### UE20CS323

## Graph Theory and Applications

## Synthetic Music Dataset Report

#### Graph Database Project

SRN : PES1UG20CS331

Name : Renita Kurian

Section : F

SRN : PES1UG20CS410

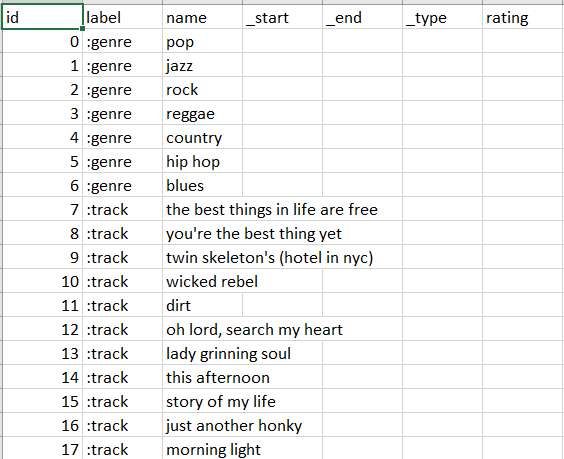
Name : Shreyash Chatterjee

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.



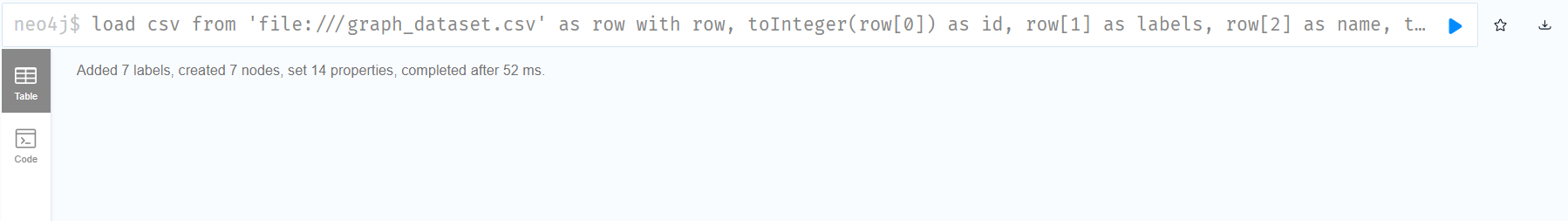
