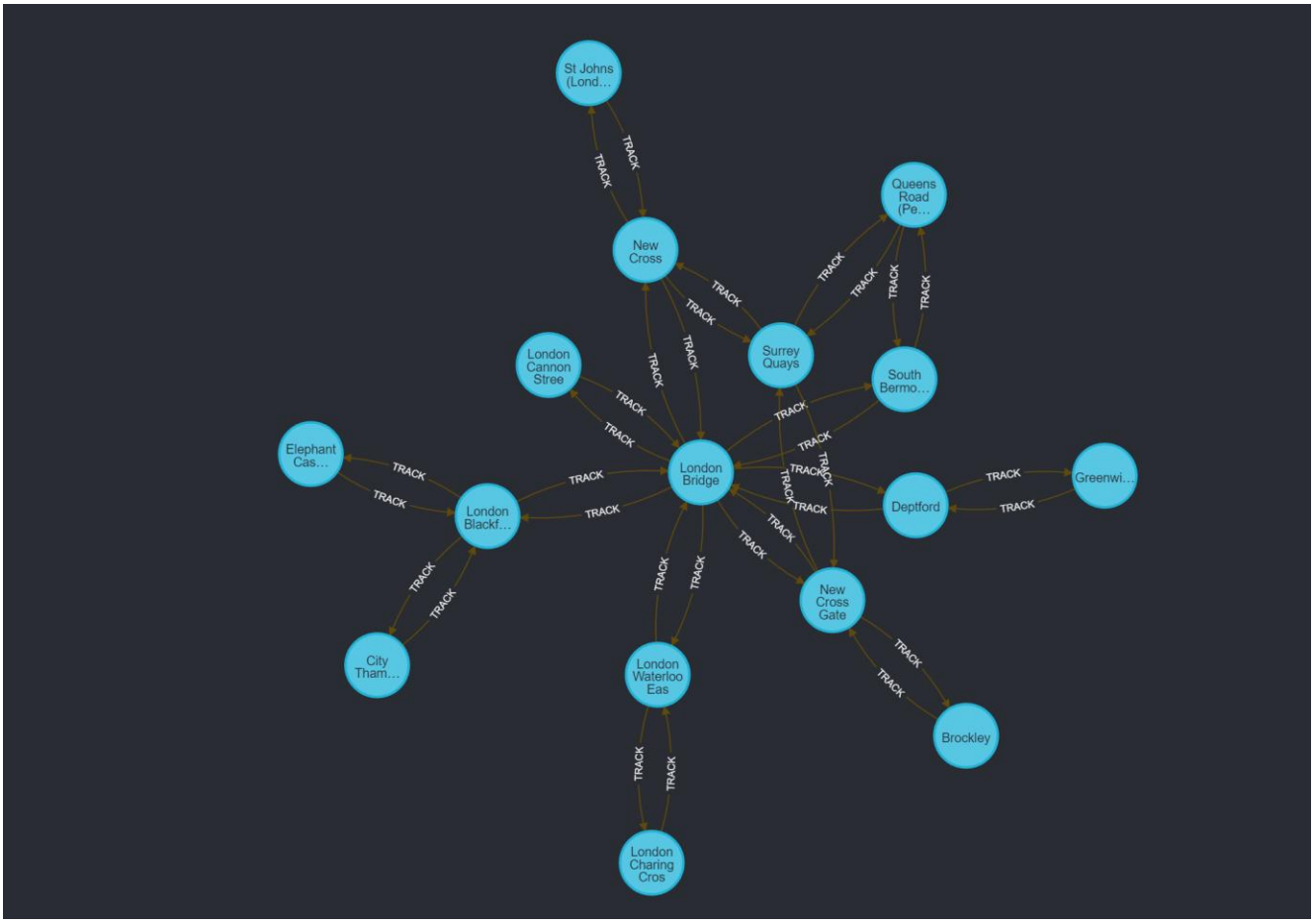
Number of Track Relationship:

```
MATCH ()-[r:TRACK]->()
RETURN count(r) AS numberOfTrackRelationships
```

