GROUP-4	
Arpana Singh	**Harshitha Boinepally**
Srilakshmi Narayana Murthy	**Vidushi Verma**
Vishnu Pendyala	

Meaningful goal

The primary goal of this project is to perform meaningful analytics on the Facebook ego-network data using graph NoSQL databases. Social network datasets, such as ego-Facebook, contain a rich structure of user connections and metadata that are ideal for multi-model analysis. Each user not only has a set of features that describe their interests, behaviors but also belongs to social circles that represent community structures.

In practical scenarios like friend recommendation systems, targeted advertising, or analysis of influence spread, the graph-based models can yield valuable insights. Conventional relational databases frequently struggle to efficiently manage complex and interlinked data architectures, which makes this project an ideal opportunity to investigate the advantages of NoSQL databases. Employing Cypher queries in Neo4j, we investigate which users possess comparable feature sets, how users are spread across social networks, what the predominant features are within a particular group, community intersections, and network motifs that indicate clustering tendencies.

This initiative aims not just to showcase technical skills but also to create a learning environment that:
1.Contrasts NoSQL models with conventional relational database management systems.
2.Presents graph theory ideas via Neo4j.

By achieving these objectives, we demonstrate a comprehensive data engineering and analytics pipeline that transitions from raw data to insights, constructed entirely with open-source technologies and contemporary best practices.

Workflow

The complete workflow for this project followed a well-defined sequence of stages that reflect a realistic data engineering and analytics pipeline:

1.Data Gathering and Acquisition

We obtained the ego-Facebook dataset from the Stanford SNAP archive.
The collection contains several files:

edges: A collection of edges (friend connections) linking users to their friends.
circles: Groups (or communities) that each friend is a part of.A user can have multiple circles depending on the friend lists he has .one friend list constitute of one group of friend the user has.
egofeat: Features of the primary or "ego user".This file has '0'/'1' data depending on if a feature (given in '**featnames')**is present for the ego user or not.

feat: List of all the feature that denote the characteristics of friends.This file has '0'/'1' data depending on if a feature (given in '**featnames')**is present or not for a given friend.
featnames: Descriptions of features. For one user , featnames comprises all the features a user and its whole friend group has.

2. **Data Cleaning and Preparation**

- To ready the data for import:

  -**Transformed all files into CSV format as required.**
  1.feat and egofeat files were combined and further headers were added to the dataset file.
  2.circles files converted into a structured CSV format compatible with Neo4j.
  Fixed issues like "missing userid" to avoid errors during database ingestion.

  -**Python was widely utilized during this phase to convert and verify CSVs, including:**
  Insert_column_headers_to_csv for adding header names.
  Reformatting featnames and feat file to make it readable to Neo4j.
  Handling any issues ,like null or non ASCII characters.

3.Data Modeling

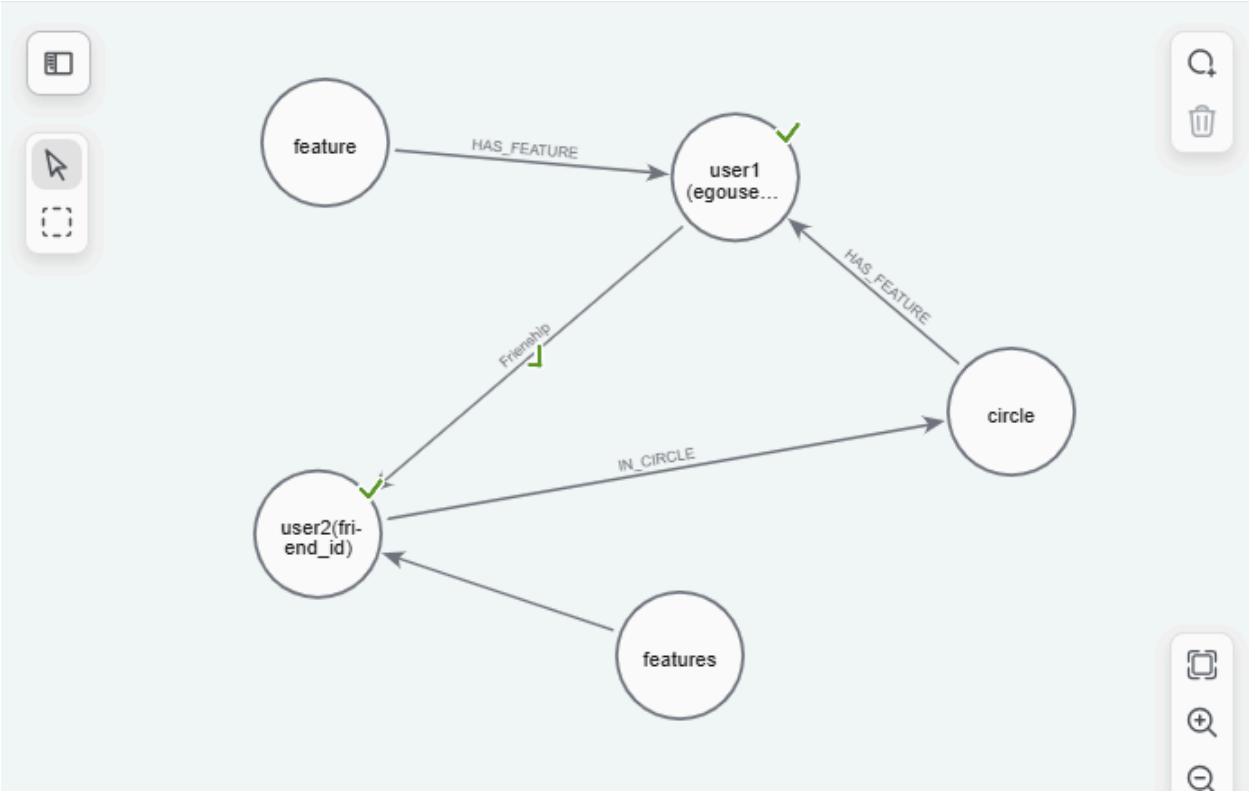
The data was modeled as

- Nodes: User1, User2, Feature, Circle
- Relationships:
  - (:User)-[:**HAS_FEATURE**]->(:Feature) : Each ego user (primary user) and its friend has features defined in egofeat & feat file , the same has been used to establish this relationship.

- (:User)-[:**IN_CIRCLE**]->(:Circle) : Each ego user's have been assigned 'circle' from circle file depending on number of friend list the user has.
- (:User1)-[:**HAS_FRIENDS**]->(:User2) : Each user is connected to another user in the network by a relationship defined as 'Friends'.

· Properties: Each node and relationship was assigned relevant metadata (IDs, names, etc.).

By data definition , one user has been identified as "ego user" and friends of ego user are further analyzed in terms of circle relationship and has feature relationship. For defining feature data , collection of ego user and all the related friend id has been merged together and specified under 'featnames' data file , which is used for feature mapping of the ego user and friends of ego user.



4.Data Import

For Neo4j:

- Used Cypher's queries to dynamically load the formatted file from git to Neo4jAura. 'LOAD CSV WITH HEADERS' command to import CSV files.

- Manually uploaded file to Aura and established relationship using the graph model definition. ‘RUN IMPORT’ used to upload the data into Neo4jAura. (For bigger files, Neo4j aura cannot dynamically upload it , with the free tier option)

6 Visualization

- Neo4j browser provided an interactive way to visualize the social graph.
- Users and their features/circles were displayed using color-coded nodes and styled relationships.

Innovation

Data driven multi model strategy

Rather than applying the same dataset across both Neo4j and MongoDB, this project takes a novel approach by selecting distinct datasets optimized for each system. Neo4j makes use of the Facebook Ego Network to fully utilize its graph-based relationships, whi ch are perfect for depicting circles, friendships, and user-feature mappings. Conversely, MongoDB is used for a separate dataset that better aligns with document-oriented modeling, such as activity logs or user metadata. This approach acknowledges that real-world systems often manage heterogeneous datasets and require tailored storage solutions. It also underscores the importance of selecting a database not just by popularity but by structural and operational fit.

GitHub-Hosted Data Integration

All Neo4j imports are conducted using CSV files hosted on GitHub, demonstrating how external data sources can be pulled into a graph database without requiring local filesystem access. This not only increases portability and shareability, but also creates opportunities for collaborative open-source development and live demos.

Dual-Mode Visualization

Most projects focus on a single form of visualization, but this one incorporates:

- **Graphical Visualization** using Neo4j’s native tools for interactive exploration of social networks, features, and communities.
- **Analytical Visualization** using Python’s Matplotlib to create histograms and charts based on feature prevalence, circle sizes, and more.

This layered visualization strategy provides both exploratory and explanatory value.

Real-World Data System Simulation

Finally, the project does not simulate a single-use case but reflects how actual data ecosystems are managed in industry—using multiple systems, multiple languages. The decisions made in modeling, scripting, visualizing, and deploying the data environment simulate the kinds of trade-offs and architectures seen in real-world applications, offering a sandbox for applied innovation.

Technical Difficulty

Data preprocessing, graph modeling, document structuring, dynamic querying, containerization, and were among the many technological problems this project brought.

Handling Raw and Inconsistent Data Formats

The input files, such as feat, egofeat, circles, were raw text files without headers or clear structure. These files had to be interpreted and cleaned manually with Python scripts.

- Add appropriate headers programmatically
- Align user IDs with their corresponding features across files.
- Normalize and export data for use with Neo4j and MongoDB. This was particularly complex due to the high dimensionality of the features and their binary nature.

Advanced Cypher Query Design

Cypher queries in this project go far beyond simple MATCH and MERGE patterns. Key technical accomplishments include Aggregation of features by circle or by user ID. Mapping many-to-many relationships between users and features, and users and circles. These required an in-depth understanding of Cypher's execution model and list processing.

Sharding and cluster computing:

High level details of sharding and clustering in Neo4j is given below with few setups config required. Current Neo4j Aura free tier version did not allow the access and configuration of the same.Hence implementation is not achieved in current graph project.

Sharding :

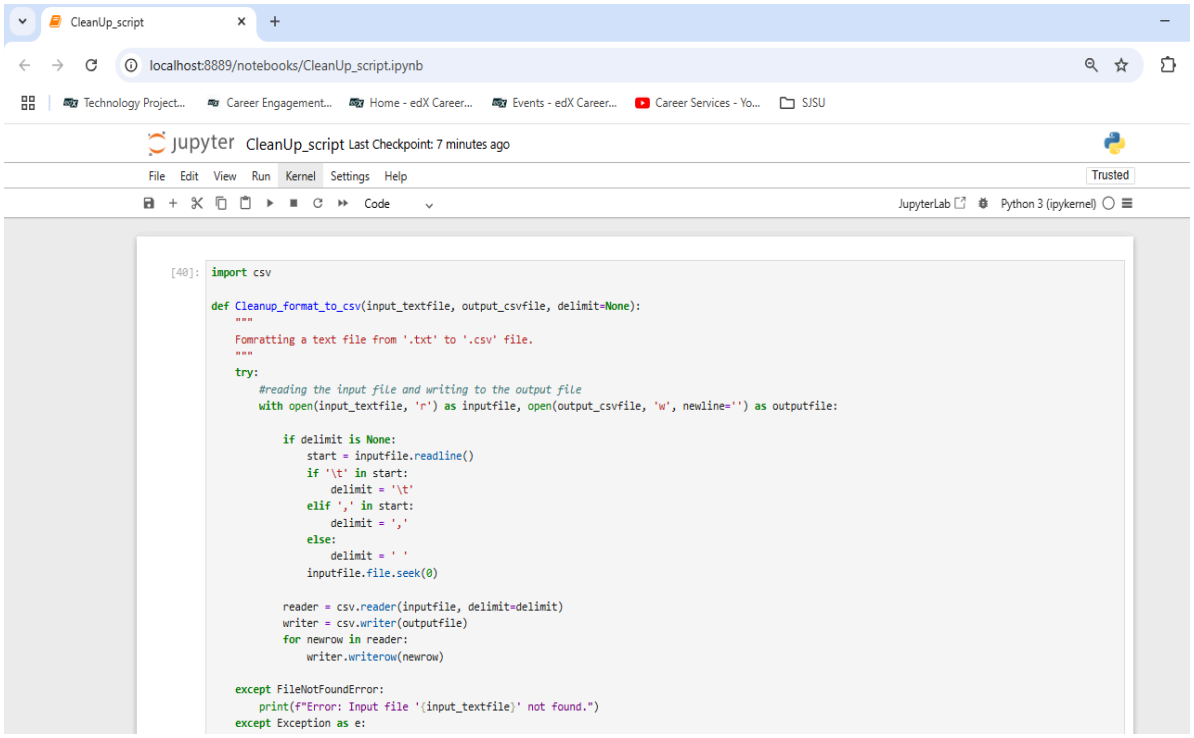
Neo4j 4.0 has a huge update, named Fabric.Fabric comes with it's own database that acts as an entry point to the Neo4j environment. A driver will connect to a *proxy* server or cluster of proxy servers with a set of configuration on it to give it a picture of each shard. There is then a new Cypher USE keyword introduced in 4.0 that will allow you to query across shards.Although we can query across the shards, traversing across the shards is not allowed.

Clustering setup :

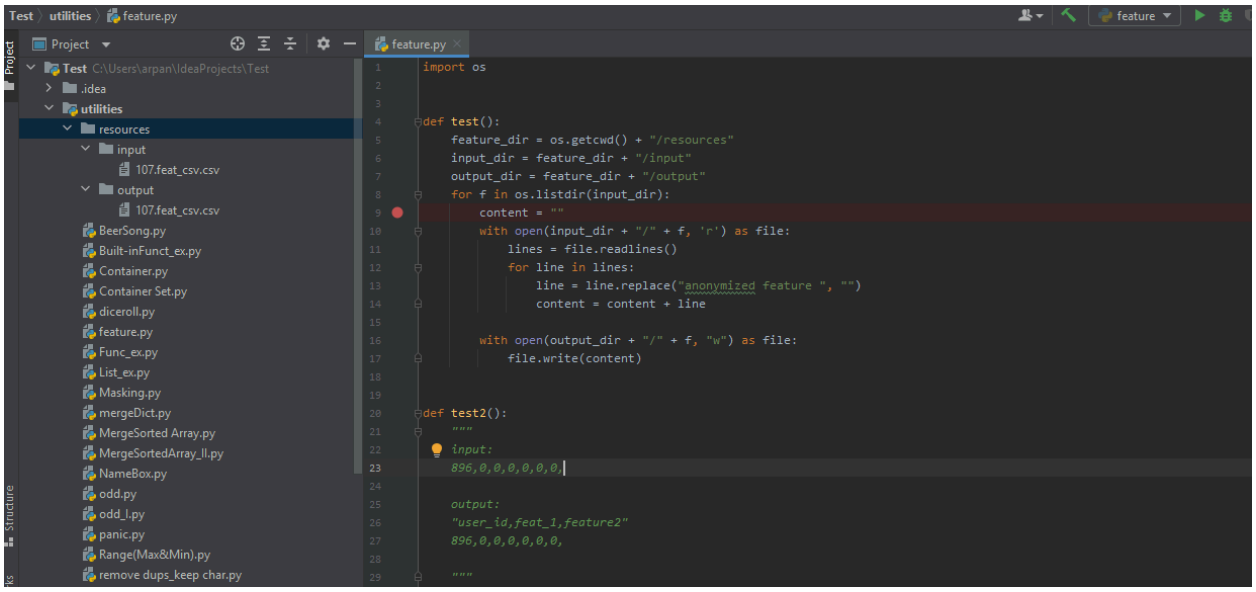
- Server.default_advertised_address : Domain name or IP address that other machines are configured to connect.
- Server.default_listen_address : Usually set up to "0.0.0.0" binding to all the available network.
- Dbms.cluster.endpoints : List of endpoints where a given server can discover other cluster members.
- Initial.dbms.default_primaries_count : No of endpoints in primary mode. If not defined ,it defaults one of the endpoint as 'primary mode'
- Initial.dbms.default_secondaries_count : No of databases hosting in secondary mode.

Use of Cypher/Python to perform meaningful tasks :

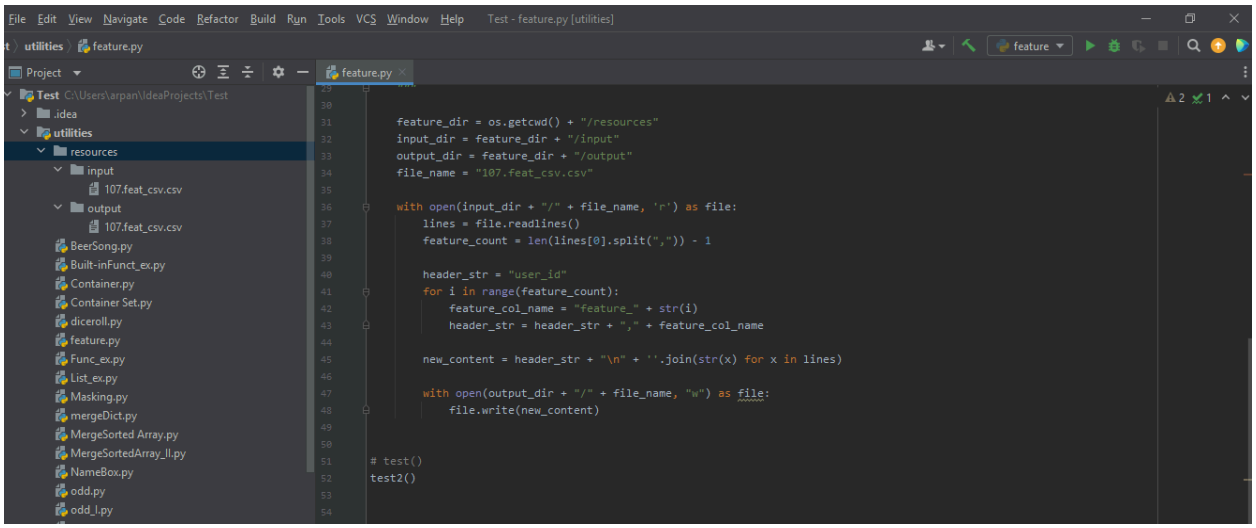
- **Python has been intensively used to convert the data from text file to csv file .**



-Python script has been used to merge features with correct header and the feature numbers.