## **Neo4j Commands**

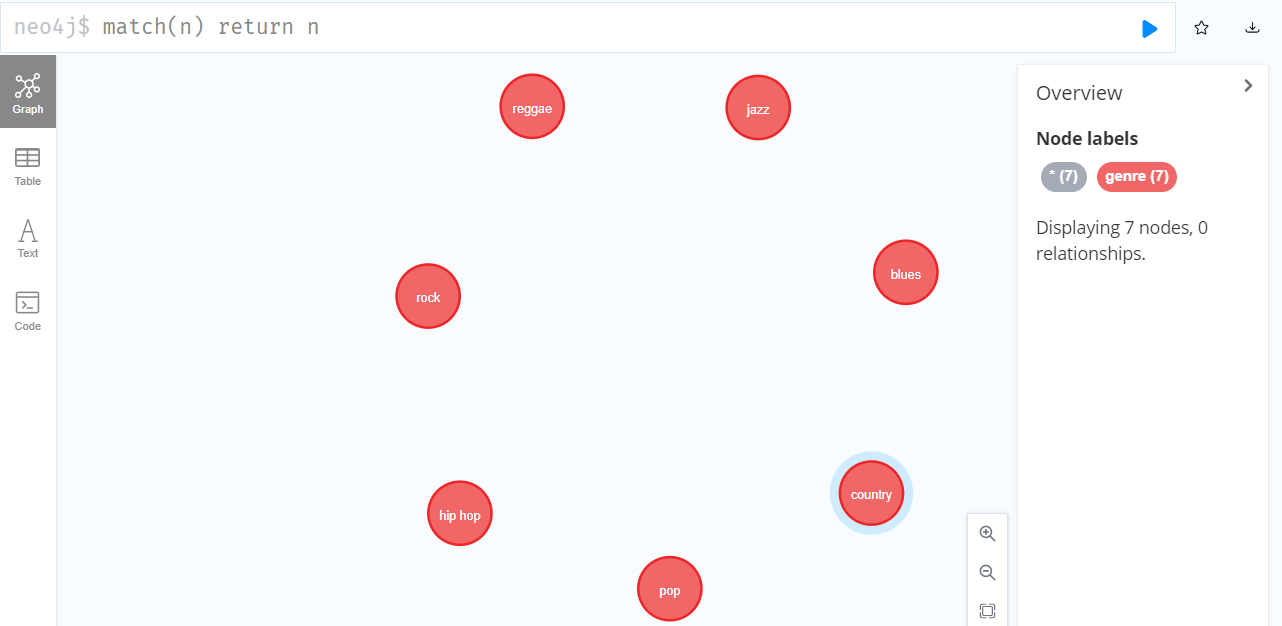
**Inserting genre nodes**

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})



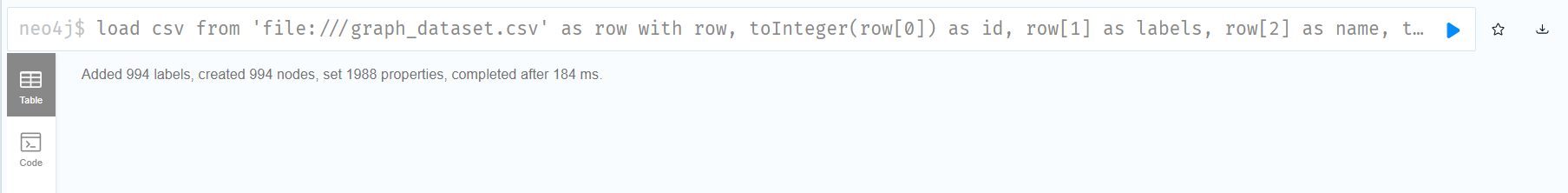


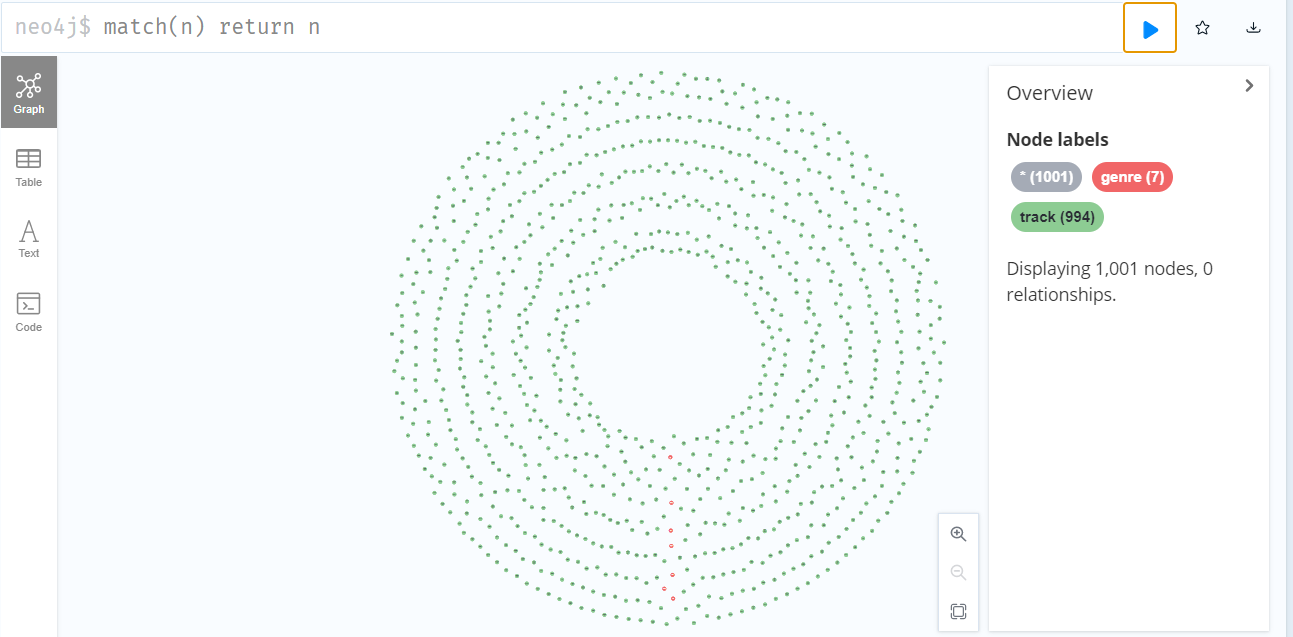
**Inserting track nodes**

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})



