

UE20CS323

Graph Theory and Applications

Synthetic Music Dataset Report *Graph Database Project*

SRN : PES1UG20CS331

Name : Renita Kurian

Section : F

SRN : PES1UG20CS410

Name : Shreyash Chatterjee

Section : G

Introduction to Dataset

The dataset was synthetically generated using a python script. The generated dataset consists of 7 columns – id, label, name, _start, _end, _type, rating

The data consists of songs from the 1950s, the different genres that they belong to, and around 300 people. There are two relationships between the labels user (people listening to music), genre and track (or song). Users listen to tracks and every track belongs to a genre. There are 7 genres, 300 users and approximately 1000 tracks. There are nearly 5500 relationships in this dataset.

id	label	name	_start	_end	_type	rating
0	:genre	pop				
1	:genre	jazz				
2	:genre	rock				
3	:genre	reggae				
4	:genre	country				
5	:genre	hip hop				
6	:genre	blues				
7	:track	the best things in life are free				
8	:track	you're the best thing yet				
9	:track	twin skeleton's (hotel in nyc)				
10	:track	wicked rebel				
11	:track	dirt				
12	:track	oh lord, search my heart				
13	:track	lady grinning soul				
14	:track	this afternoon				
15	:track	story of my life				
16	:track	just another honky				
17	:track	morning light				

Neo4j Commands

Inserting genre nodes

load csv from 'file:///graph_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6]) as rating

where labels=":genre"

create(:genre{id:id, name:name})

The image displays the Neo4j Desktop interface. The top panel shows a command executed in the Neo4j CLI: `neo4j$ load csv from 'file:///graph_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6]) as rating`. Below the command, a status message indicates: "Added 7 labels, created 7 nodes, set 14 properties, completed after 52 ms." The bottom panel shows the graph visualization. The command entered is `neo4j$ match(n) return n`. The graph displays 7 nodes, each labeled with a music genre: reggae, jazz, rock, blues, hip hop, pop, and country. The 'country' node is highlighted with a blue border. On the right, the 'Overview' panel shows 'Node labels' with a count of 7 for the 'genre' label. Below this, it states 'Displaying 7 nodes, 0 relationships.'

neo4j\$ load csv from 'file:///graph_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6]) as rating

Added 7 labels, created 7 nodes, set 14 properties, completed after 52 ms.

neo4j\$ match(n) return n

Overview

Node labels

7 (7) genre (7)

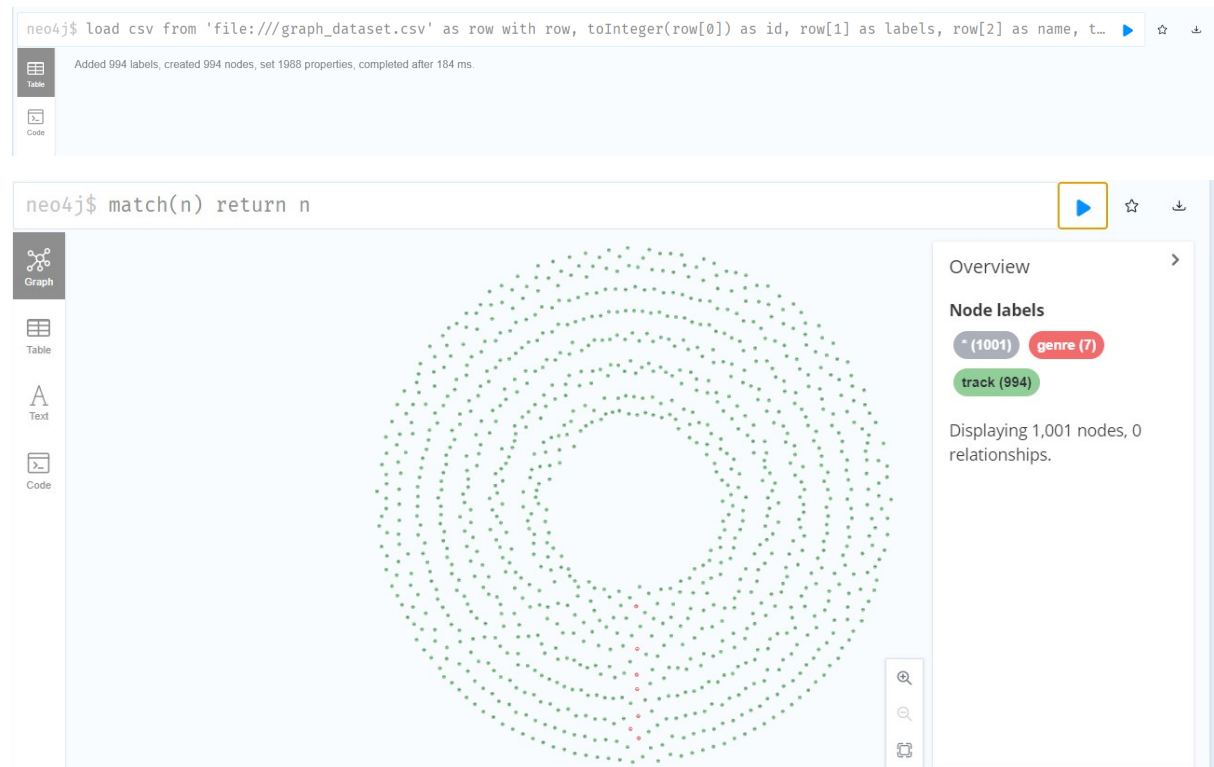
Displaying 7 nodes, 0 relationships.