-Python script has been used to cleanup featnames file , display only feat number and feat description for loading and correct mapping into Neo4j.



-Cypher query to load data into Neo4j Aura :

```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ArpanaSinghSJSU/DataWarehouse/refs/heads/main/faceboo
k_107circles_reformatted.csv' AS row
MERGE (ego:user1 {id: 107})
MERGE (u:user2 {id: toInteger(row.friend_id)})
MERGE (c:Circle {name: row.circle_name, owner: 107})
MERGE (u)-[:IN_CIRCLE]->(c)
```

o/p : Created 491 nodes, created 501 relationships, set 500 properties, added 491 labels

3.create features.

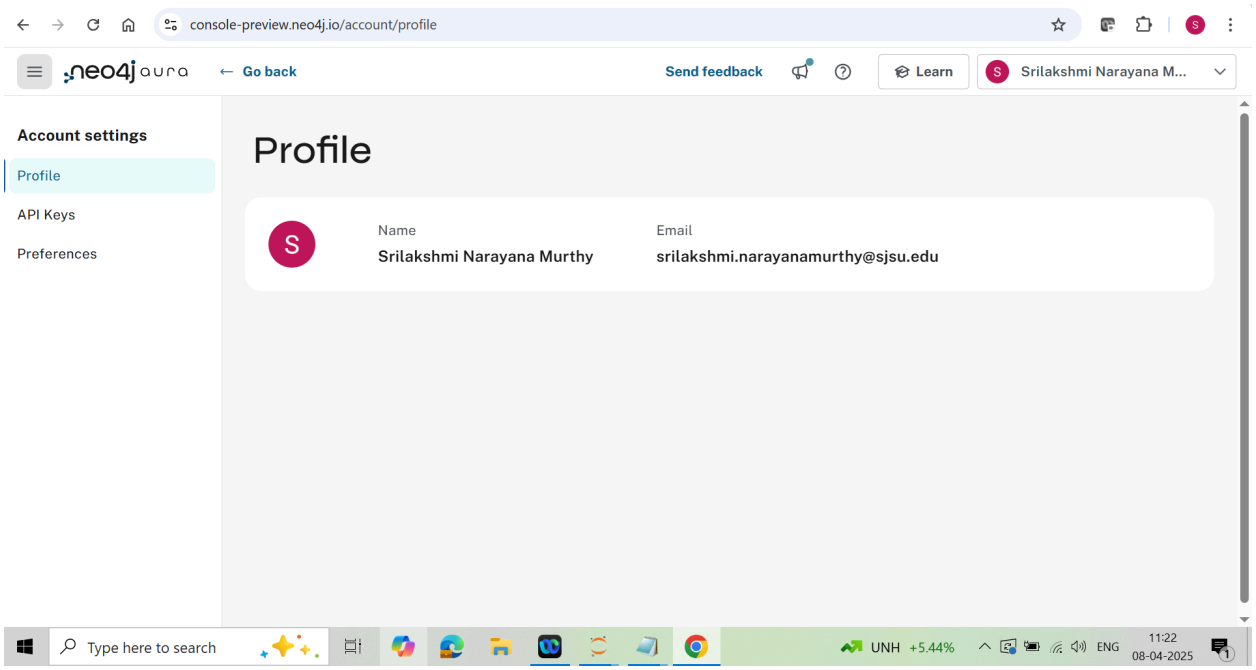
```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ArpanaSinghSJSU/DataWarehouse/refs/heads/main/107.feat
names_csvl.csv' AS row
WITH DISTINCT row.feature_num AS featnum, row.feat_name AS featname
MERGE (:Feature {number: featnum, name: featname})
```

Created 576 nodes, set 1152 properties, added 576 labels

Visualization and Analytics using Queries

DATASET-https://snap.stanford.edu/data/ego-Facebook.html. This dataset is taken from stanford.edu.

Setting up a neo4j aura account for graph databases



Setting up and creating an instance

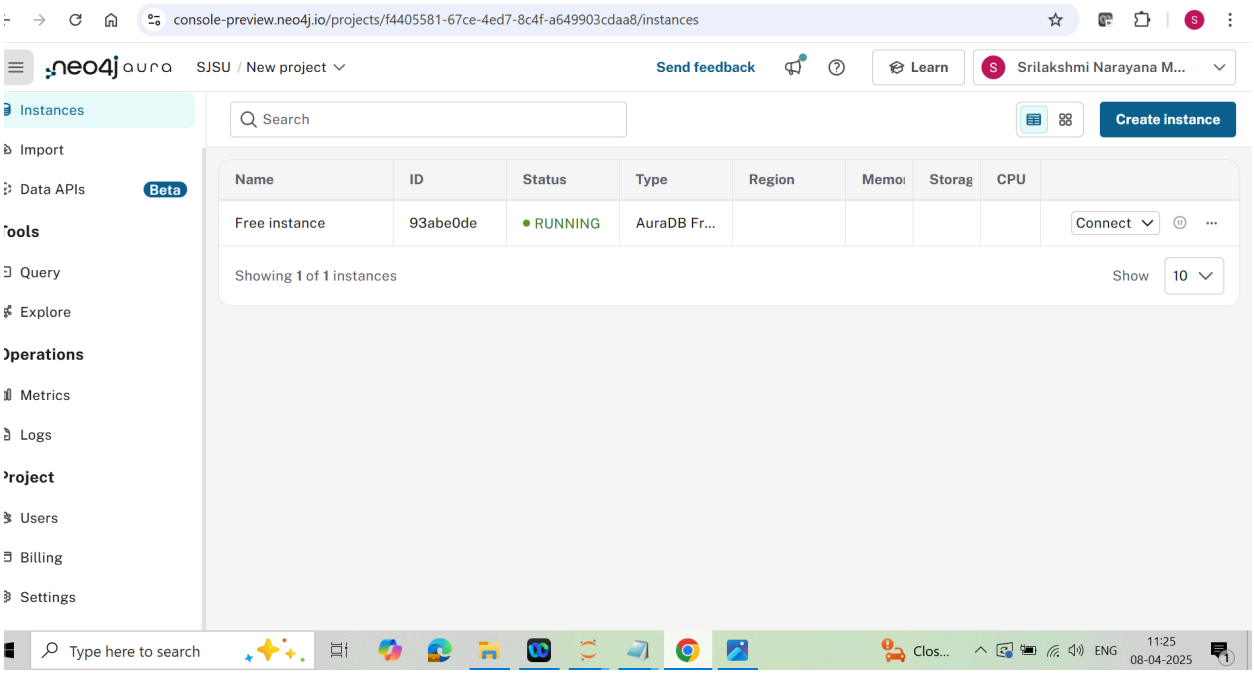
NEO4J_URI=neo4j+s://93abe0de.databases.neo4j.io

NEO4J_USERNAME=neo4j

NEO4J_PASSWORD= **********

AURA_INSTANCEID=93abe0de

AURA_INSTANCENAME=Free instance



Explanation of dataset

Our dataset contains mainly 5 types of files. Edges, Circles, Feat, Egofeat, Featnames.

Edges - This captures the friendship relationships in the ego-network. In the edges file, each row contains two user IDs(User1, User2). **These are modeled as Relationship FRIEND in neo4j.**

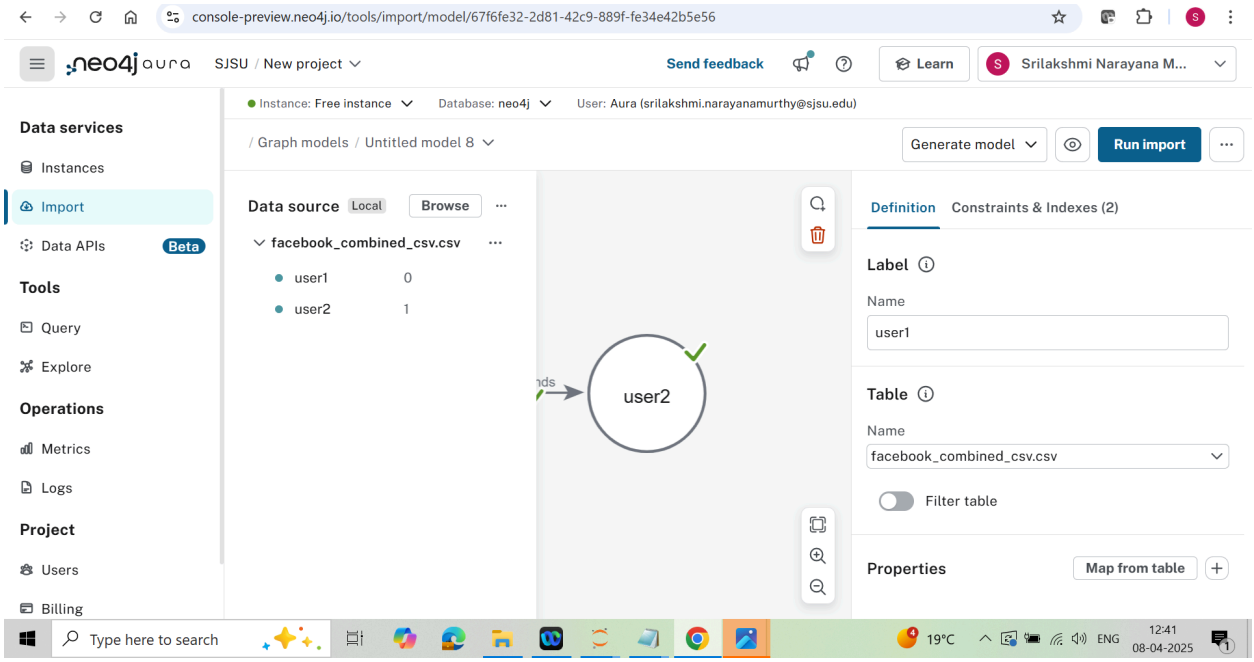
Circles - This represents the groupings (or friend lists) created by the ego user. Each line contains a circle name followed by a list of user IDs in that circle. **These are modeled as Relationship IN_CIRCLE with user in neo4j.**

Feat - This represents the feature vectors of every friend of the ego user. These features captures the profile attributes and behaviours. They are binary encoded. **These are modeled as Relationship HAS_FEATURE relationship with the user2**

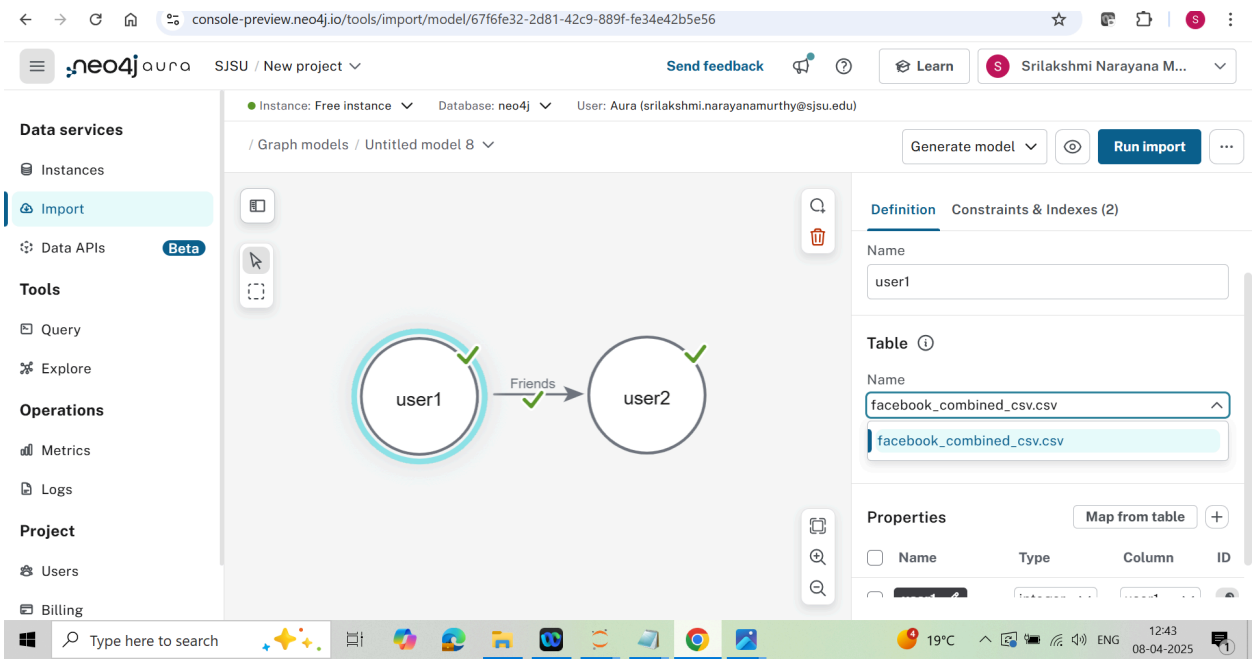
Egofeat - This represents the single line containing the ego user's own feature vector. This just represents the features of just one user. **These are modeled as Relationship HAS_FEATURE relationship with the user1**

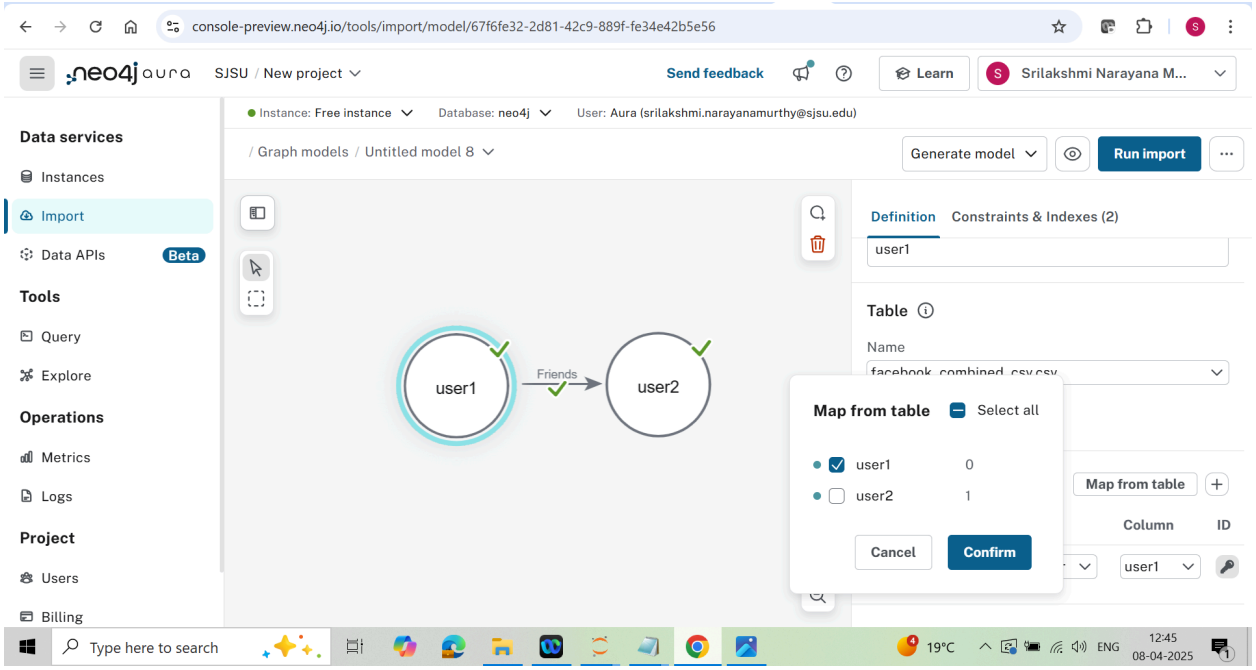
Featnames - This file represents the mapping between feature ID and human-readable feature names. This file describes birthdays, education, school etc. **In neo4j, this enriches users with features for better interpretation**