	numberOfTrackRelationships
1	5782

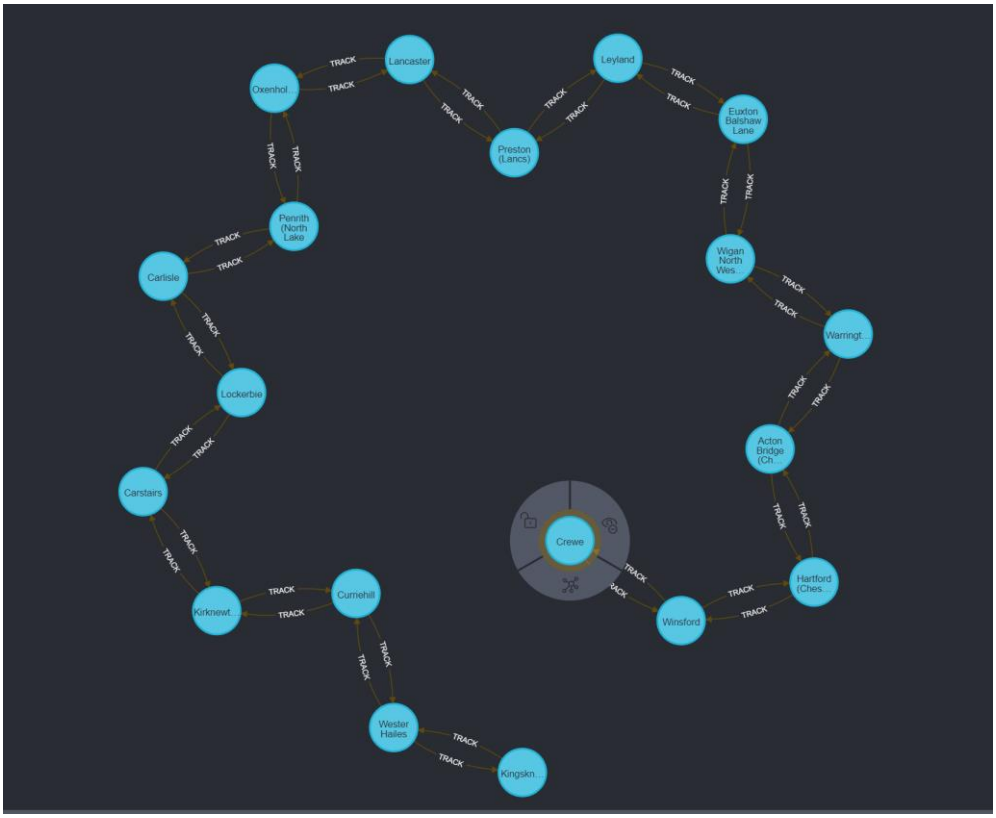
2.2)

```
MATCH path = (start:Station {crs: "LBG"})-[:TRACK*1..2]->(end:Station)
RETURN path
```



2.3)

```
MATCH (start:Station {crs: "CRE"}), (end:Station {crs: "KGE"})
MATCH path = shortestPath((start)-[:TRACK*]-(end))
RETURN path,
      reduce(totalDistance = 0, rel IN relationships(path) | totalDistance + rel.distance) AS totalDi
stance
```



CRE->WSF->HTF->ACB->WBQ->WGN->EBA->LEY->PRE->LAN->OXN->PNR->CAR->LOC->CRS->KKN->CUH->WTA->KGE

3)