Inserting track nodes

```
load csv from 'file:///graph_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels,
row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6])
as rating
```

```
where labels=":track"
```

```
create(:track{id:id, name:name})
```

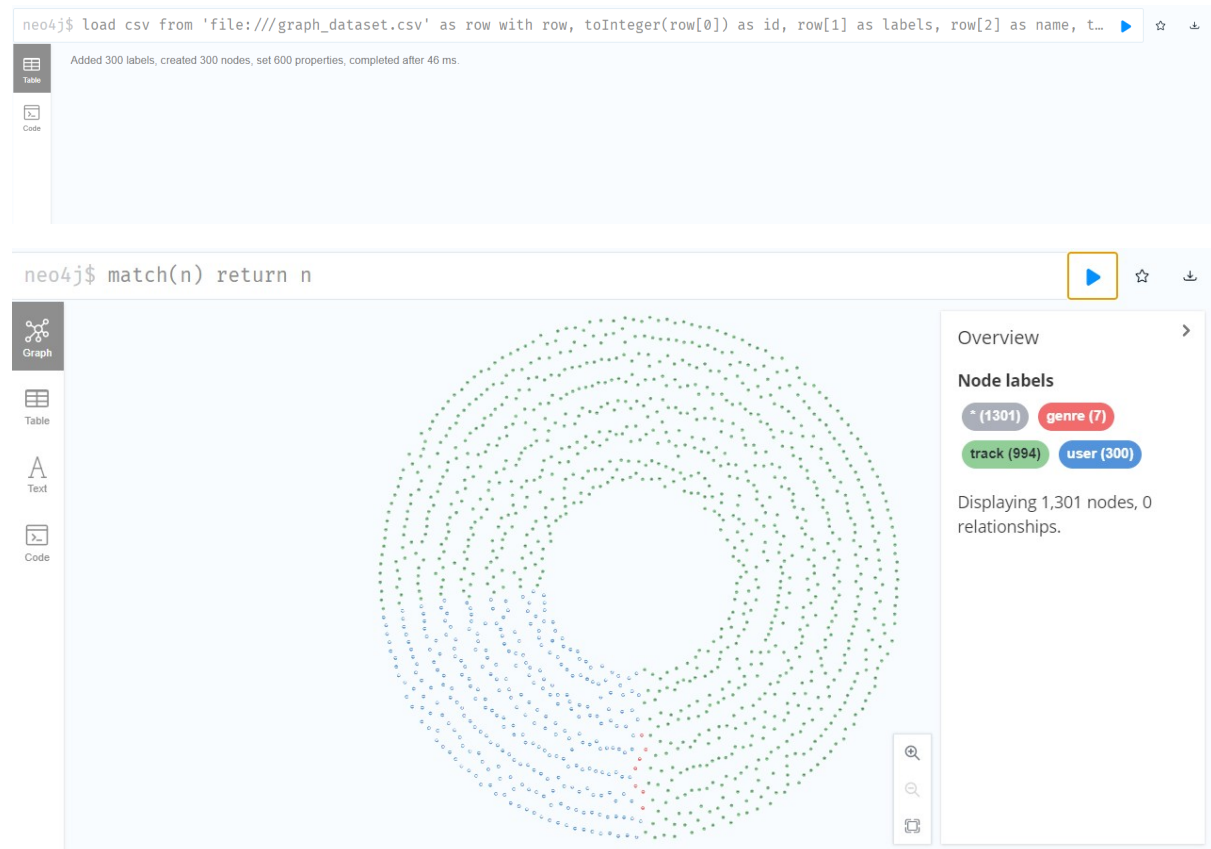


Inserting user nodes

```
load csv from 'file:///graph_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels,
row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6])
as rating
```

```
where labels=":user"
```

```
create(:user{id:id, name:name})
```