Importing datasets in Neo4j

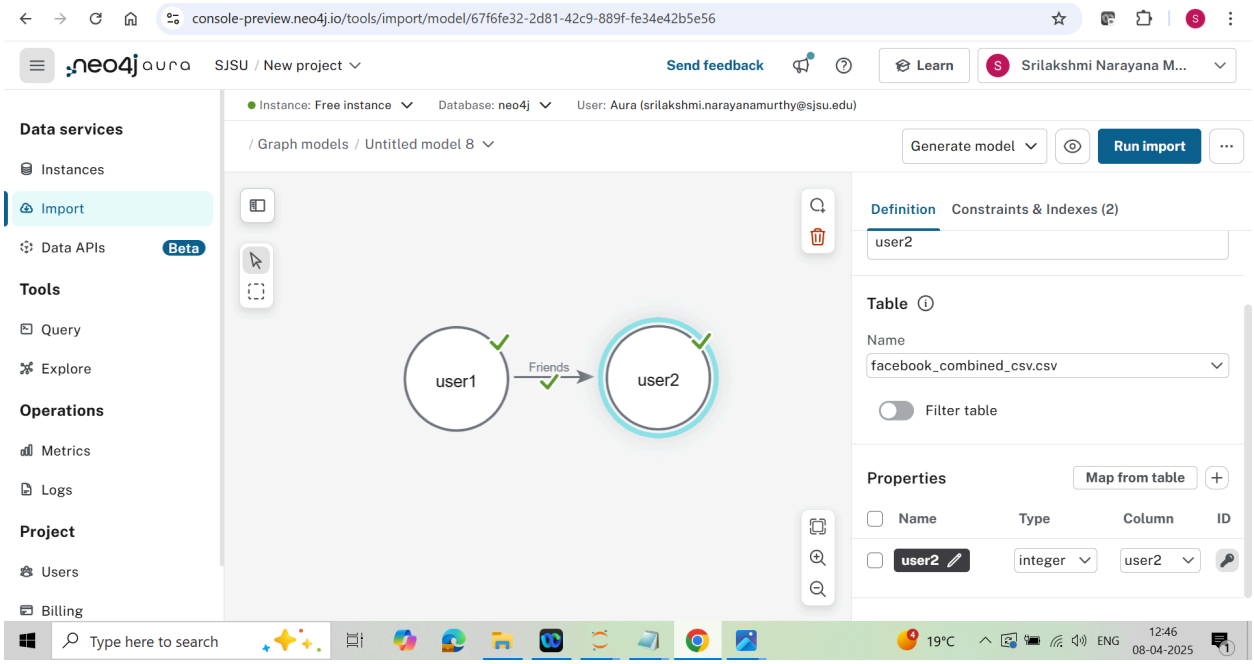


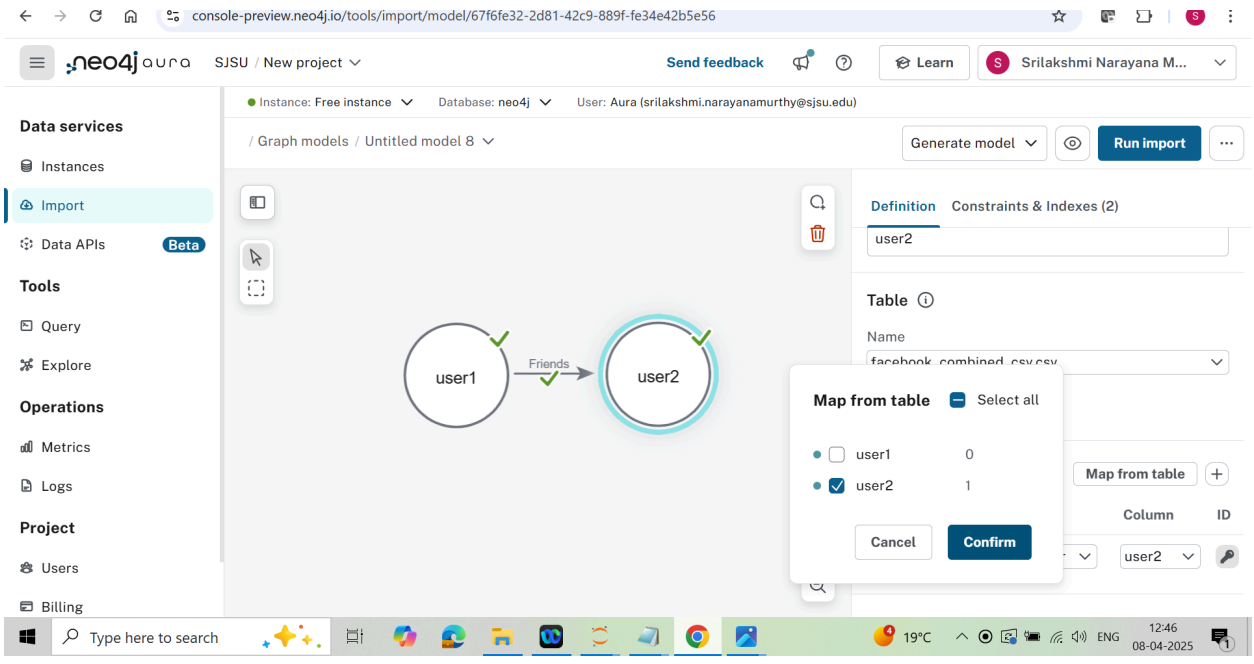
Creating User1 Node and mapping the user1 node with imported dataset



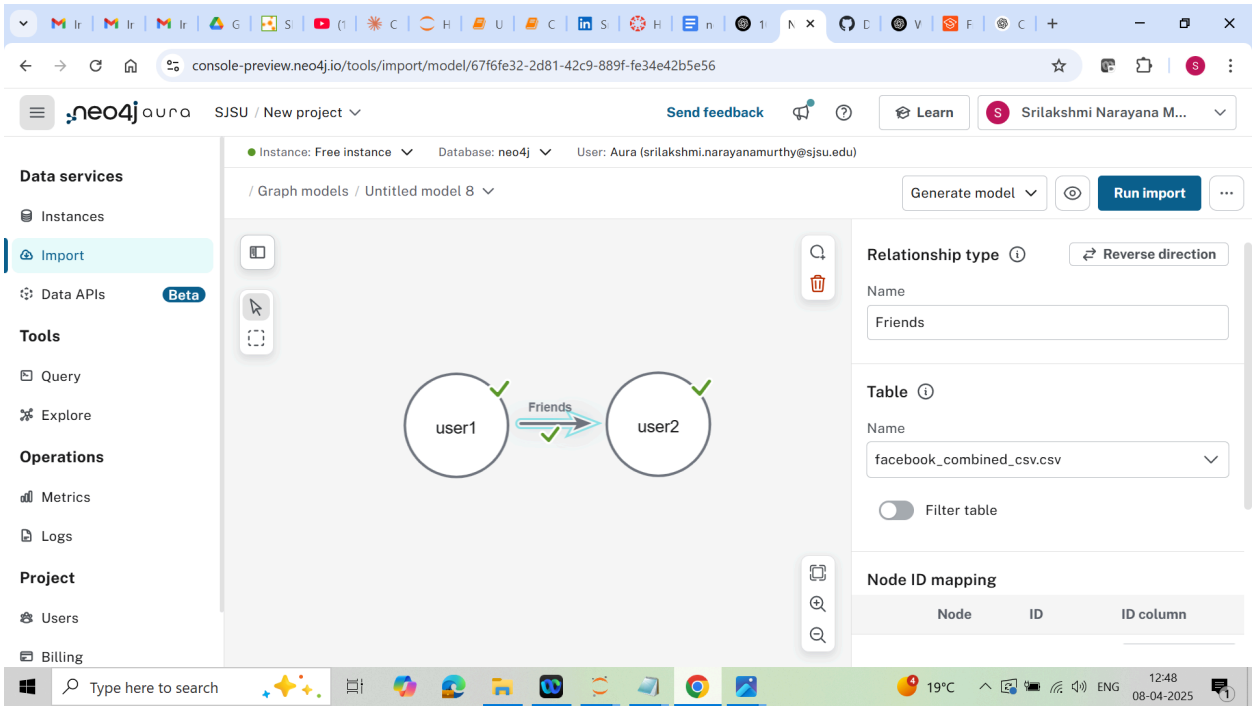


Creating User2 Node and mapping the user2 node with imported dataset

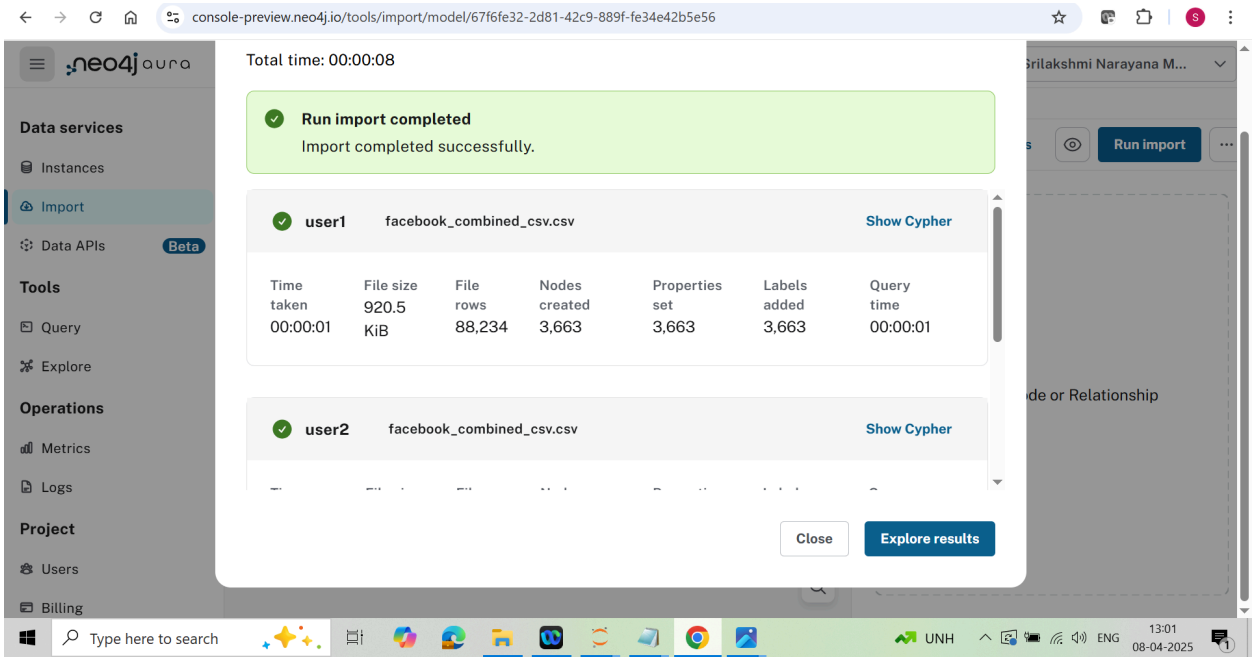

console-preview.neo4j.io/tools/import/model/67f6fe32-2d81-42c9-889f-fe34e42b5e56
neo4j aura
SJSU / New project
Send feedback
Learn
Srilakshmi Narayana M...
Instance: Free instance
Database: neo4j
User: Aura (srilakshmi.narayanamurthy@sjsu.edu)
Data services
Instances
Import
Data APIs
Beta
Tools
Query
Explore
Operations
Metrics
Logs
Project
Users
Billing
/ Graph models / Untitled model 8
Generate model
Run import
user1
Friends
user2
Definition
Constraints & Indexes (2)
user2
Table
Name
facebook_combined_csv.csv
Filter table
Properties
Map from table
Name
Type
Column
ID
user2
integer
user2
Type here to search
19°C
ENG
12:46
08-04-2025

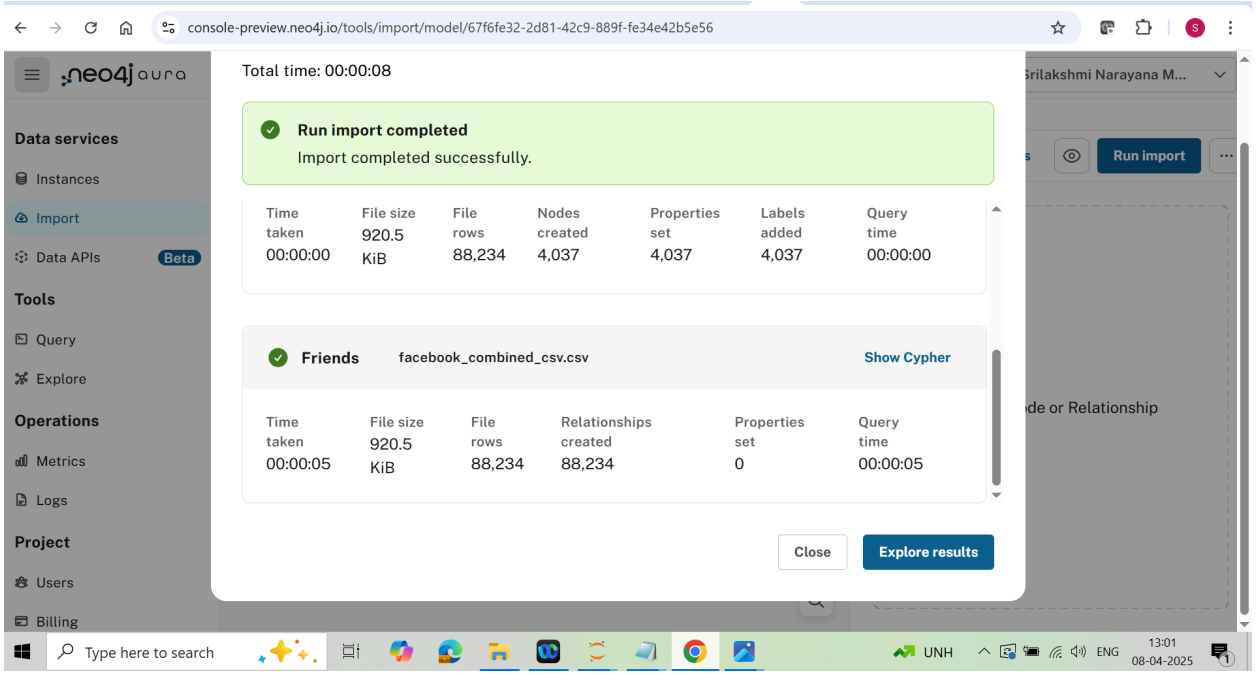

console-preview.neo4j.io/tools/import/model/67f6fe32-2d81-42c9-889f-fe34e42b5e56
neo4j aura
SJSU / New project
Send feedback
Learn
Srilakshmi Narayana M...
Instance: Free instance
Database: neo4j
User: Aura (srilakshmi.narayanamurthy@sjsu.edu)
Data services
Instances
Import
Data APIs
Beta
Tools
Query
Explore
Operations
Metrics
Logs
Project
Users
Billing
/ Graph models / Untitled model 8
Generate model
Run import
user1
Friends
user2
Definition
Constraints & Indexes (2)
user2
Table
Name
Map from table
Select all
user1
0
user2
1
Cancel
Confirm
Map from table
Column
ID
user2
Type here to search
19°C
ENG
12:46
08-04-2025

Creating relationship FRIENDS between User1 and User2



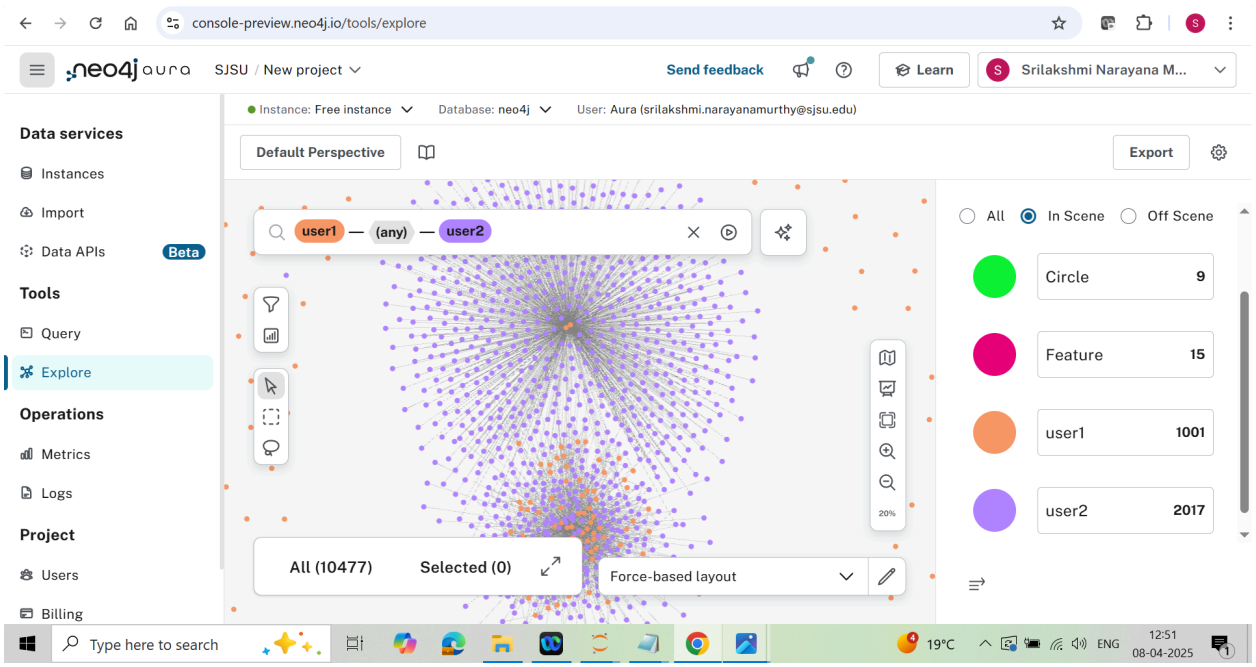
Importing from csv file





COLOR REPRESENTATIONS

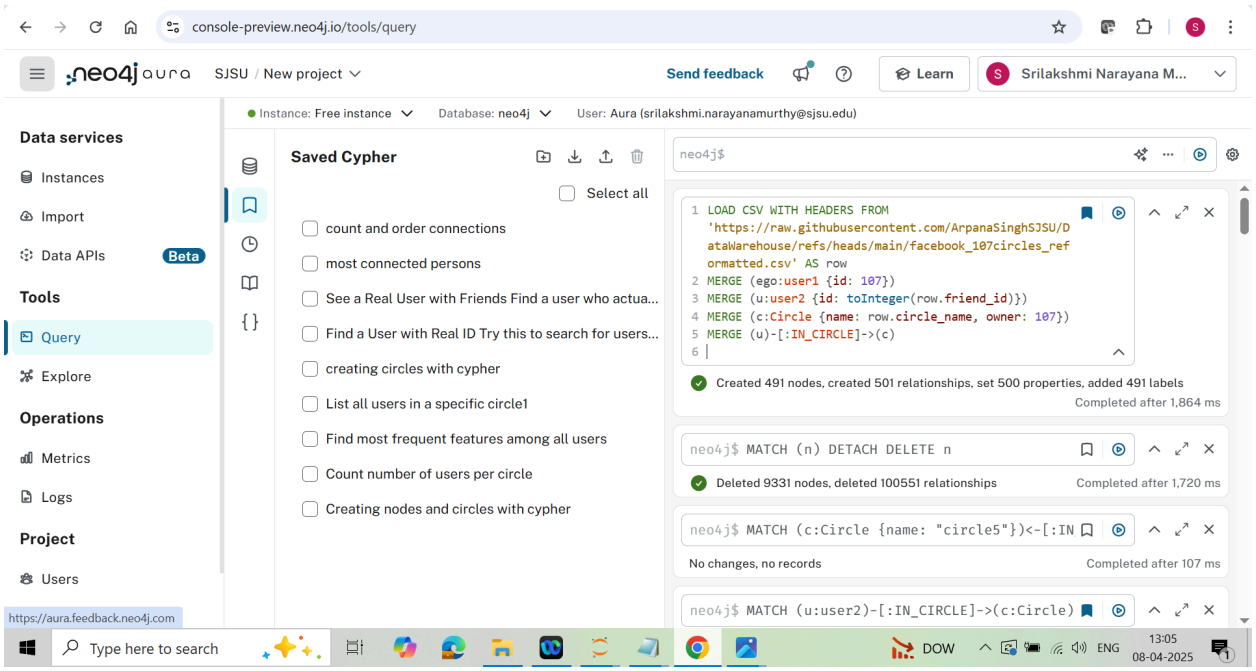
Here User1 is represented in orange. User2 is represented in blue.