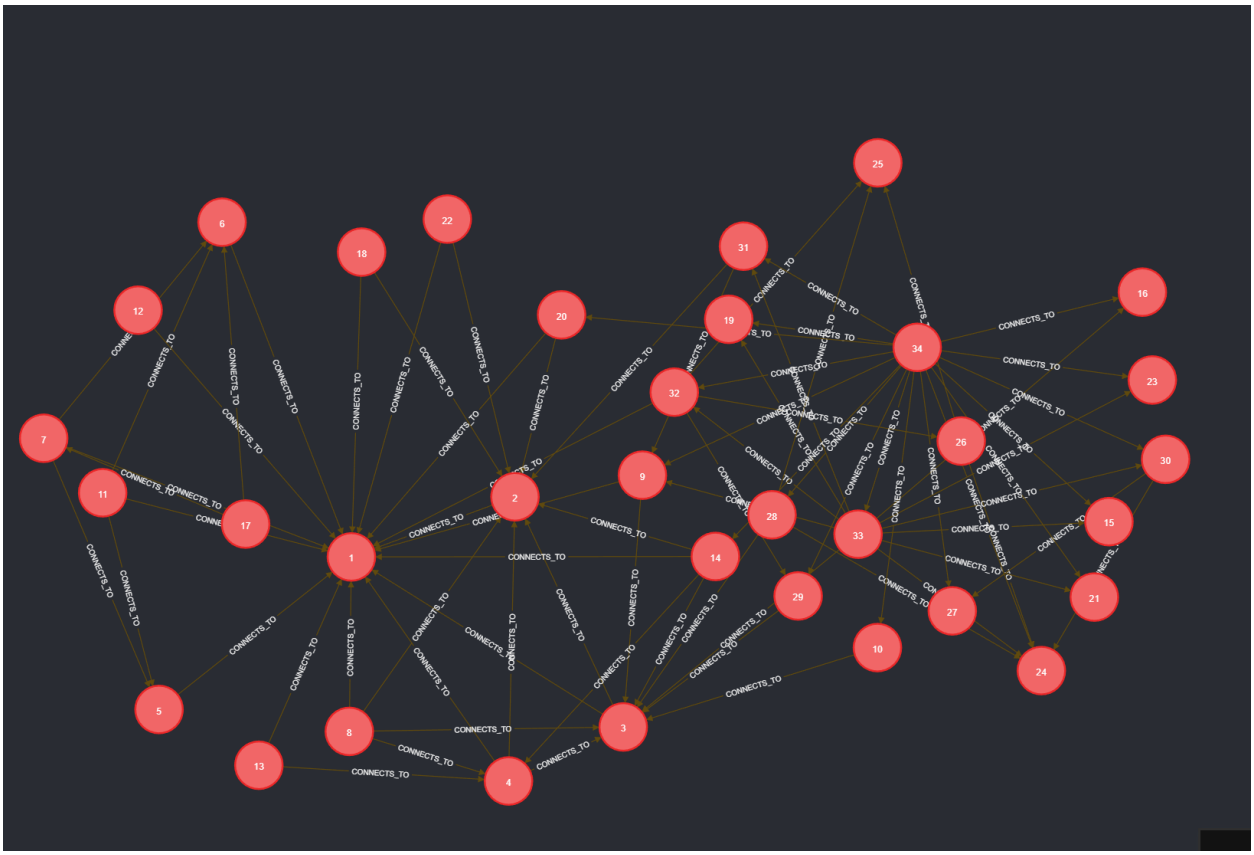
3.1)

CSV File: Attached with this document.

Script to load CSV File:

```
LOAD CSV WITH HEADERS FROM 'file:///karate.csv' AS row
WITH row, SPLIT(row.nid2, ' ') AS targets
MERGE (n1:Node {id: toInteger(row.nid1)})
FOREACH (targetId IN targets |
   MERGE (n2:Node {id: toInteger(targetId)})
   MERGE (n1)-[:CONNECTS_TO]->(n2)
)
```

Graph:



3.2)

Centrality Analysis

Before running the centrality algorithm scripts, create an in-memory graph projection in GDS:

```
CALL gds.graph.project(
  'KarateGraph',
  'Node',
  'CONNECTS_TO'
)
```

Degree Centrality:

Nodes with the highest degree score are the most connected.

Script:

```
CALL gds.degree.stream('KarateGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, score
ORDER BY score DESC
LIMIT 10;
```

Result:

	id	score
1	34	17.0
2	33	11.0
3	8	4.0
4	14	4.0
5	32	4.0
6	7	3.0

Betweenness Centrality:

Nodes with high betweenness act as key bridges in the graph.

Script:

```
CALL gds.betweenness.stream('KarateGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, score
ORDER BY score DESC
LIMIT 10;
```

Result:

	id	score
1	3	8.833333333333334
2	32	5.083333333333334
3	9	2.25
4	29	2.1666666666666665
5	4	2.0

3.3)

Each node will be assigned a communityId that represents the community to which it belongs. Nodes with the same comunityId are considered to be part of the same community.

This analysis can help you understand the structure and groupings within the network.

Script:

```
CALL gds.louvain.stream('KarateGraph')
YIELD nodeId, communityId
RETURN gds.util.asNode(nodeId).id as id, communityId
ORDER BY communityId, id
```

Result:

	id	communityId
1	1	25
2	2	25
3	3	25
4	4	25
5	5	25
6	6	25