Adding Track - Genre relationship [BELONGS_TO]

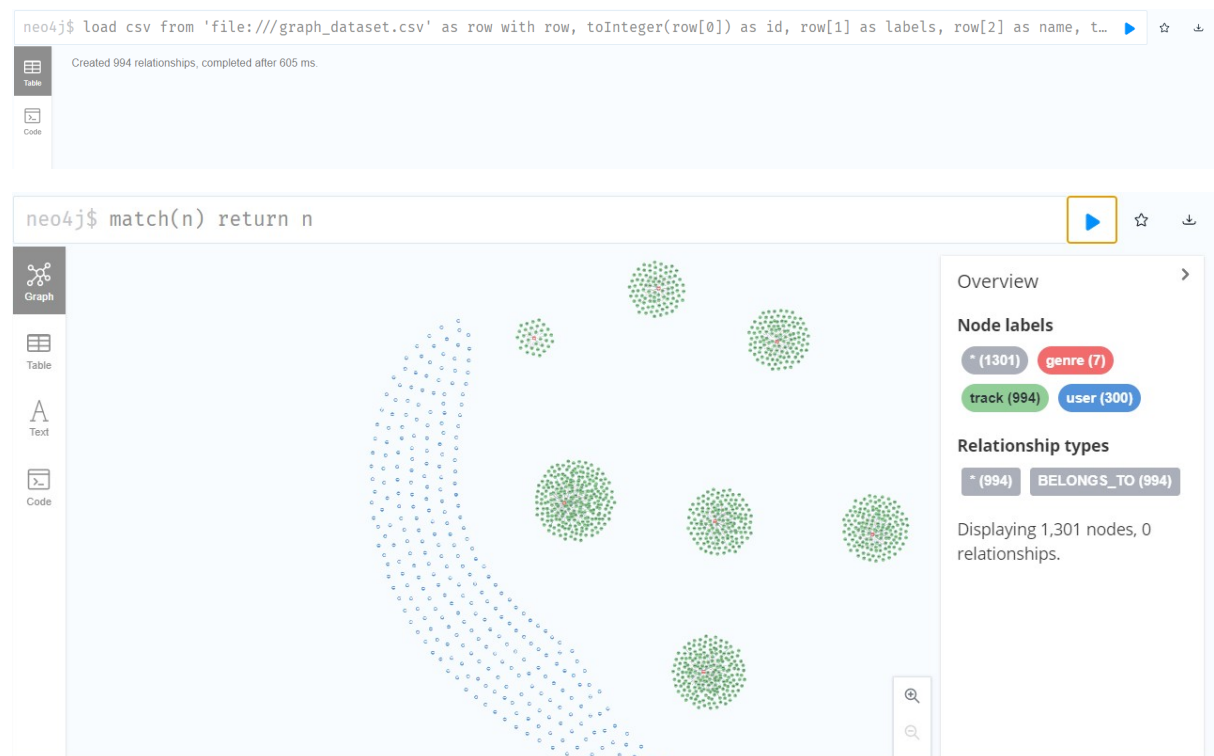
load csv from 'file:///graph_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6]) as rating

where type = "BELONGS_TO"

match(t:track) where t.id=start

match(g:genre) where g.id=end

create (t)-[:BELONGS_TO]->(g)



Adding User - Track relationship [LISTENS_TO]

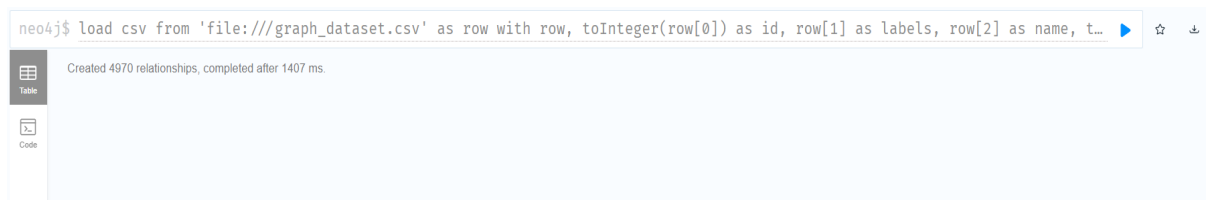
```
load csv from 'file:///graph_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6]) as rating
```

```
where type = "LISTENS_TO"
```

```
match(u:user) where u.id=start
```