Creating circles and edges with cypher

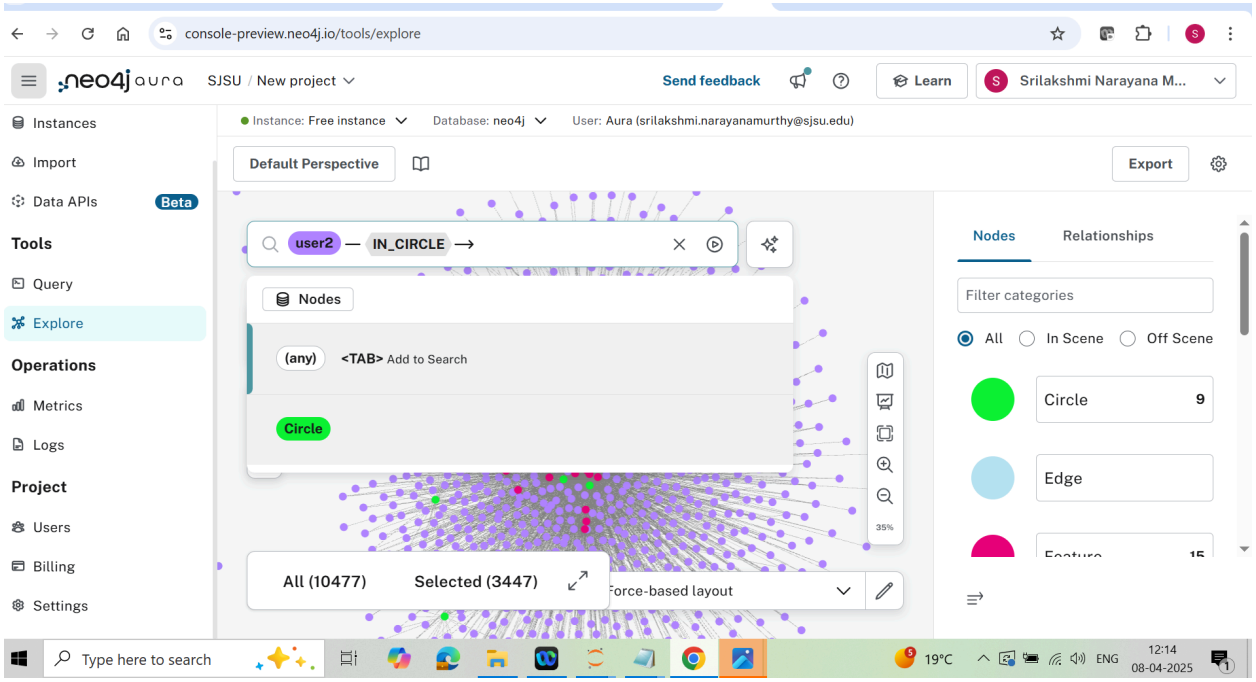
Creating CIRCLES AND EDGES by pulling the circles and edges files from dataset through github and merging all the circles with the particular user with user id 107. In the query, you can see that IN_CICLE relation has been created between user and circle

Created 491 nodes, created 501 relationships, set 500 properties, added 491 labels

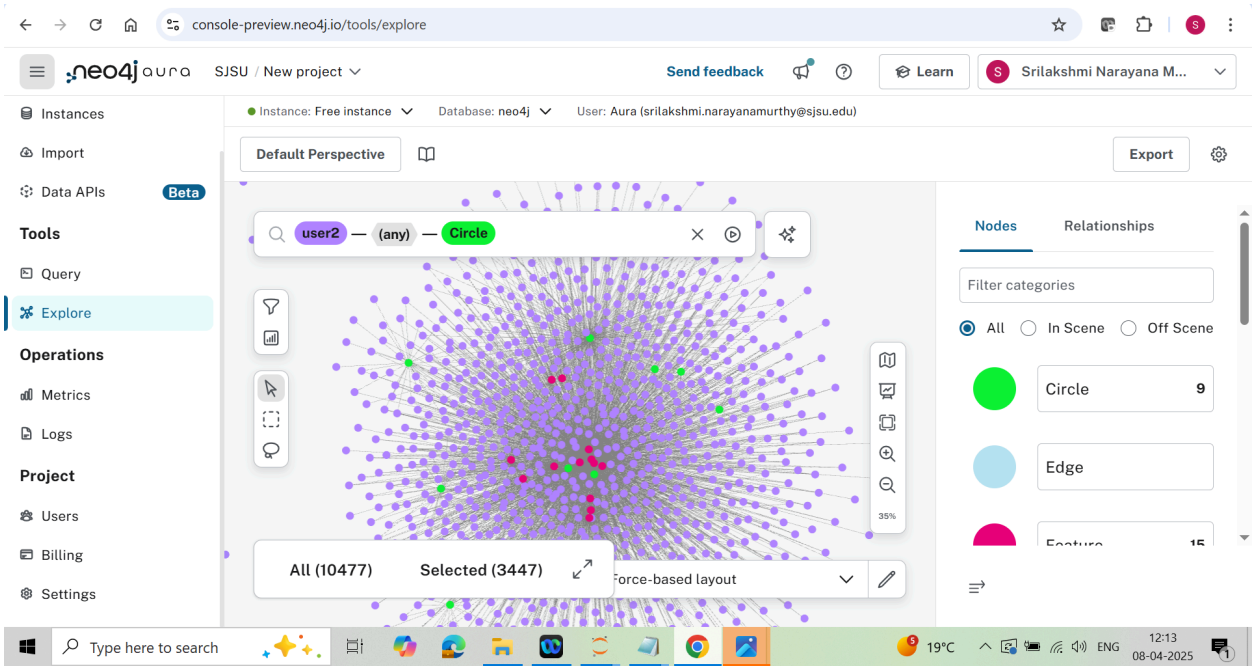


QUERY

```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ArpanaSinghSJSU/DataWarehouse/refs/heads/main/facebook_107circles_reformatted.csv' AS row
MERGE (ego:user1 {id: 107})
MERGE (u:user2 {id: toInteger(row.friend_id)})
MERGE (c:Circle {name: row.circle_name, owner: 107})
MERGE (u)-[:IN_CIRCLE]->(c)
```



Circles are represented in green color



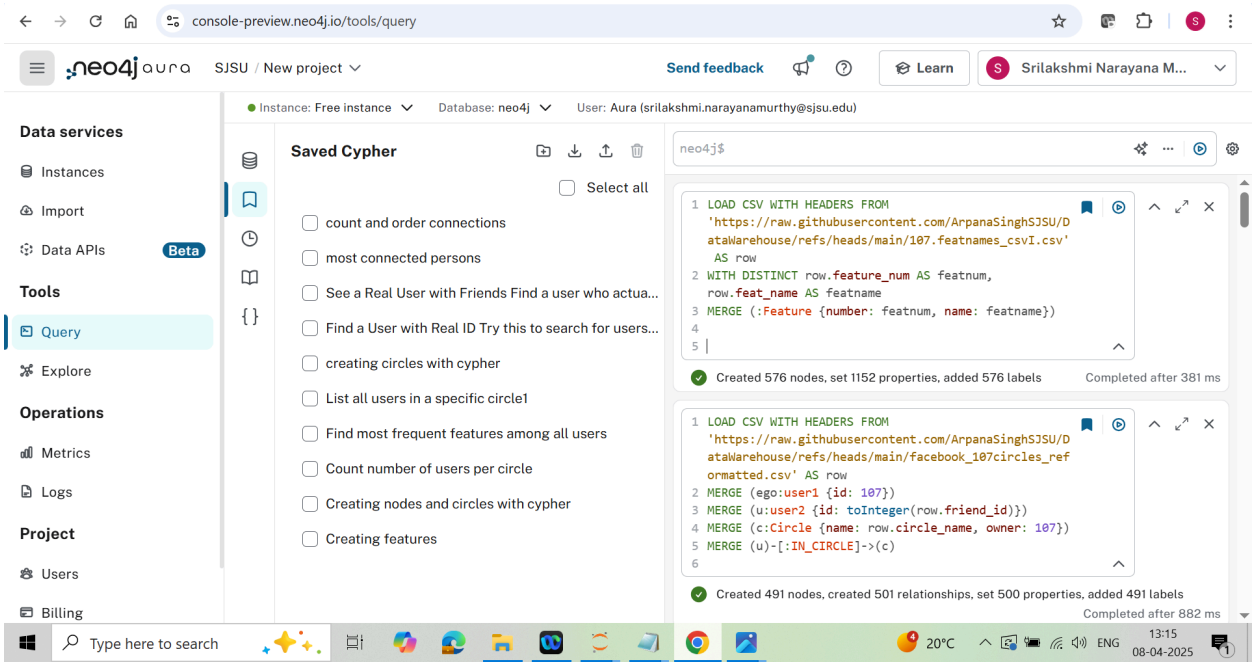
Creating Features

Creating circles by pulling the FEAT_NAMES file from dataset through github by merging featurenumber and featurename

Created 576 nodes, set 1152 properties, added 576 labels

QUERY

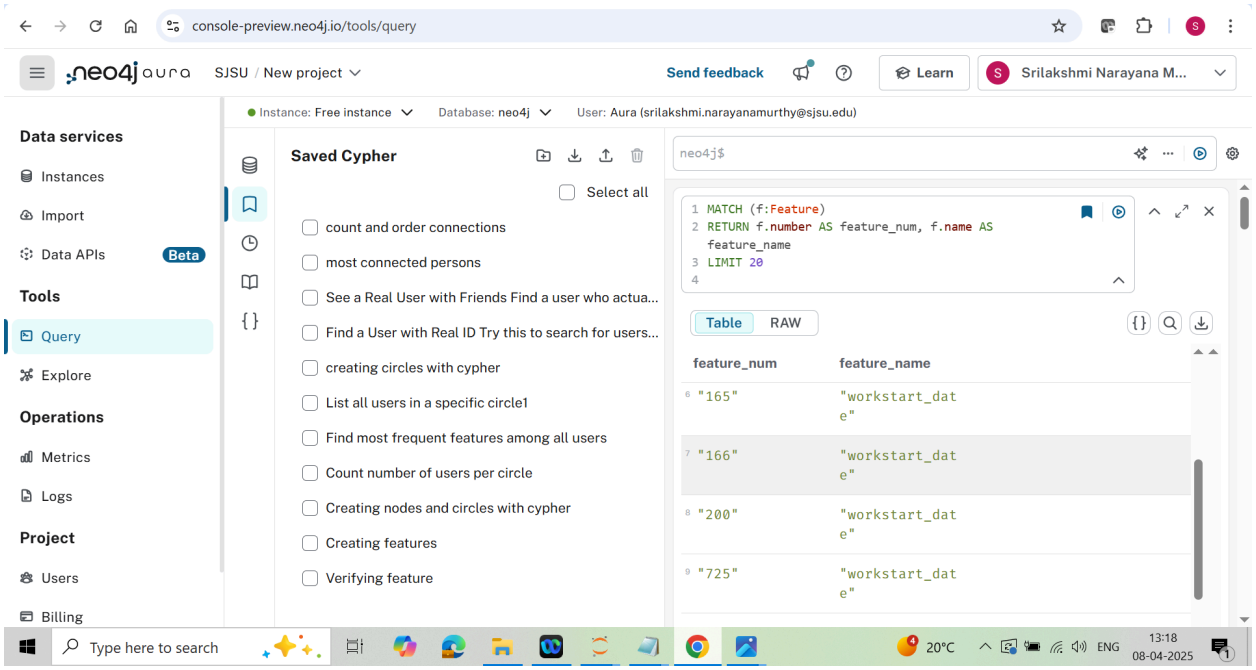
```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ArpanaSinghSJSU/DataWarehouse/refs/heads/main/107.featnames_csvI.csv' AS row
WITH DISTINCT row.feature_num AS featnum, row.feat_name AS featname
MERGE (:Feature {number: featnum, name: featname})
```

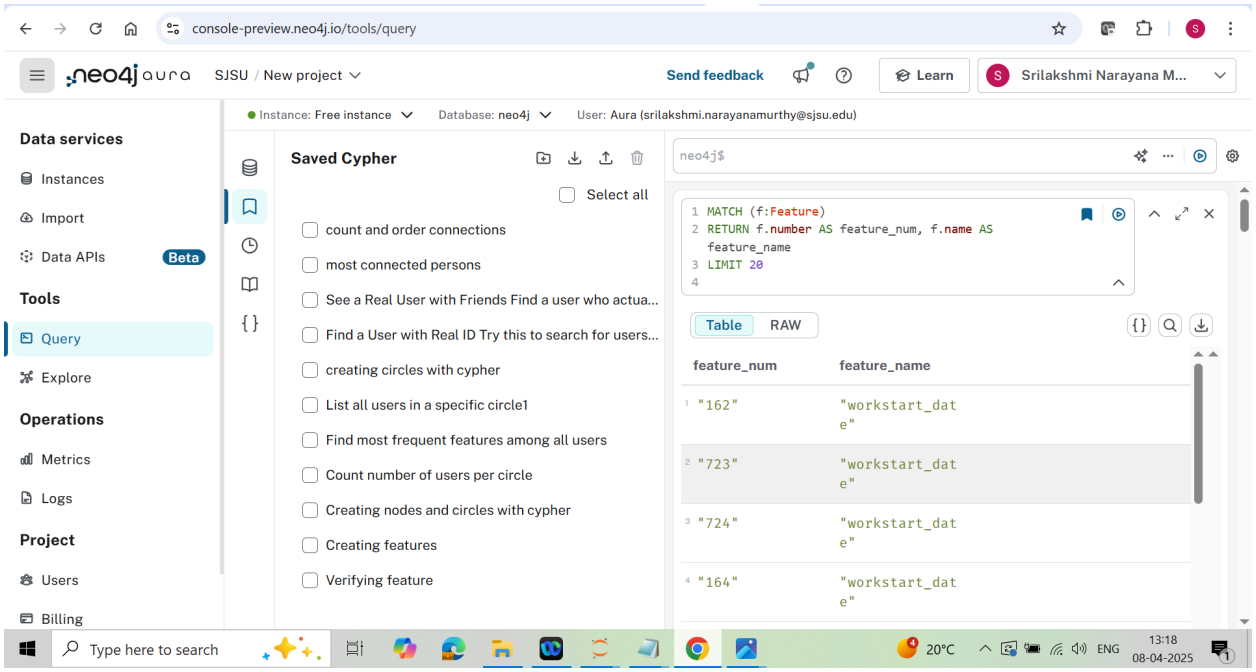


Verifying the feature

```
MATCH (f:Feature)
RETURN f.number AS feature_num, f.name AS feature_name
LIMIT 20
```

This query is matching the features and returning the feature number and feature names and limiting to 20 records.


console-preview.neo4j.io/tools/query
neo4j aura
SJSU / New project
Send feedback
Learn
Srilakshmi Narayana M...
Instance: Free instance
Database: neo4j
User: Aura (srilakshmi.narayanamurthy@sjsu.edu)
Data services
Instances
Import
Data APIs
Beta
Tools
Query
Explore
Operations
Metrics
Logs
Project
Users
Billing
Saved Cypher
Select all
count and order connections
most connected persons
See a Real User with Friends Find a user who actua...
Find a User with Real ID Try this to search for users...
creating circles with cypher
List all users in a specific circle1
Find most frequent features among all users
Count number of users per circle
Creating nodes and circles with cypher
Creating features
Verifying feature
neo4j$
1 MATCH (f:Feature)
2 RETURN f.number AS feature_num, f.name AS feature_name
3 LIMIT 20
4
Table
RAW
feature_num
feature_name
6 "165"
"workstart_date"
7 "166"
"workstart_date"
8 "200"
"workstart_date"
9 "725"
"workstart_date"
Type here to search
20°C
ENG
13:18
08-04-2025