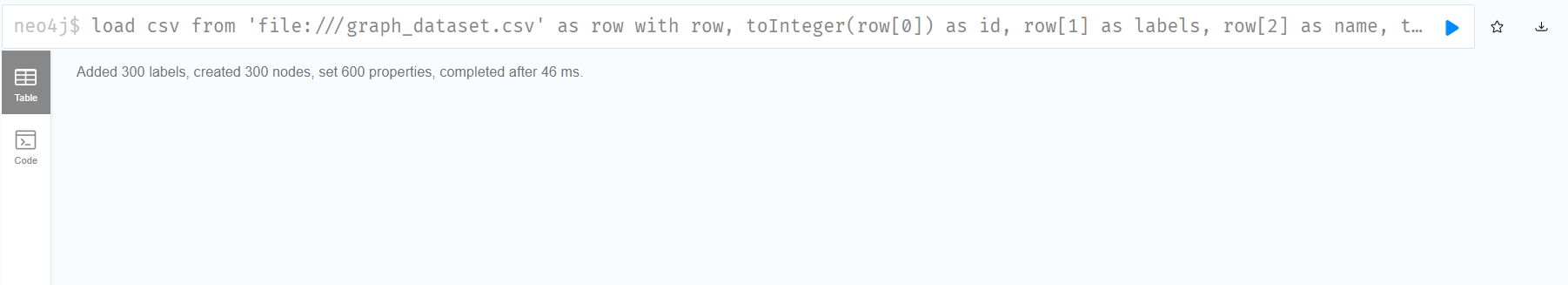


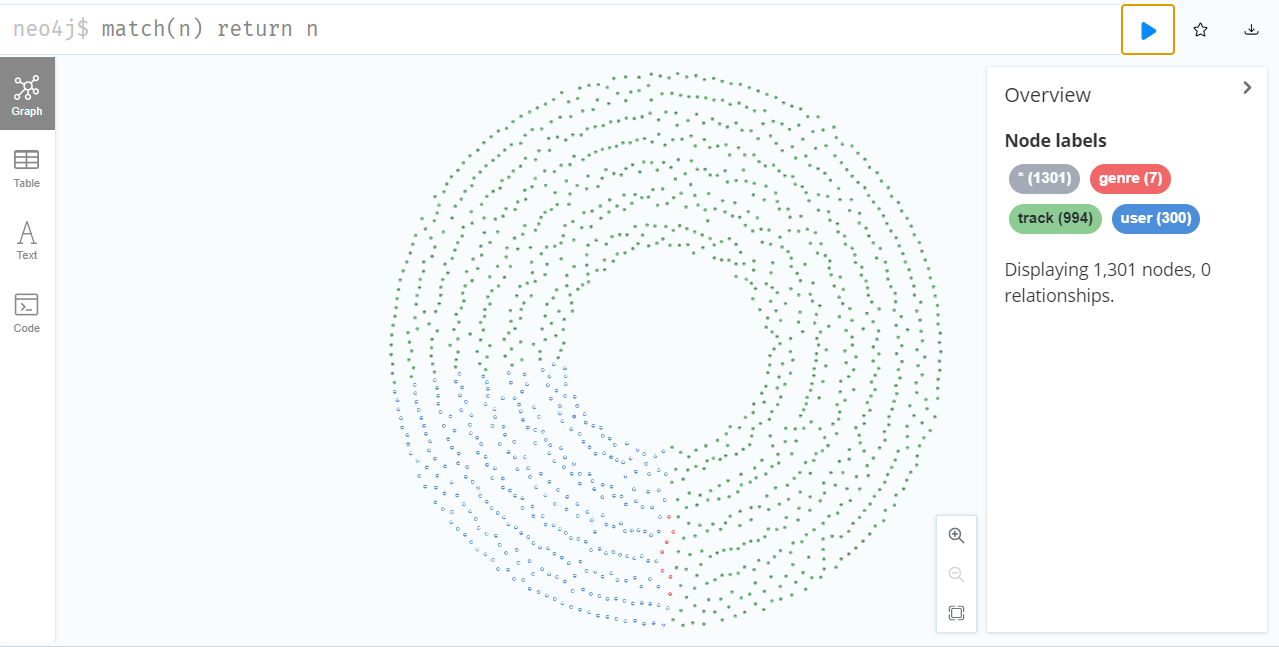
**Inserting user nodes**

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]**