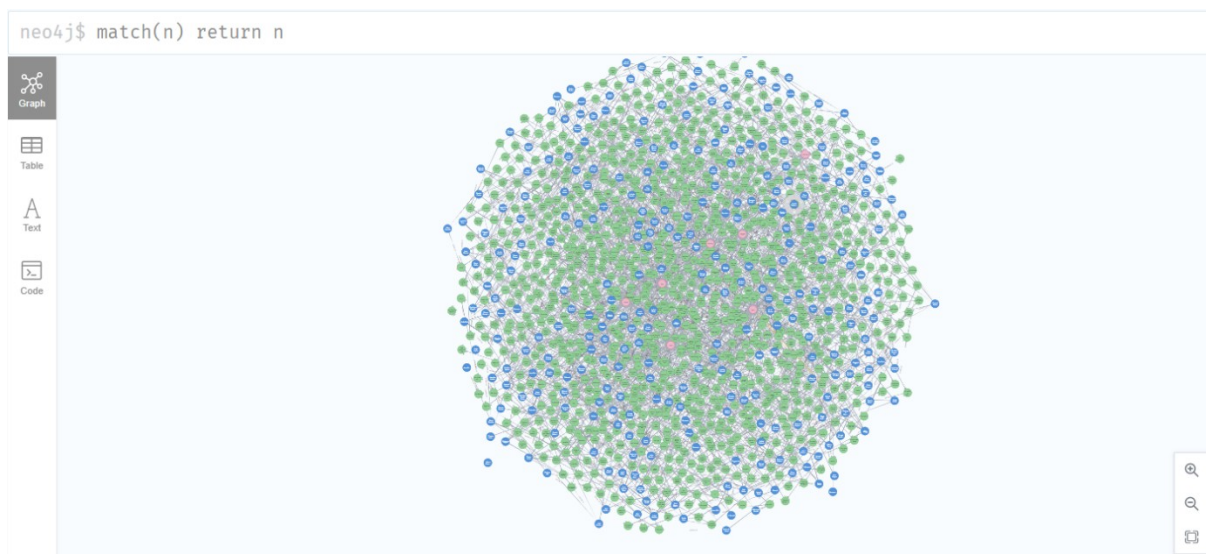
```
match(t:track) where t.id=end
```

```
create (u)-[:LISTENS_TO]->(t)
```



View final graph

Match(n) return n



Content Filtering

- Recommend for user – Harry Mulisch

```
match(u:user{name:"Harry Mulisch"})-[:LISTENS_TO]->
```

```
(t:track)-[:BELONGS_TO]->(g:genre)<-[:BELONGS_TO]-(z:track)
```

```
where not exists((u)-[:LISTENS_TO]->(z)) with t, z, count(g) as intersection
```

```
match (t)-[:BELONGS_TO]->(sc:genre)
```

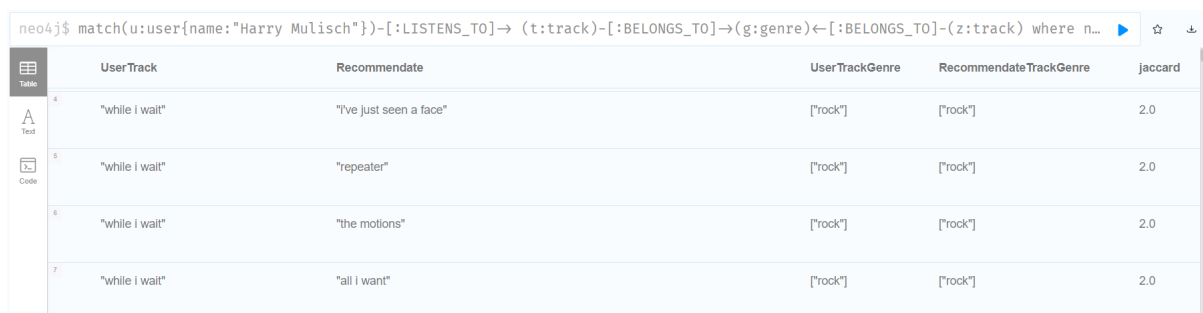
```
with t, z, intersection, COLLECT(sc.name) as s1
```

```
match (z)-[:BELONGS_TO]->(zc:genre)
```

```
with t, z, s1, intersection, COLLECT(zc.name) as s2
```

```
with t, z, intersection, s1 + [x in s2 where not x in s1] as union, s1, s2
```

```
return t.name as UserTrack, z.name as Recommendate, s1 as UserTrackGenre, s2 as  
RecommendateTrackGenre, ((1.0*intersection)/SIZE(union)) as jaccard order by jaccard desc
```