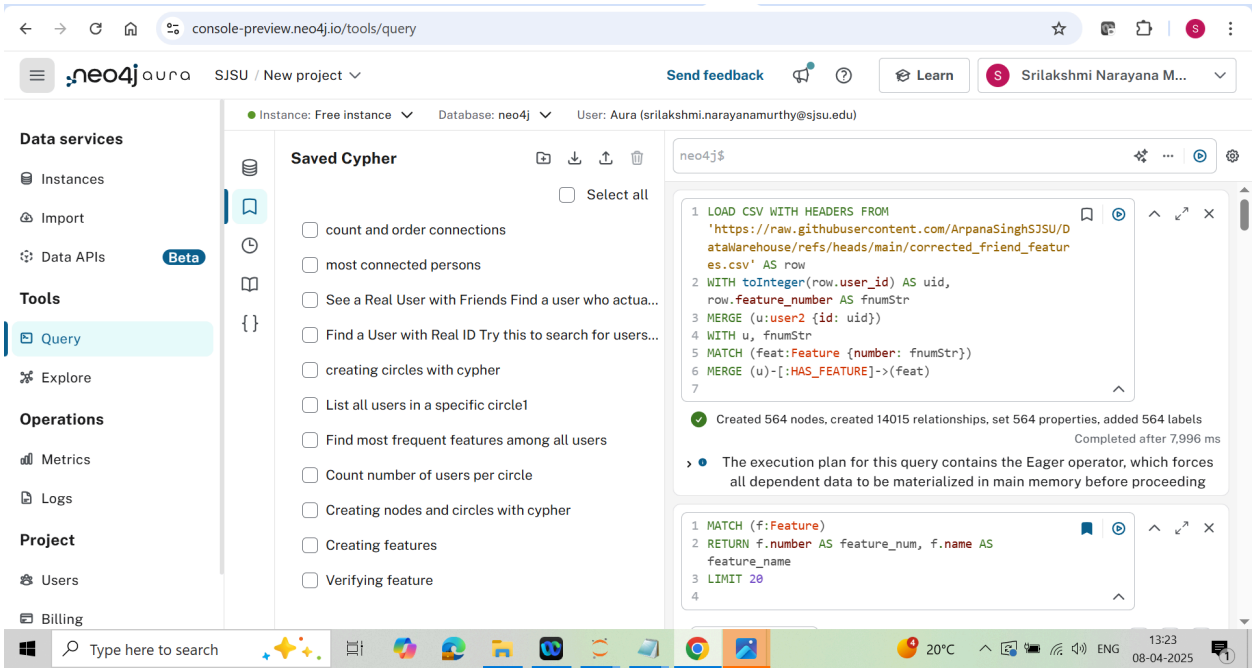

console-preview.neo4j.io/tools/query
neo4j aura
SJSU / New project
Send feedback
Learn
Srilakshmi Narayana M...
Instance: Free instance
Database: neo4j
User: Aura (srilakshmi.narayanamurthy@sjsu.edu)
Data services
Instances
Import
Data APIs
Beta
Tools
Query
Explore
Operations
Metrics
Logs
Project
Users
Billing
Saved Cypher
Select all
count and order connections
most connected persons
See a Real User with Friends Find a user who actua...
Find a User with Real ID Try this to search for users...
creating circles with cypher
List all users in a specific circle1
Find most frequent features among all users
Count number of users per circle
Creating nodes and circles with cypher
Creating features
Verifying feature
neo4j$
1 MATCH (f:Feature)
2 RETURN f.number AS feature_num, f.name AS feature_name
3 LIMIT 20
4
Table
RAW
feature_num
feature_name
1 "162"
"workstart_date"
2 "723"
"workstart_date"
3 "724"
"workstart_date"
4 "164"
"workstart_date"
Type here to search
20°C
ENG
13:18
08-04-2025

Creating relationship between friends and features

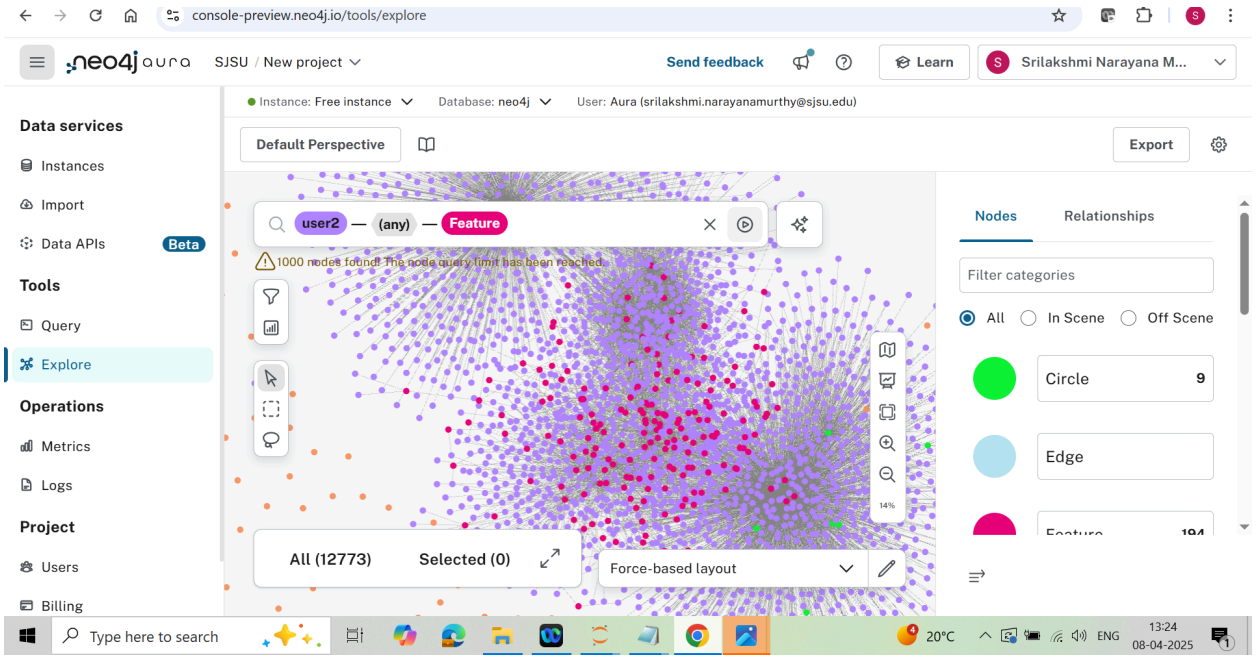
The below query is creating relationship between friends and features with cypher query. We are loading feat file from dataset through github into Neo4j Aura.

QUERY

```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ArpanaSinghSJSU/DataWarehouse/refs/heads/main/corrected_friend_features.csv' AS row
WITH toInteger(row.user_id) AS uid, row.feature_number AS fnumStr
MERGE (u:user2 {id: uid})
WITH u, fnumStr
MATCH (feat:Feature {number: fnumStr})
MERGE (u)-[:HAS_FEATURE]->(feat)
```



Features are represented in pink



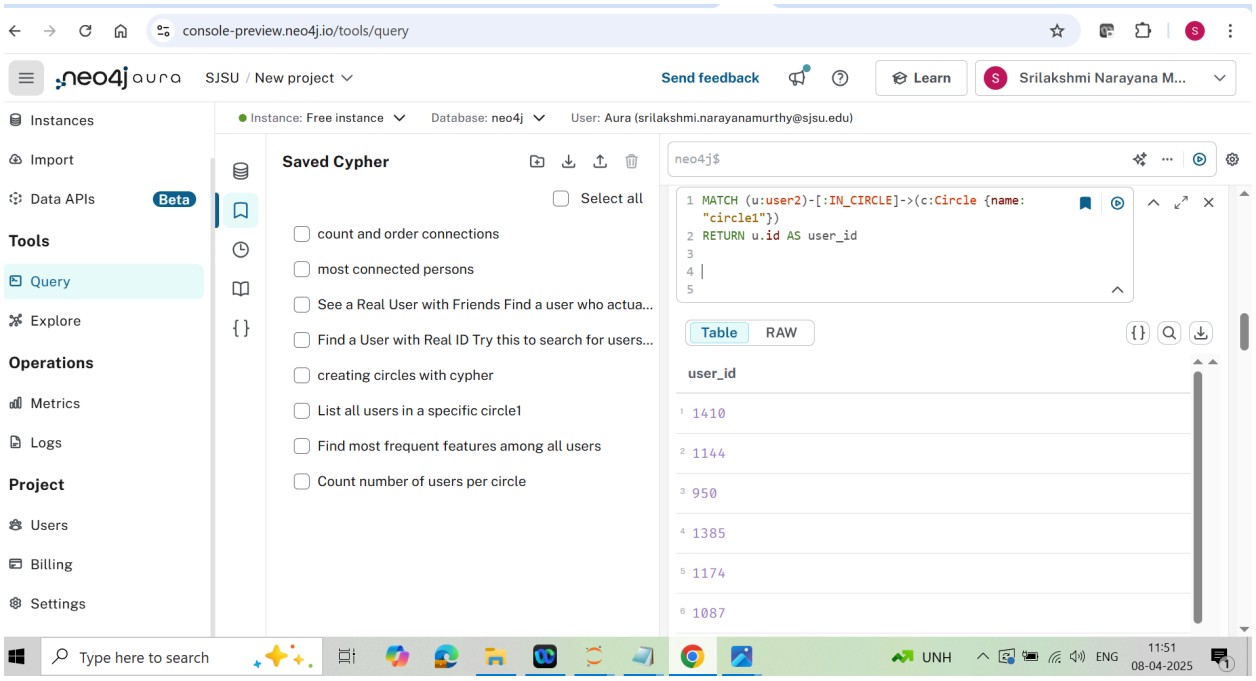
List all users in a specific circle.

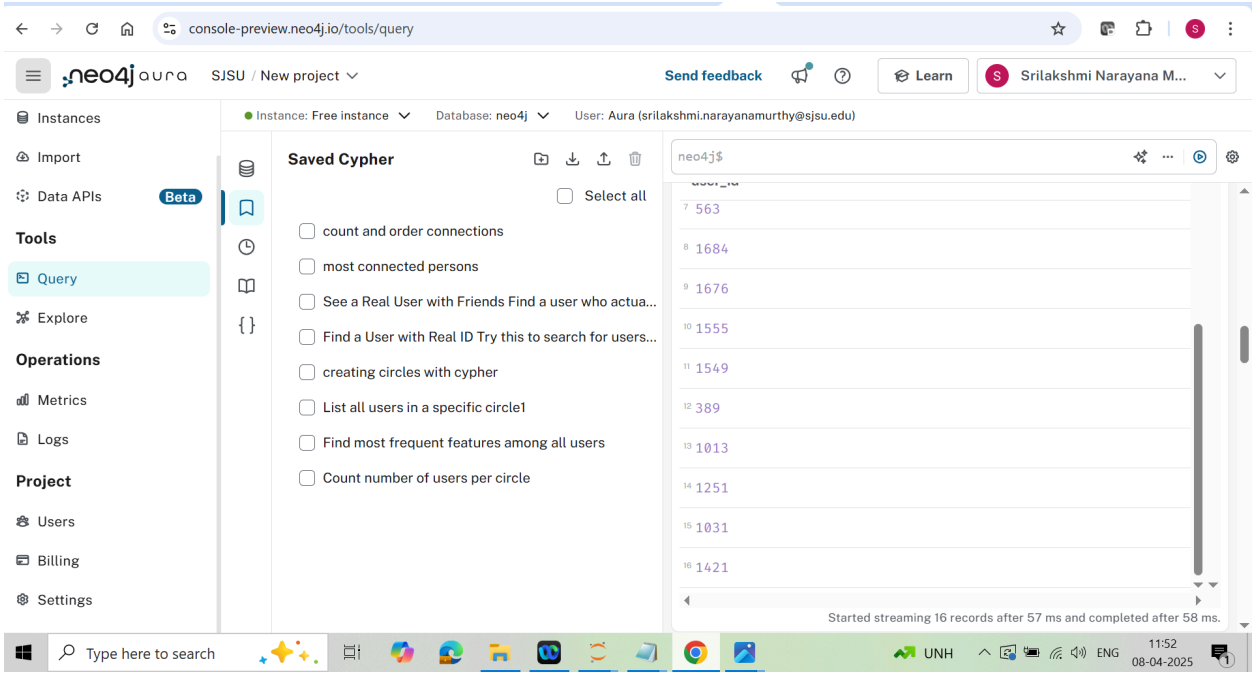
There are different circles present for the users. In each circle, there will be a particular number of users. In order to show all the users in that particular circle, here is the below query. Here we are taking the example of circle1

QUERY

```
MATCH (u:user2)-[:IN_CIRCLE]->(c:Circle {name: "circle1"})

RETURN u.id AS user_id
```

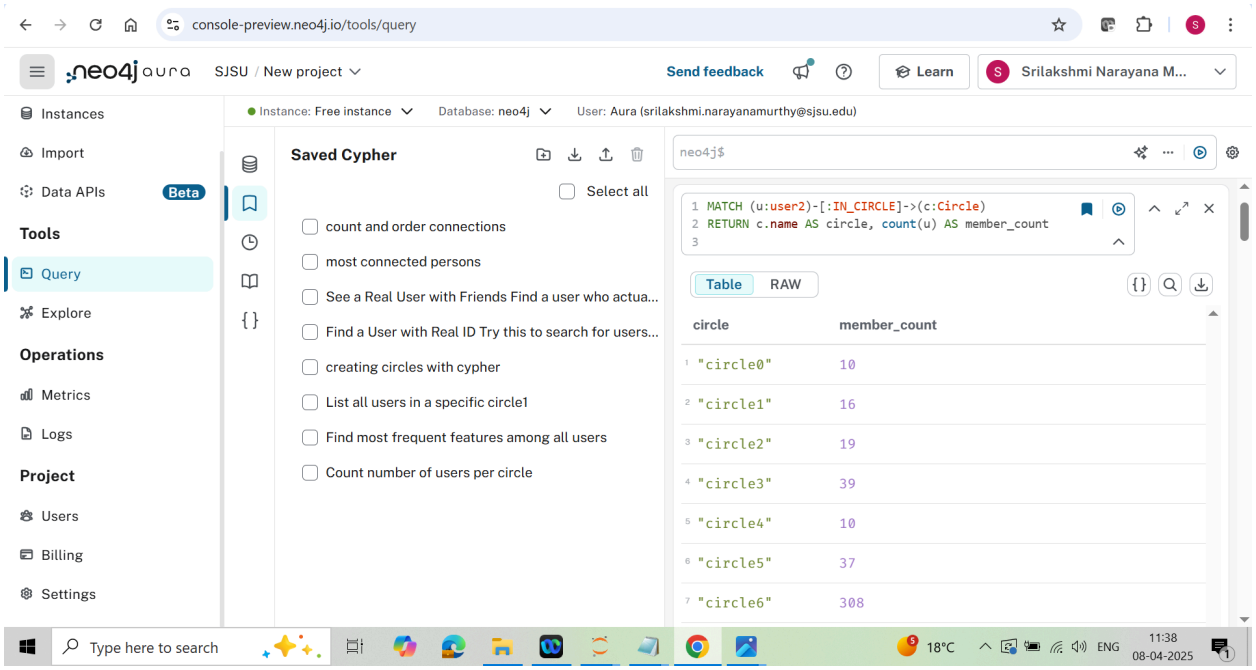




Count number of users per circle

Query - This query matches the count of number of users per circle. For example for user 107 in dataset, there are 8 circles. Hence this query counts number of users per circle.

```
MATCH (u:user2)-[:IN_CIRCLE]->(c:Circle)
RETURN c.name AS circle, count(u) AS member_count
```



Find most frequent features among all users

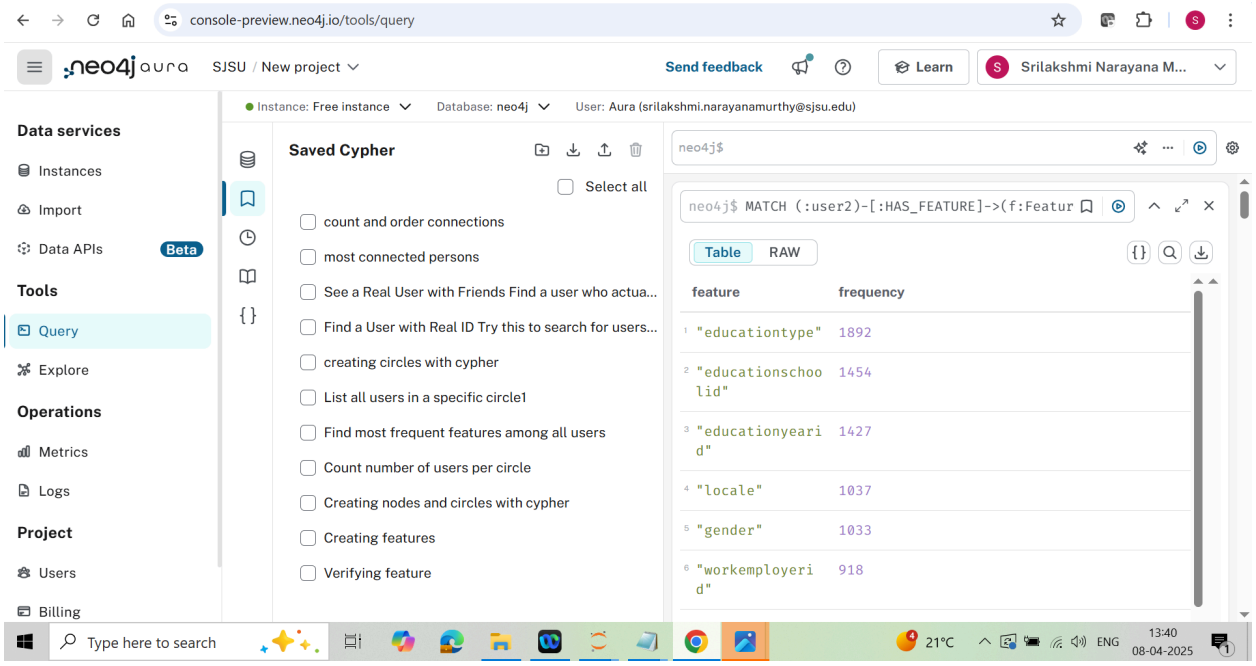
There are many features for many users. In the dataset, presence of features is defined in bits. This query is mentioning most frequent features among all users and ordering them in descending order.