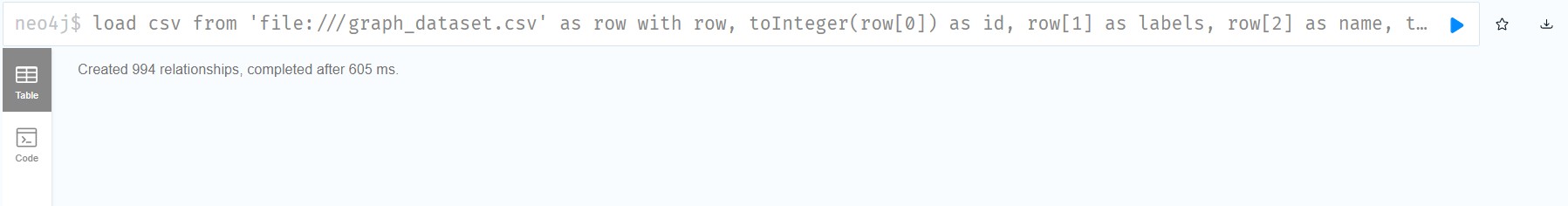
load csv from 'file:///graph\_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6]) as rating

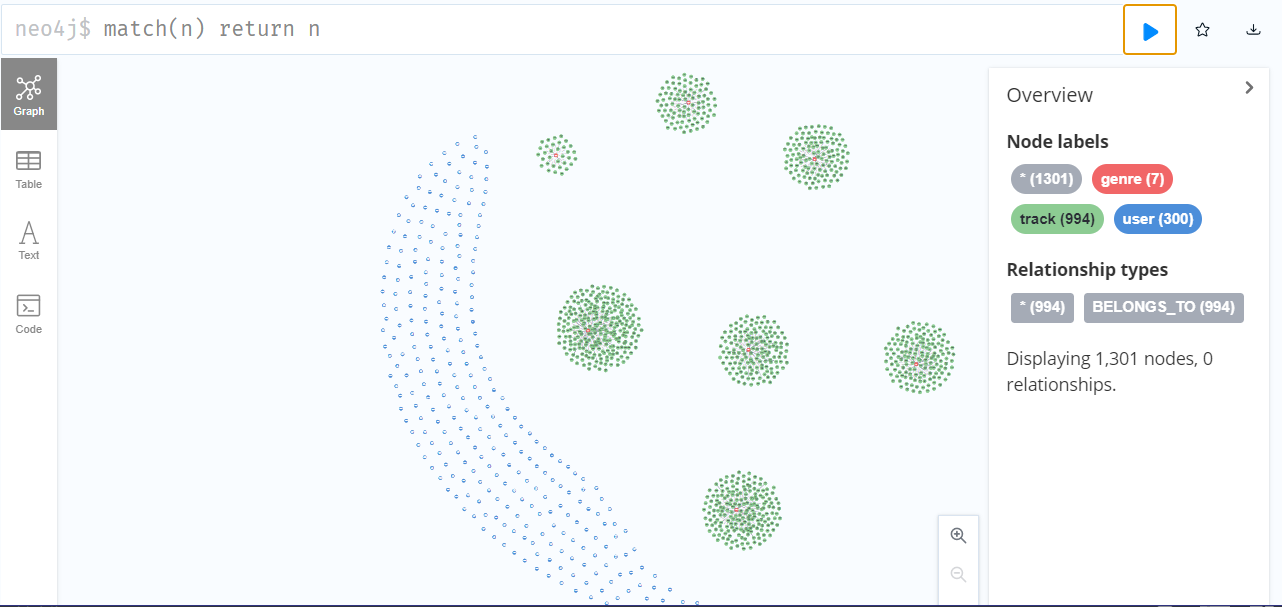
where type = "BELONGS\_TO"