	UserTrack	Recommendate	UserTrackGenre	RecommendateTrackGenre	Jaccard
4	"while i wait"	"I've just seen a face"	["rock"]	["rock"]	2.0
5	"while i wait"	"repeater"	["rock"]	["rock"]	2.0
6	"while i wait"	"the motions"	["rock"]	["rock"]	2.0
7	"while i wait"	"all i want"	["rock"]	["rock"]	2.0

Collaborative Filtering

- Recommend for user – Harry Mulisch

```
match(t:track)
```

```
with collect(t.name) as tracks
```

```
match(u:user{name:"Harry Mulisch"})-[:LISTENS_TO]->(z:track)
```

```
with tracks, collect(z.name) as mytrack, u
```

```
with tracks, mytrack, gds.alpha.ml.oneHotEncoding(tracks, mytrack) as t1, u
```

```
match(o:user)-[:LISTENS_TO]->(x:track)
```

```
where u<>o
```

```
with tracks, collect(x.name) as otherusertrack, mytrack, t1, o, u
```

```
with t1, gds.alpha.ml.oneHotEncoding(tracks, otherusertrack) as t2, o, u
```

```
with o, gds.similarity.cosine(t1, t2) as simindex, u
```

```
order by simindex desc limit 1
```

```
with o, u
```

```
match(o:user)-[:LISTENS_TO]->(t:track)
```

```
where not exists((u)-[:LISTENS_TO]->(t))
```

```
return t.name as Recommendations
```



The screenshot shows a Neo4j Cypher query interface. The query is: `neo4j$ match(t:track) with collect(t.name) as tracks match(u:user{name:"Harry Mulisch"})-[:LISTENS_TO]->(z:track) with tracks, collect(z.name) as mytrack, u with tracks, mytrack, gds.alpha.ml.oneHotEncoding(tracks, mytrack) as t1, u match(o:user)-[:LISTENS_TO]->(x:track) where u<>o with tracks, collect(x.name) as otherusertrack, mytrack, t1, o, u with t1, gds.alpha.ml.oneHotEncoding(tracks, otherusertrack) as t2, o, u with o, gds.similarity.cosine(t1, t2) as simindex, u order by simindex desc limit 1 with o, u match(o:user)-[:LISTENS_TO]->(t:track) where not exists((u)-[:LISTENS_TO]->(t)) return t.name as Recommendations`. The results are displayed in a table titled "Recommended" with 7 rows of track names.

	Recommended
1	"waiting for a girl like you"
2	"hip hop circus"
3	"confess it to your heart"
4	"bill bailey, won't you please come home?"
5	"i'm not calling you a liar"
6	"the bird and the worm"
7	

Started streaming 15 records in less than 1 ms and completed after 90 ms.

Centrality Measures

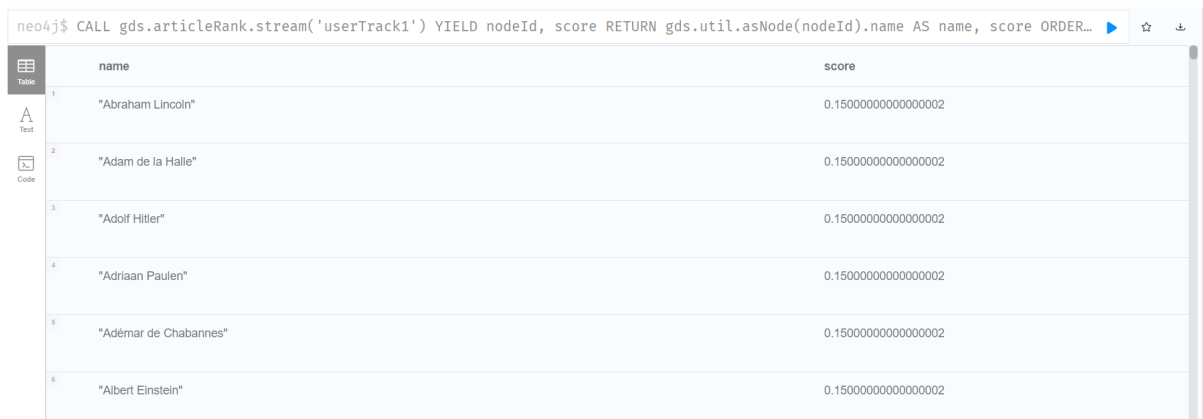
Article Rank

```
CALL gds.articleRank.stream('userTrack1')
```

```
YIELD nodeId, score
```