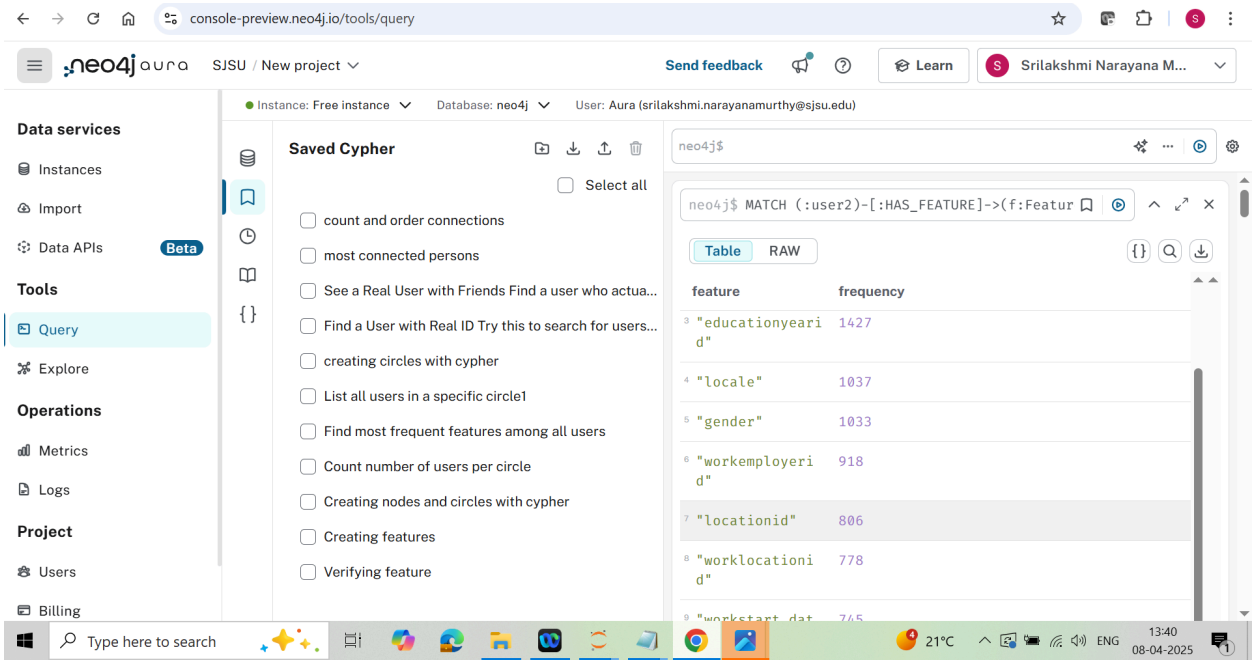
Query

```
MATCH (:user2)-[:HAS_FEATURE]->(f:Feature)

RETURN f.name AS feature, count(*) AS frequency

ORDER BY frequency DESC LIMIT 10
```

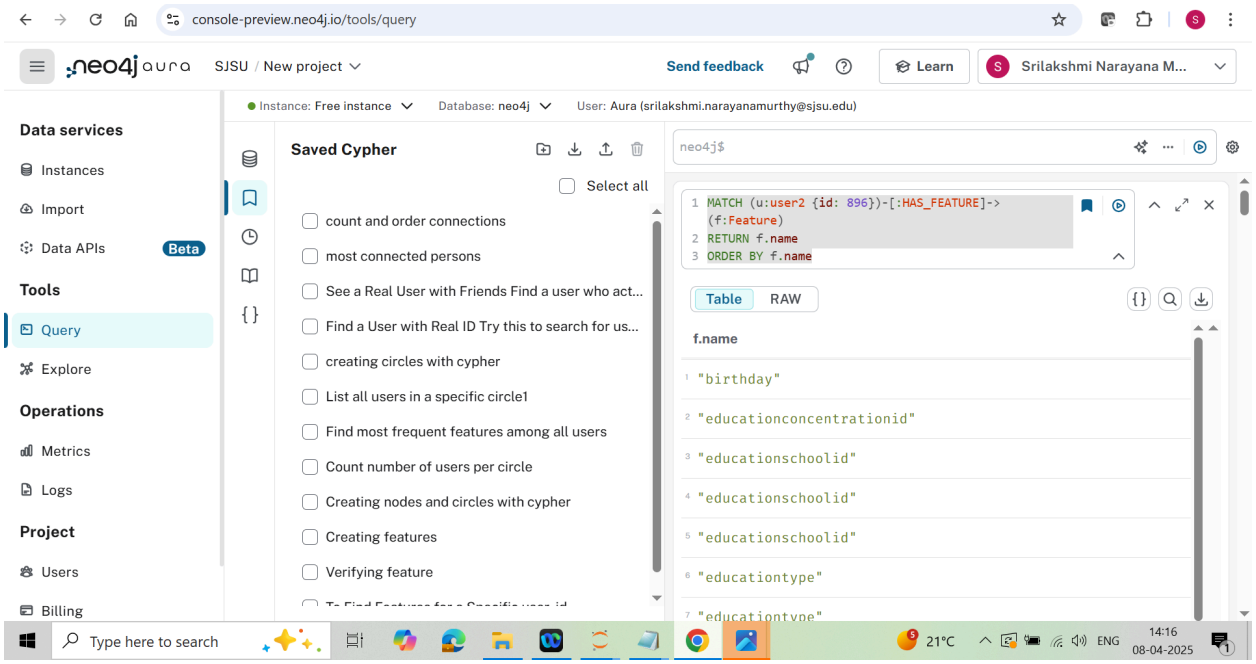


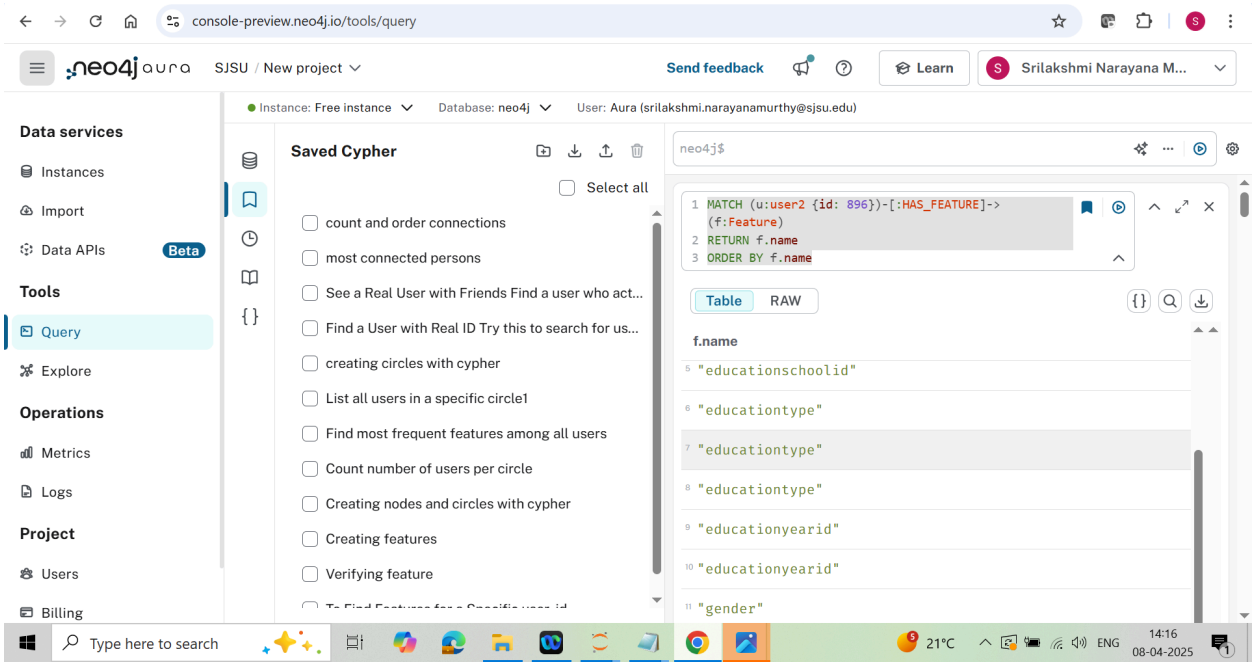


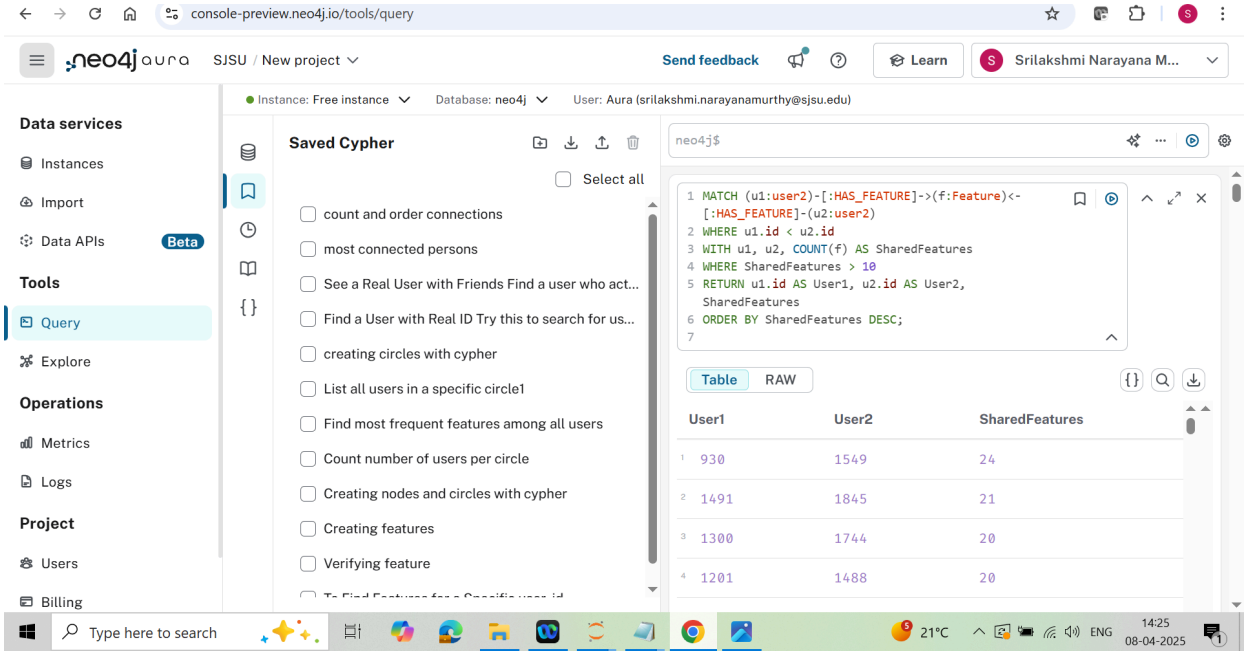
Find features of specific user_id

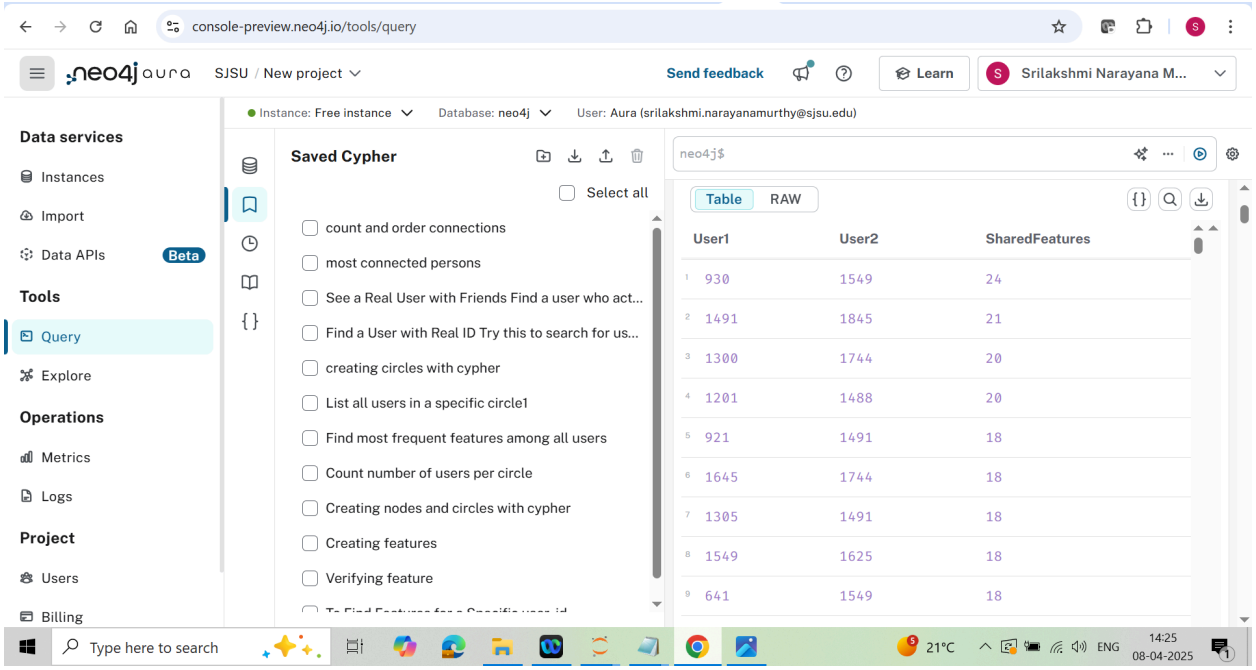
This query is finding the features of specific user id 896

```
MATCH (u:user2 {id: 896})-[:HAS_FEATURE]->(f:Feature)
RETURN f.name
ORDER BY f.name
```




console-preview.neo4j.io/tools/query
neo4j aura
SJSU / New project
Send feedback
Learn
Srilakshmi Narayana M...
Instance: Free instance
Database: neo4j
User: Aura (srilakshmi.narayanamurthy@sjsu.edu)
Data services
Instances
Import
Data APIs
Beta
Tools
Query
Explore
Operations
Metrics
Logs
Project
Users
Billing
Saved Cypher
Select all
count and order connections
most connected persons
See a Real User with Friends Find a user who act...
Find a User with Real ID Try this to search for us...
creating circles with cypher
List all users in a specific circle1
Find most frequent features among all users
Count number of users per circle
Creating nodes and circles with cypher
Creating features
Verifying feature
neo4j$
1 MATCH (u:user2 {id: 896})-[:HAS_FEATURE]->
(f:Feature)
2 RETURN f.name
3 ORDER BY f.name
Table
RAW
f.name
5 "educationschoolid"
6 "educationtype"
7 "educationtype"
8 "educationtype"
9 "educationyearid"
10 "educationyearid"
11 "gender"
Type here to search
21°C
ENG
14:16
08-04-2025

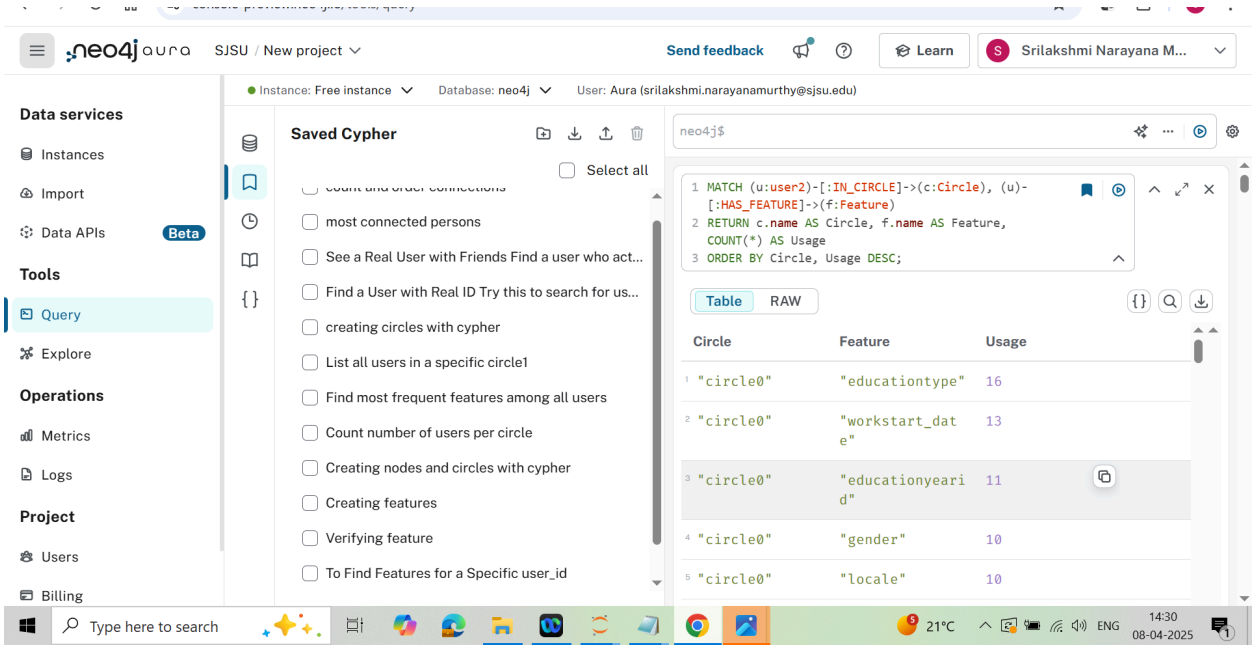

console-preview.neo4j.io/tools/query
neo4j aura
SJSU / New project
Send feedback
Learn
Srilakshmi Narayana M...
Instance: Free instance
Database: neo4j
User: Aura (srilakshmi.narayanamurthy@sjsu.edu)
Data services
Instances
Import
Data APIs
Beta
Tools
Query
Explore
Operations
Metrics
Logs
Project
Users
Billing
Saved Cypher
Select all
count and order connections
most connected persons
See a Real User with Friends Find a user who act...
Find a User with Real ID Try this to search for us...
creating circles with cypher
List all users in a specific circle1
Find most frequent features among all users
Count number of users per circle
Creating nodes and circles with cypher
Creating features
Verifying feature
neo4j$
1 MATCH (u1:user2)-[:HAS_FEATURE]->(f:Feature)<-
[:HAS_FEATURE]-(u2:user2)
2 WHERE u1.id < u2.id
3 WITH u1, u2, COUNT(f) AS SharedFeatures
4 WHERE SharedFeatures > 10
5 RETURN u1.id AS User1, u2.id AS User2,
SharedFeatures
6 ORDER BY SharedFeatures DESC;
7
Table
RAW
User1
User2
SharedFeatures
1 930 1549 24
2 1491 1845 21
3 1300 1744 20
4 1201 1488 20
Type here to search
21°C
ENG
14:25
08-04-2025