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]**

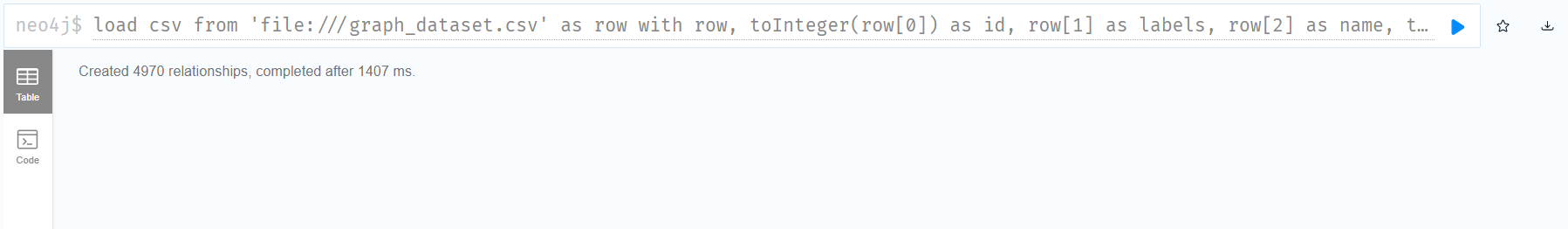
load csv from 'file:///graph\_dataset.csv' as row with row, toInteger(row[0]) as id, row[1] as labels, row[2] as name, toInteger(row[3]) as start, toInteger(row[4]) as end, row[5] as type, toFloat(row[6]) as rating

where type = "LISTENS\_TO"

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

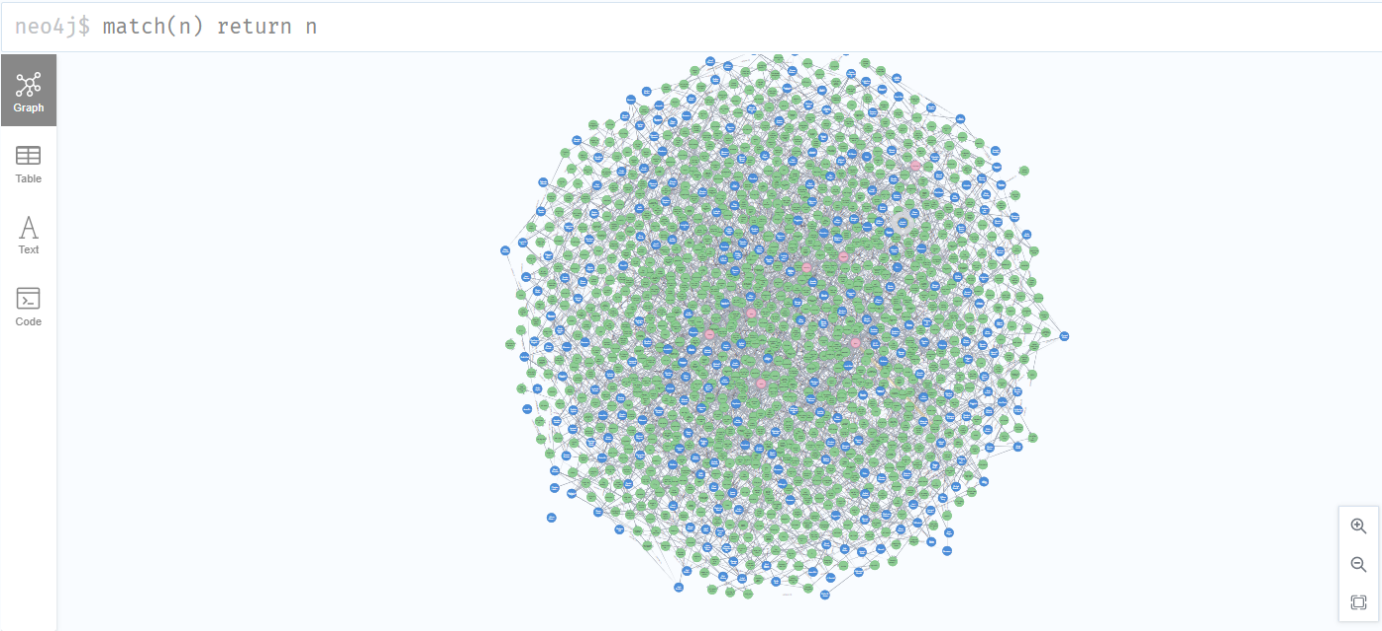
match(t:track) where t.id=end

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)