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

```
ORDER BY score DESC, name ASC
```



	name	score
1	"Abraham Lincoln"	0.15000000000000002
2	"Adam de la Halle"	0.15000000000000002
3	"Adolf Hitler"	0.15000000000000002
4	"Adriaan Paulen"	0.15000000000000002
5	"Adémar de Chabannes"	0.15000000000000002
6	"Albert Einstein"	0.15000000000000002

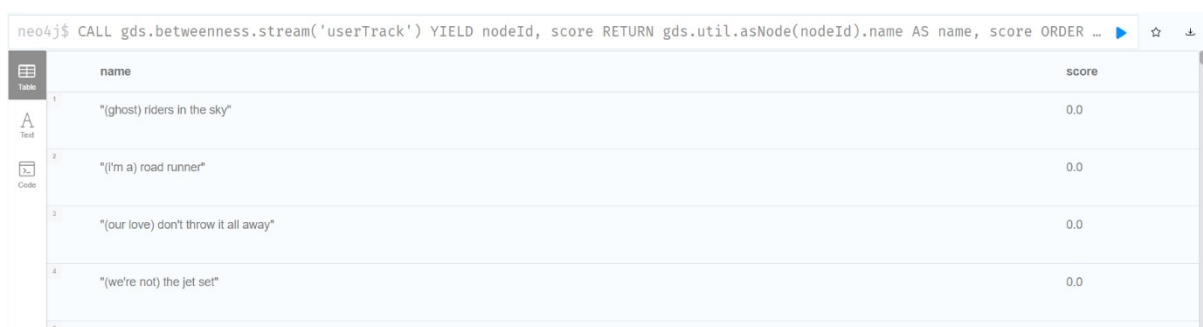
Betweenness

```
CALL gds.betweenness.stream('userTrack')
```

```
YIELD nodeId, score
```

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

```
ORDER BY name ASC
```



	name	score
1	"(ghost) riders in the sky"	0.0
2	"(i'm a) road runner"	0.0
3	"(our love) don't throw it all away"	0.0
4	"(we're not) the jet set"	0.0

Since there is no homogenous relation (track-track or user-user) betweenness measure is not useful for this dataset.

Analysis and Inferences

```
neo4j$ MATCH (x:genre)←[e:BELONGS_TO]-(g:track)←[r:LISTENS_TO]-(t:user) RETURN x.name as Genre,count(e) as Users order by Users desc
```

Genre	Users
"pop"	1159
"country"	864
"jazz"	728
"blues"	674
"rock"	549
"reggae"	474
"hip hop"	144

The above picture shows the popularity of each genre. 1159 users listen to pop music, 864 listen to country as so on. The most popular genres are pop, country and jazz. The least listened to genre is hip hop.

```
neo4j$ MATCH (g:genre)←[r:BELONGS_TO]-(t:track) RETURN g.name as Genre, COUNT(r) as No_of_Songs order by No_of_Songs desc
```

Genre	No_of_Songs
"pop"	254
"country"	195
"blues"	151
"rock"	134
"jazz"	130
"reggae"	92
"hip hop"	38

The above picture shows the distribution of songs across genre. We can see that pop has highest number of songs followed by country and blues. Hip hop is seen to have least no of songs.

Conclusion

In this project neo4j has been used to recommend music to users using collaborative and content filtering methods. In content filtering, a user is recommended music based on the genre of songs he listens to. Collaborative filtering on the other hand relies on other users, so users with similar taste will be recommended similar music. It was seen that Harry Mulisch was recommended rock songs like 'All I want', 'repeater' etc by content filtering because he listened to a rock song ('While I Wait'). Whereas in collaborative filtering the suggestions were not limited to any one genre and Harry was recommended a variety of songs based on other users with similar taste.

Various inferences can also be made from this dataset. For example, we were able to find the most popular genre (pop) and the distribution of songs for each genre.