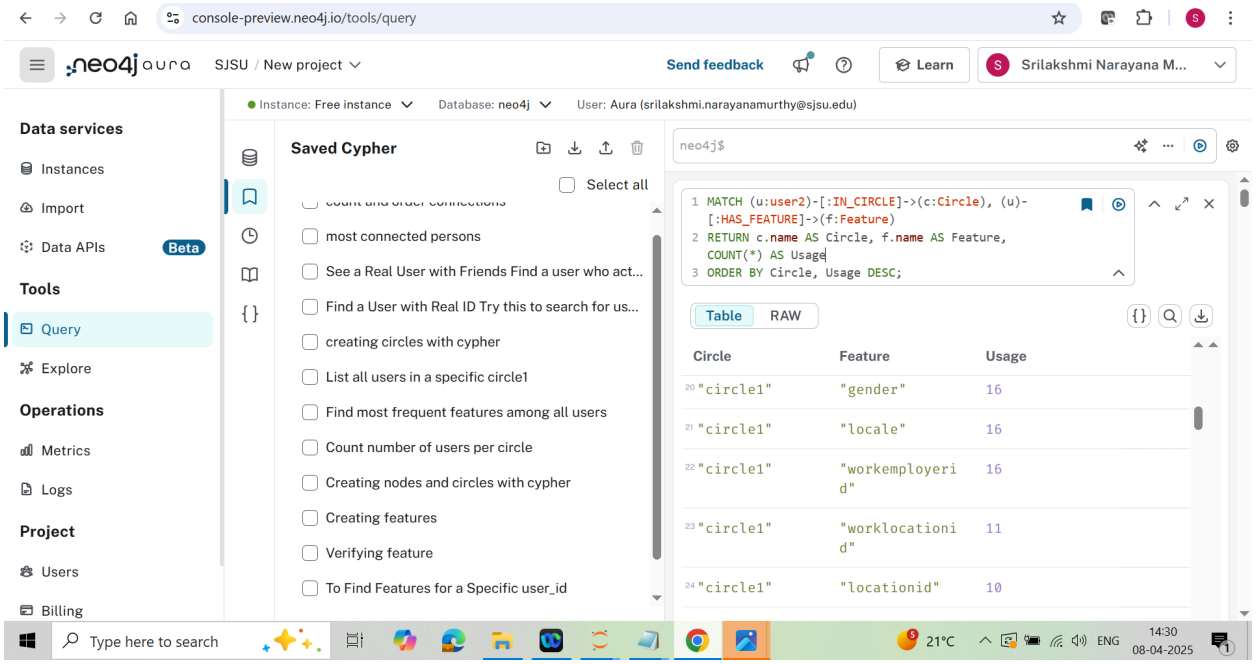


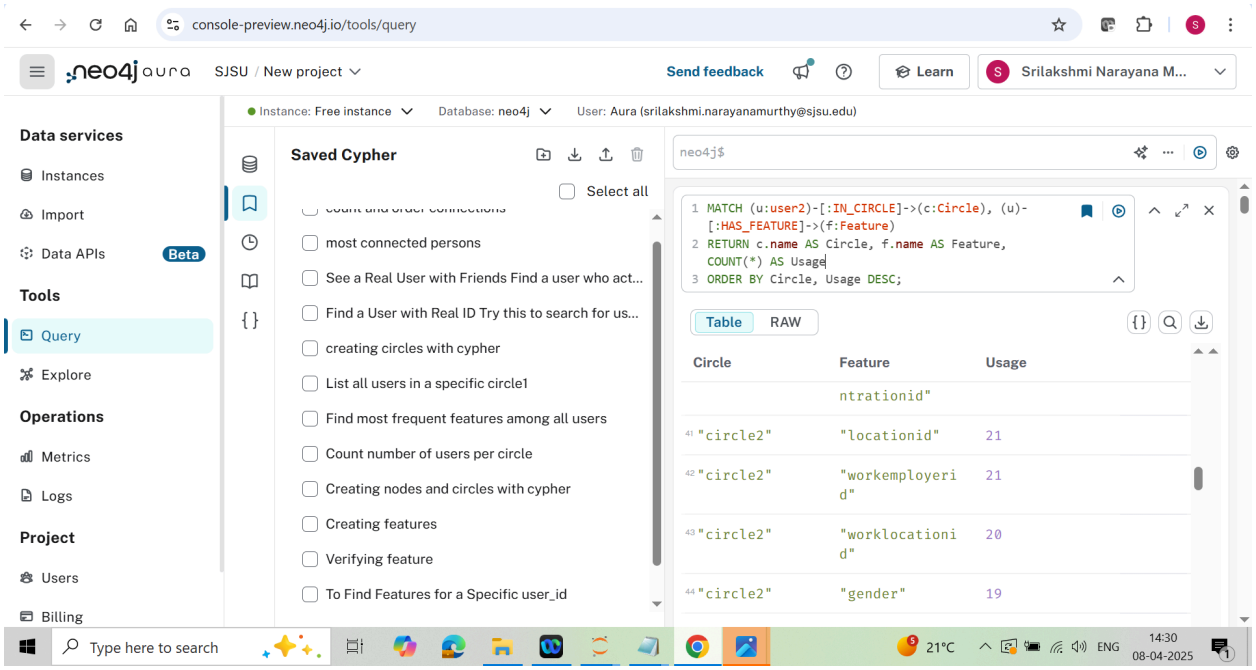
Feature popularity by circle

This query represents the popularity of particular features and its usage in each circle

```
MATCH (u:user2)-[:IN_CIRCLE]->(c:Circle), (u)-[:HAS_FEATURE]->(f:Feature)
RETURN c.name AS Circle, f.name AS Feature, COUNT(*) AS Usage
ORDER BY Circle, Usage DESC;
```







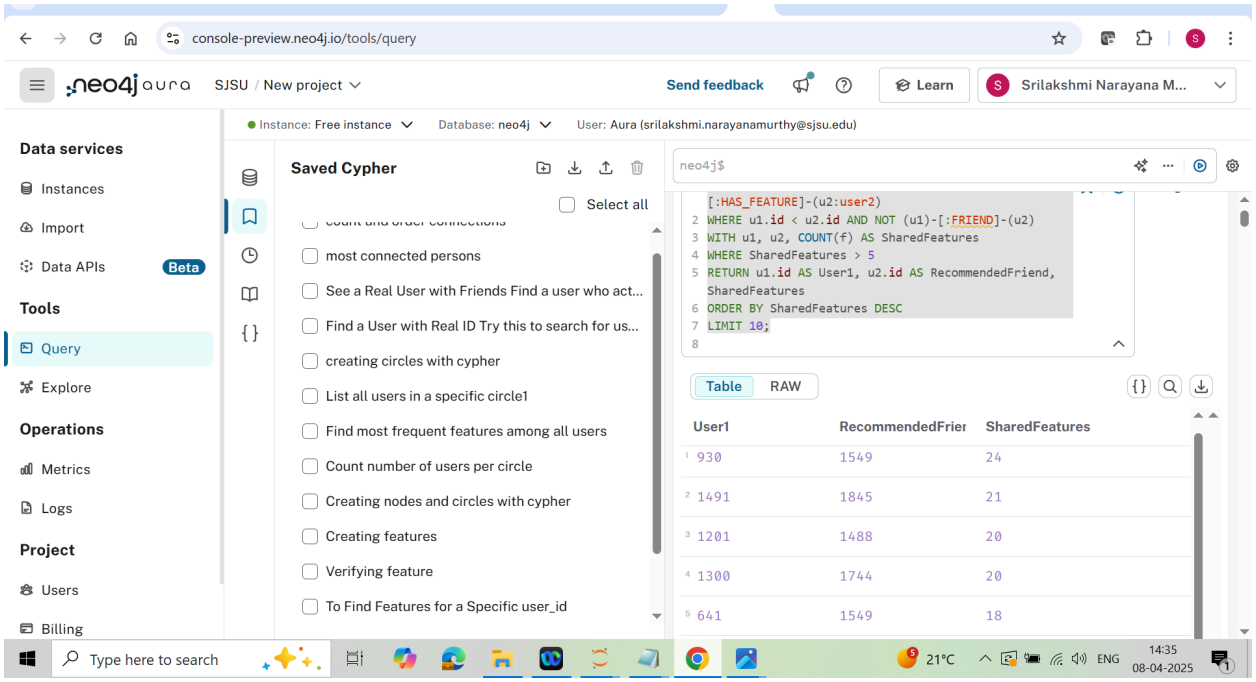
Feature driven friend recommendations

This query recommends friends based on features

```
MATCH (u1:user2)-[:HAS_FEATURE]->(f:Feature)<-[:HAS_FEATURE]-(u2:user2)

WHERE u1.id < u2.id AND NOT (u1)-[:FRIEND]-(u2)
```

```
WITH u1, u2, COUNT(f) AS SharedFeatures
WHERE SharedFeatures > 5
RETURN u1.id AS User1, u2.id AS RecommendedFriend, SharedFeatures
ORDER BY SharedFeatures DESC
LIMIT 10;
```



Find users who appear in more than one circle(Friend who are shared multiple times in different friend list of a given user)

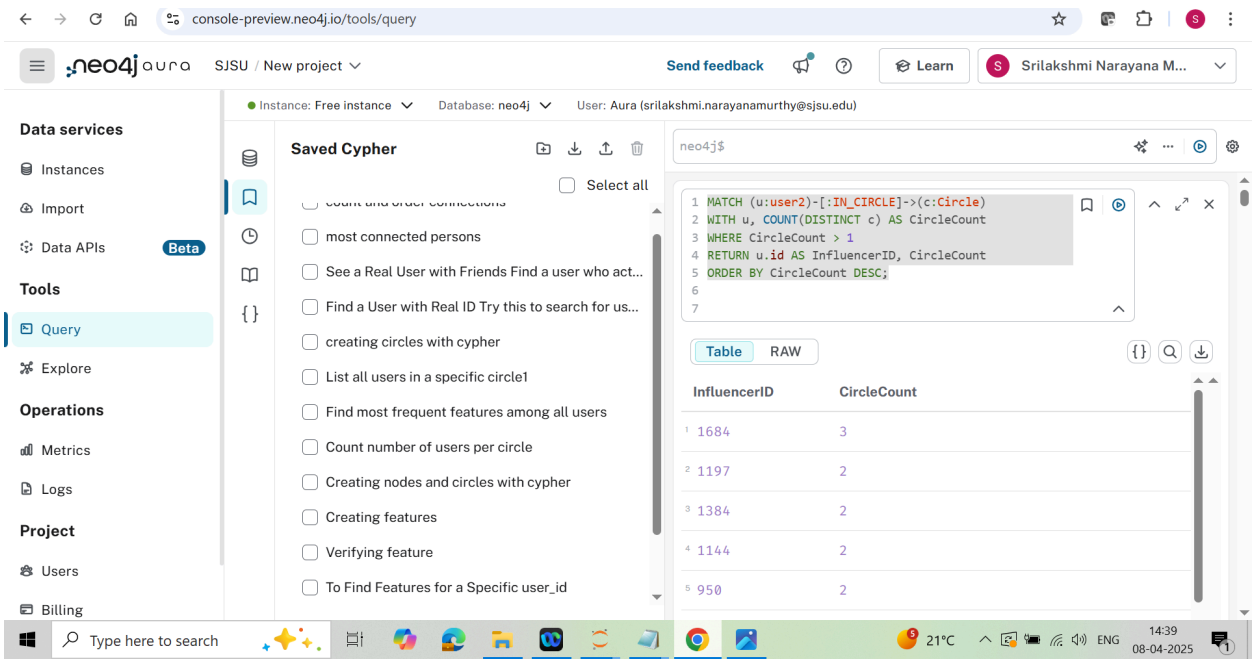
MATCH (u:user2)-[:IN_CIRCLE]->(c:Circle)

WITH u, COUNT(DISTINCT c) AS CircleCount

WHERE CircleCount > 1

RETURN u.id AS InfluencerID, CircleCount

ORDER BY CircleCount DESC;



Top 10 most popular features

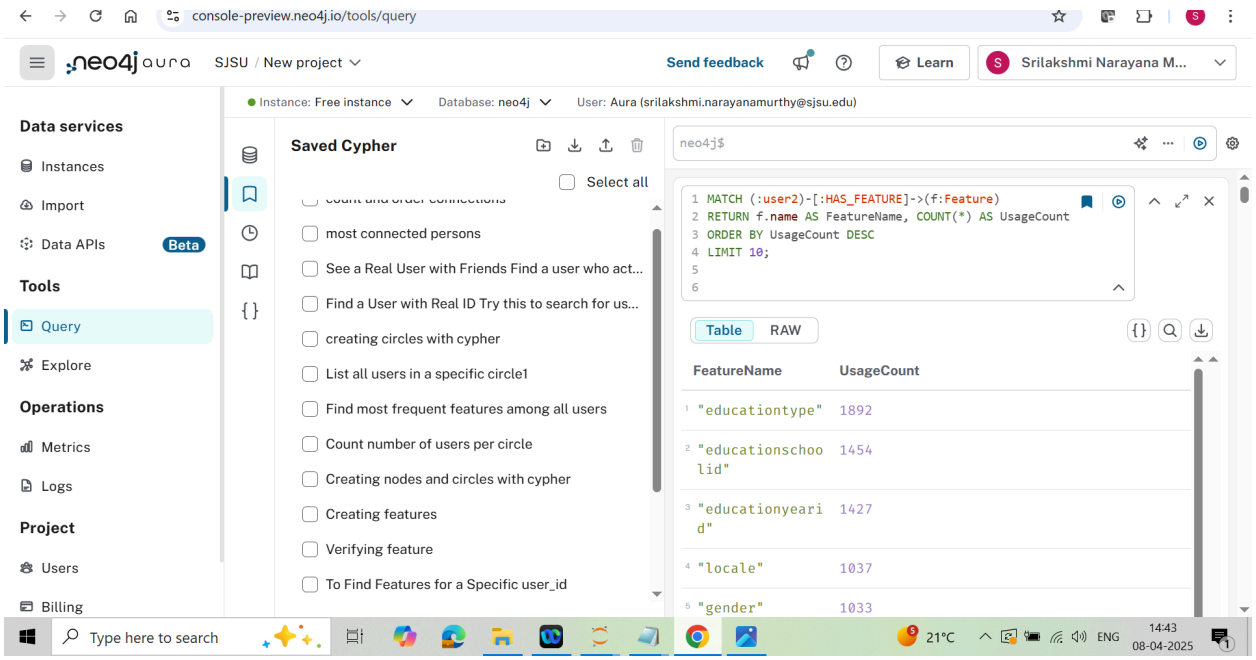
This query gives top most popular features

MATCH (:user2)-[:HAS_FEATURE]->(f:Feature)

RETURN f.name AS FeatureName, COUNT(*) AS UsageCount

ORDER BY UsageCount DESC

LIMIT 10;



Circle Homogeneity Score

Compute how “tight” a circle is in terms of shared features.

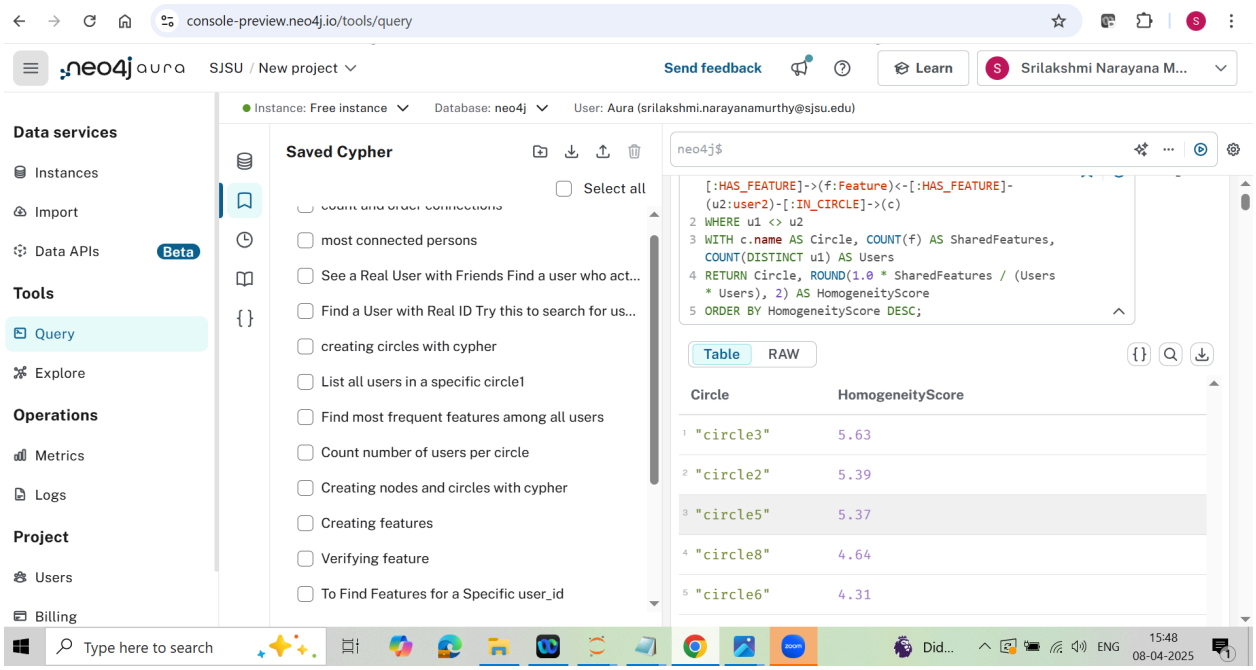
MATCH (c:Circle)<-[:IN_CIRCLE]-(u1:user2)-[:HAS_FEATURE]->(f:Feature)<-[:HAS_FEATURE]-(u2:user2)-[:IN_CIRCLE]->(c)