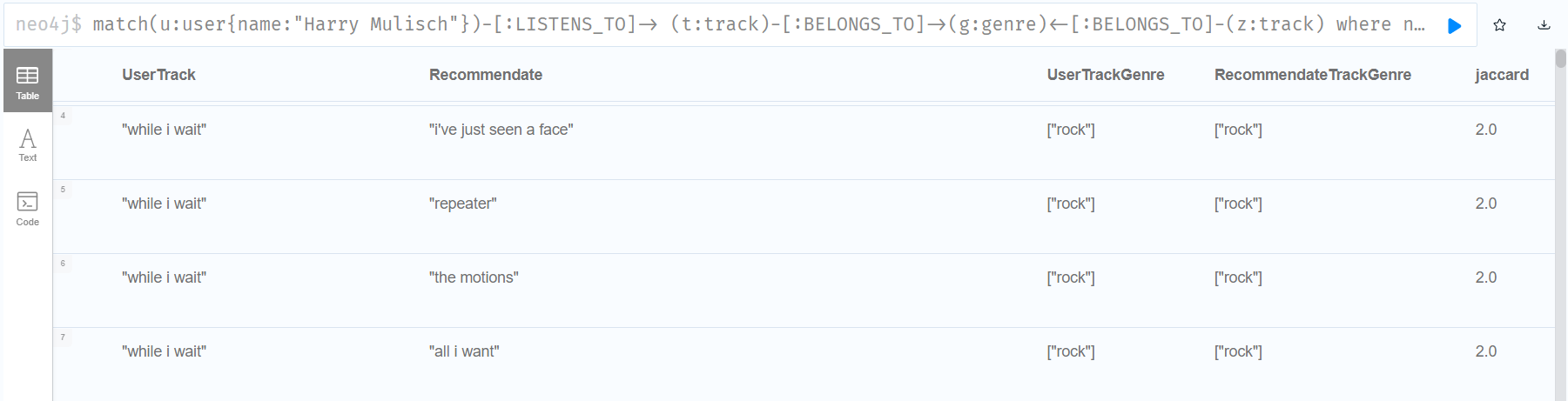
with t, z, intersection, COLLECT(sc.name) as s1

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



**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



**Centrality Measures**

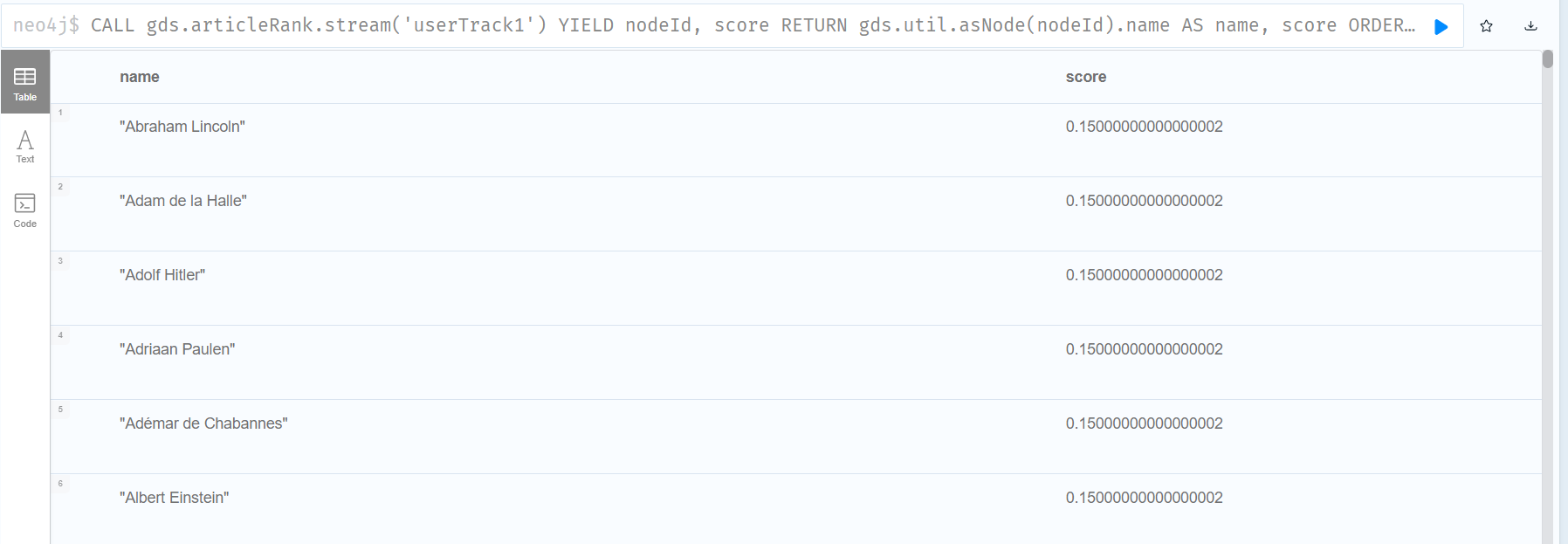
**Article Rank**

CALL gds.articleRank.stream('userTrack1')

YIELD nodeId, score

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

ORDER BY score DESC, name ASC

****

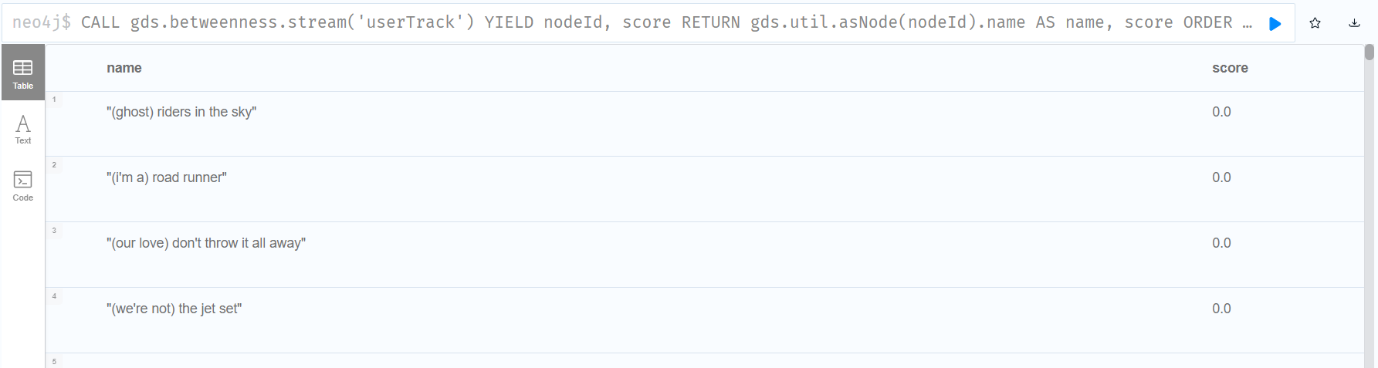
**Betweeness**

CALL gds.betweenness.stream('userTrack')