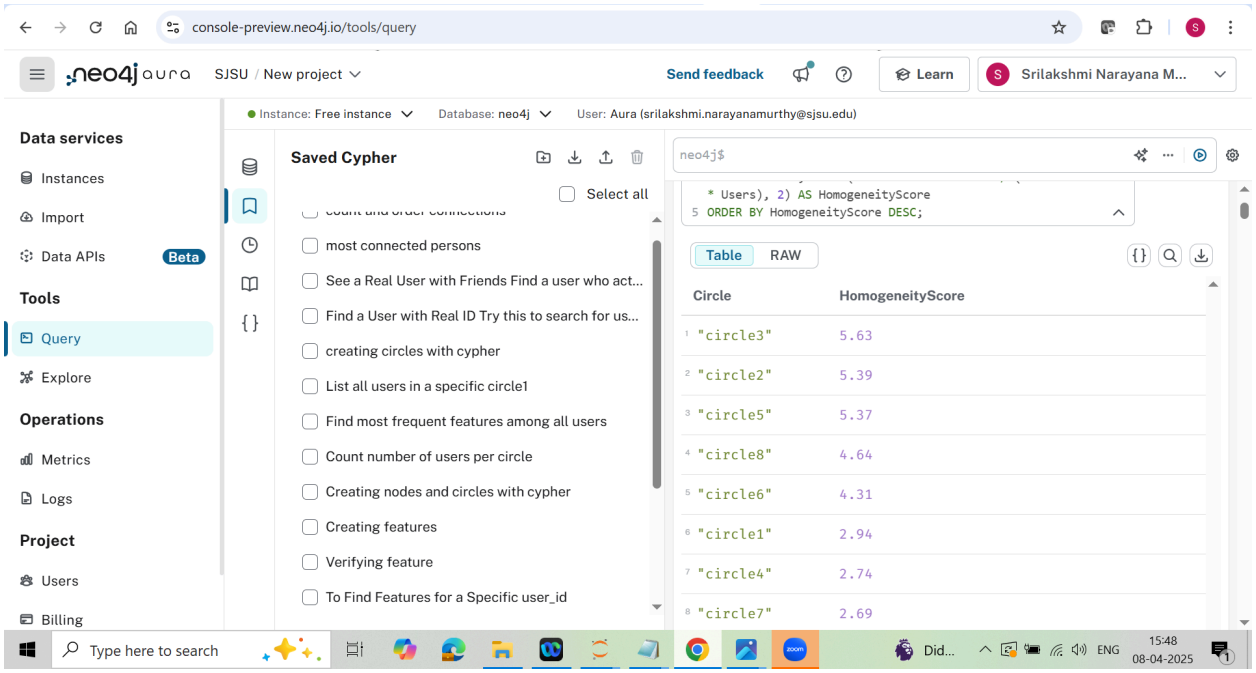
WHERE u1 <> u2

WITH c.name AS Circle, COUNT(f) AS SharedFeatures, COUNT(DISTINCT u1) AS Users

RETURN Circle, ROUND(1.0 * SharedFeatures / (Users * Users), 2) AS HomogeneityScore

ORDER BY HomogeneityScore DESC;

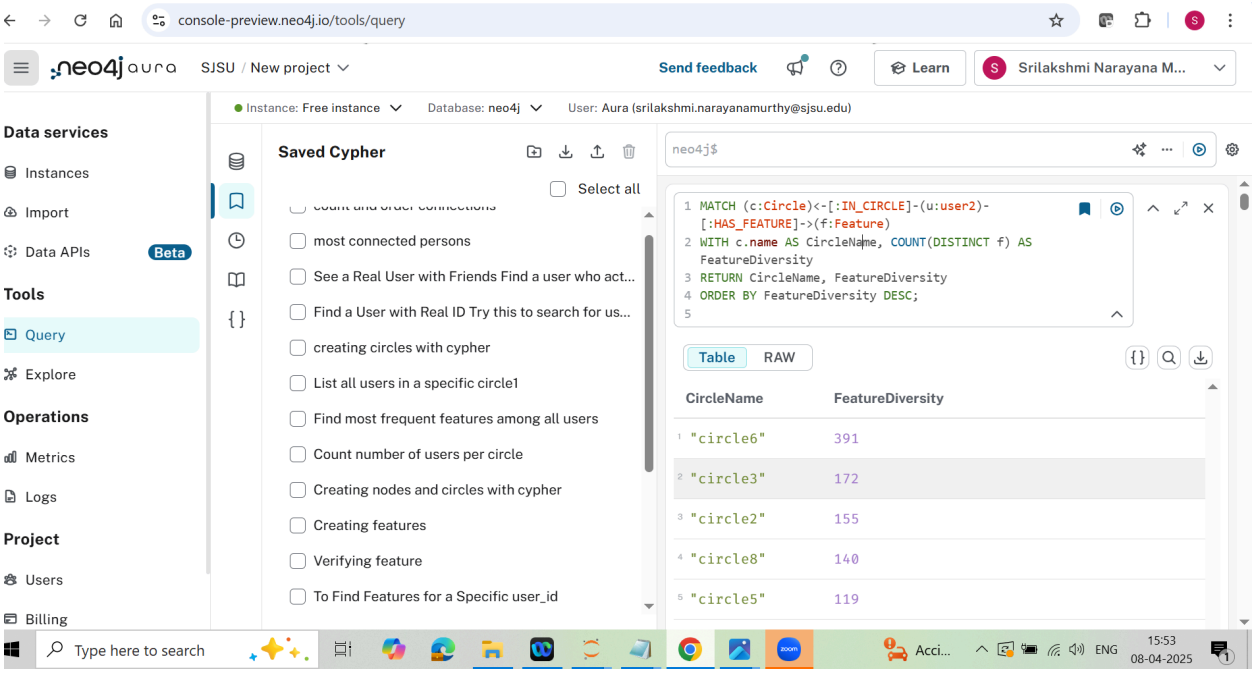


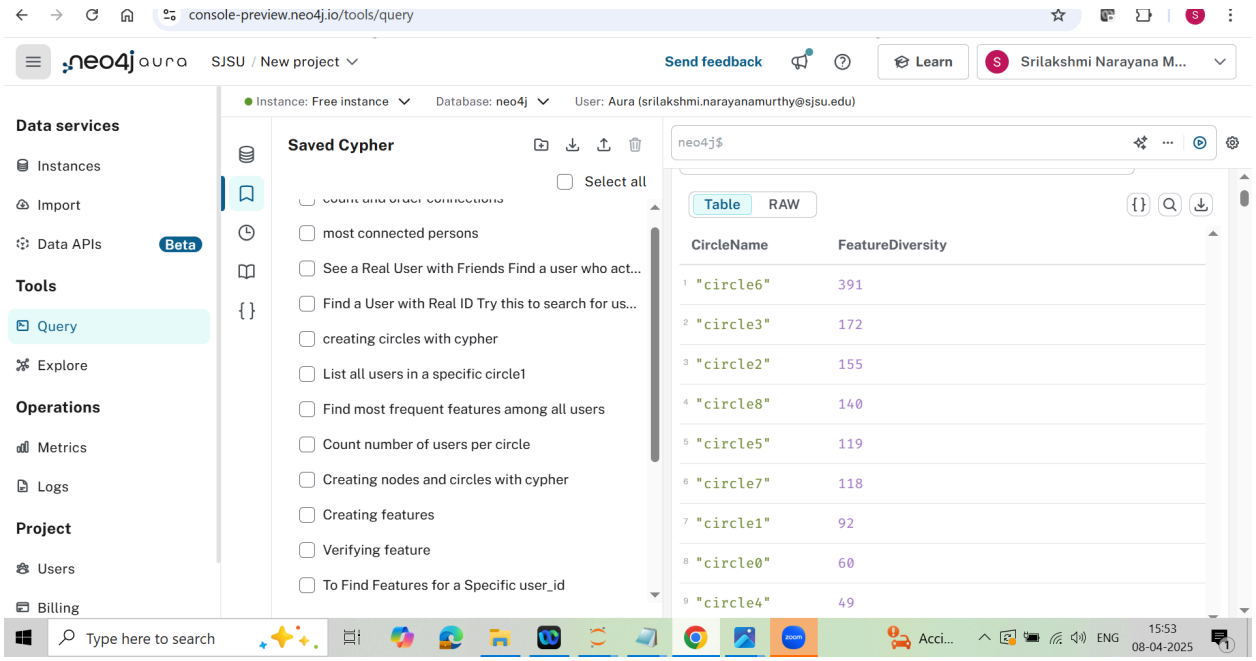


Most Diverse Circle (Users with Most Unique Features)

MATCH (c:Circle)<-[:IN_CIRCLE]-(u:user2)-[:HAS_FEATURE]->(f:Feature)

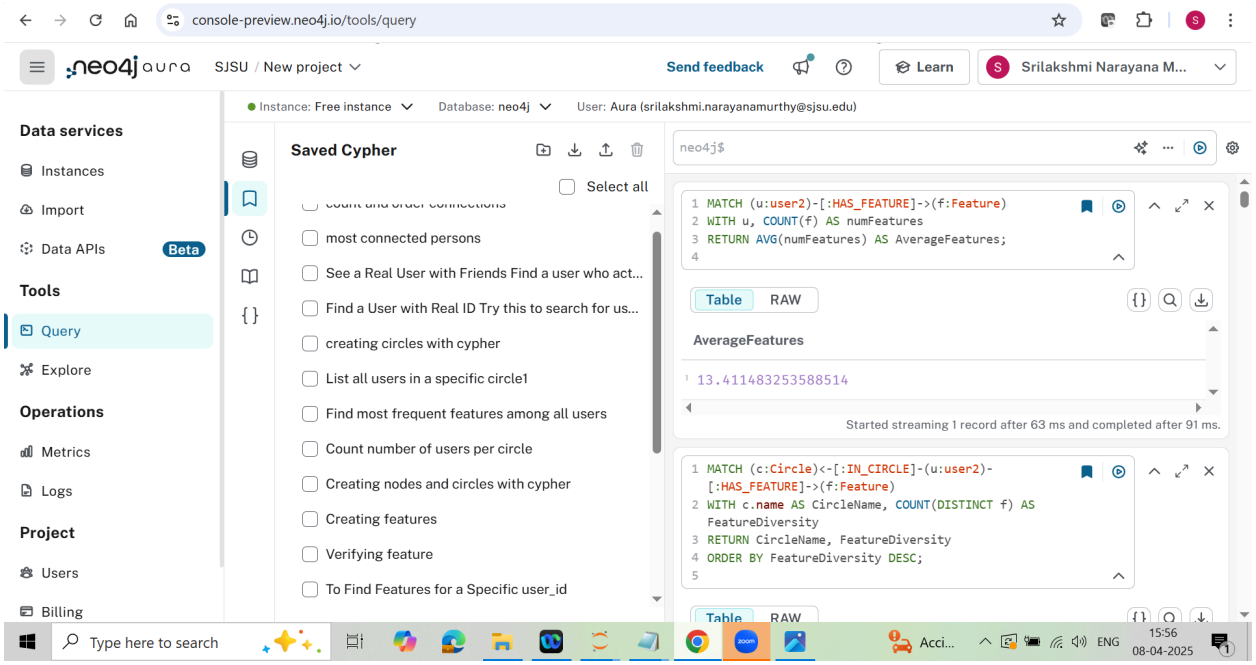
```
WITH c.name AS CircleName, COUNT(DISTINCT f) AS FeatureDiversity
RETURN CircleName, FeatureDiversity
ORDER BY FeatureDiversity DESC;
```





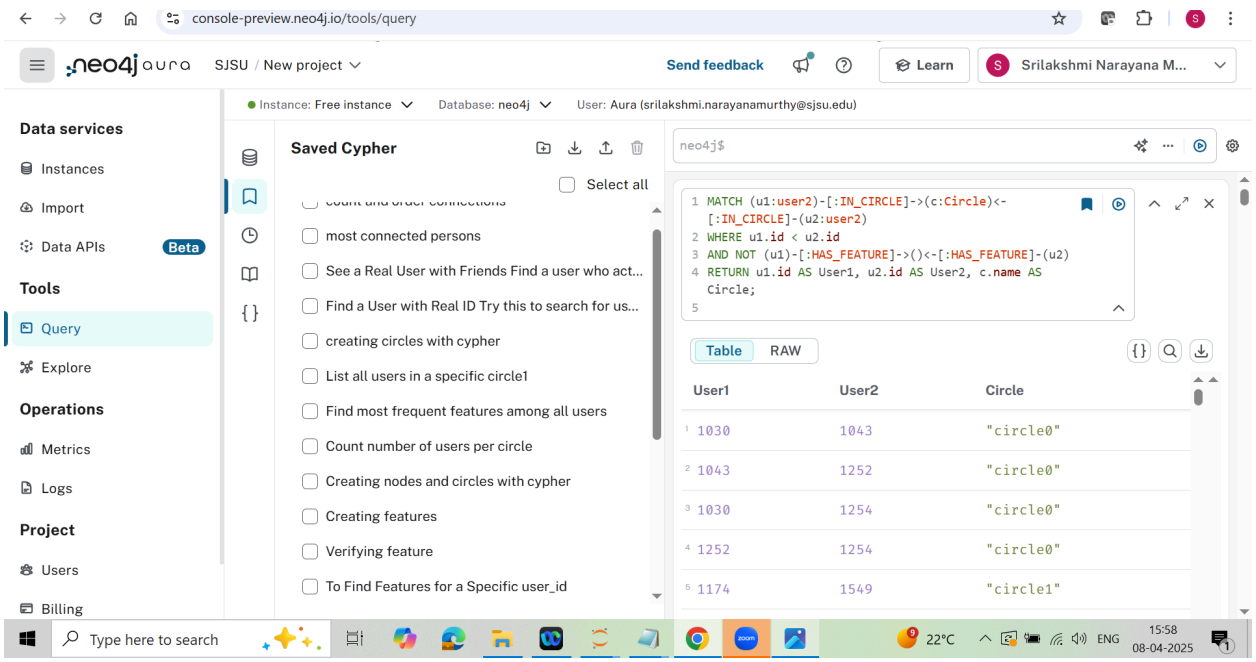
Average Number of Features Per User

```
MATCH (u:user2)-[:HAS_FEATURE]->(f:Feature)
WITH u, COUNT(f) AS numFeatures
RETURN AVG(numFeatures) AS AverageFeatures;
```



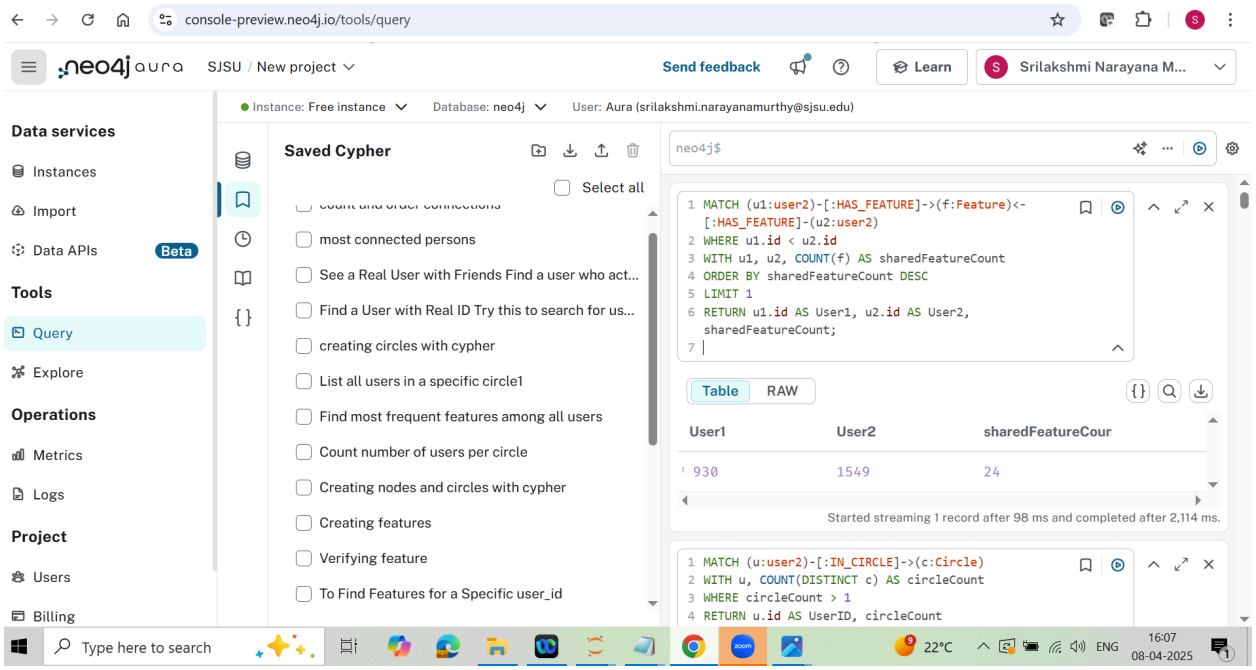
Find Users in Same Circle With No Shared Features

```
MATCH (u1:user2)-[:IN_CIRCLE]->(c:Circle)<-[:IN_CIRCLE]-(u2:user2)
WHERE u1.id < u2.id
AND NOT (u1)-[:HAS_FEATURE]->()<-[:HAS_FEATURE]-(u2)
RETURN u1.id AS User1, u2.id AS User2, c.name AS Circle;
```



Find the Pair of Users Sharing the Most Features

```
MATCH (u1:user2)-[:HAS_FEATURE]->(f:Feature)<-[:HAS_FEATURE]-(u2:user2)
WHERE u1.id < u2.id
WITH u1, u2, COUNT(f) AS sharedFeatureCount
ORDER BY sharedFeatureCount DESC
LIMIT 1
RETURN u1.id AS User1, u2.id AS User2, sharedFeatureCount;
```



HW Blog(Neo4j) :

[Canvasing the world of graph database with Neo4j | by Arpana Cusat | Apr, 2025 | Medium]

Use of LLM(s) :

Neo4j implementation involved using cypher quite extensively. Ex : loading data, validating the loaded data, verifying the same data in the explore to visualize. LLMs helped refining cypher queries where we need queries similar to CTE format in SQL. Also graph dataset were complicated with only “0/1” entries and using LLM we obtained the clarity that all the “0s” that are negative scenarios and wont have representation in Neo4j are actually for usage during Machine learning analysis. Such useful insights about the variety and usage of data helped to have a clearer understanding and better implementation of project.