YIELD nodeId, score

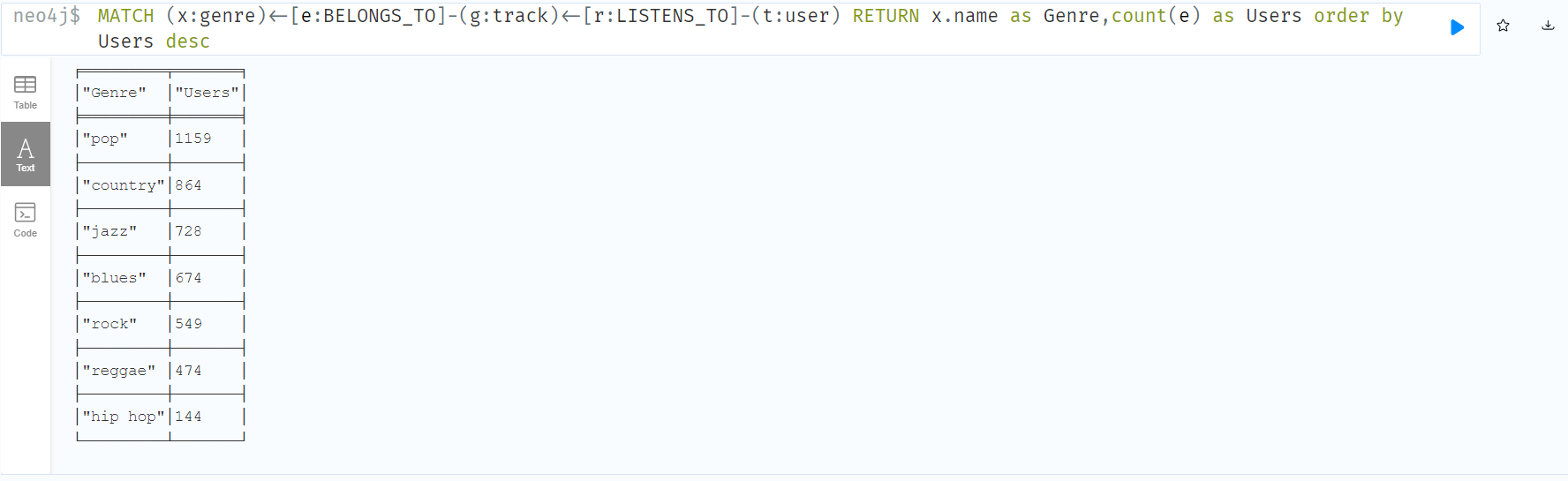
RETURN gds.util.asNode(nodeId).name AS name, score

ORDER BY name ASC

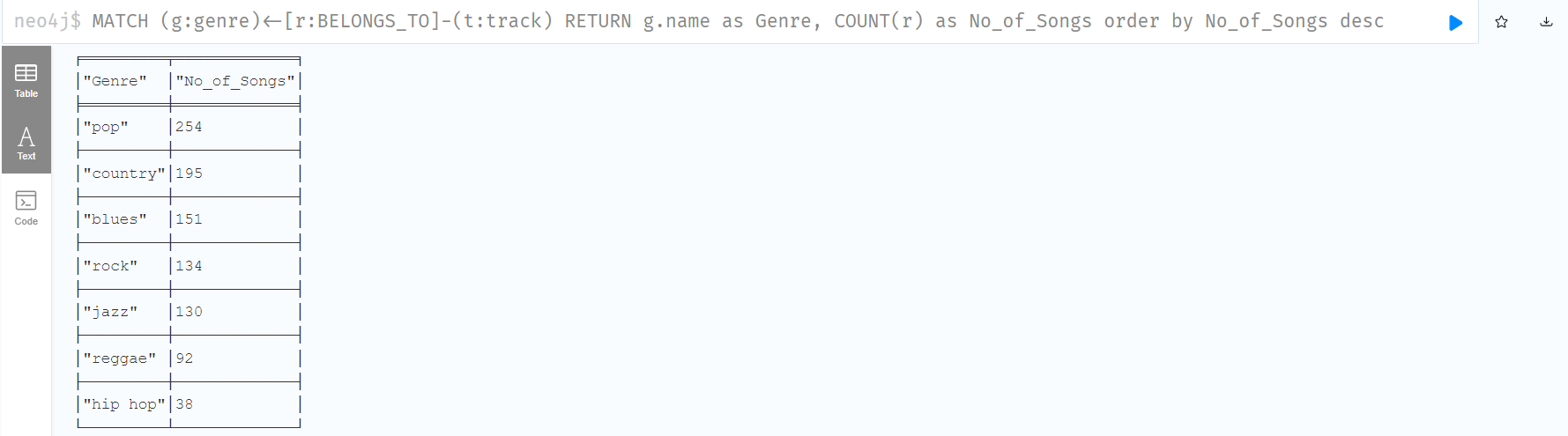


Since there is no homogenous relation (track-track or user-user) betweenness measure is not useful for this dataset.

## Analysis and Inferences



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.



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.