

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.

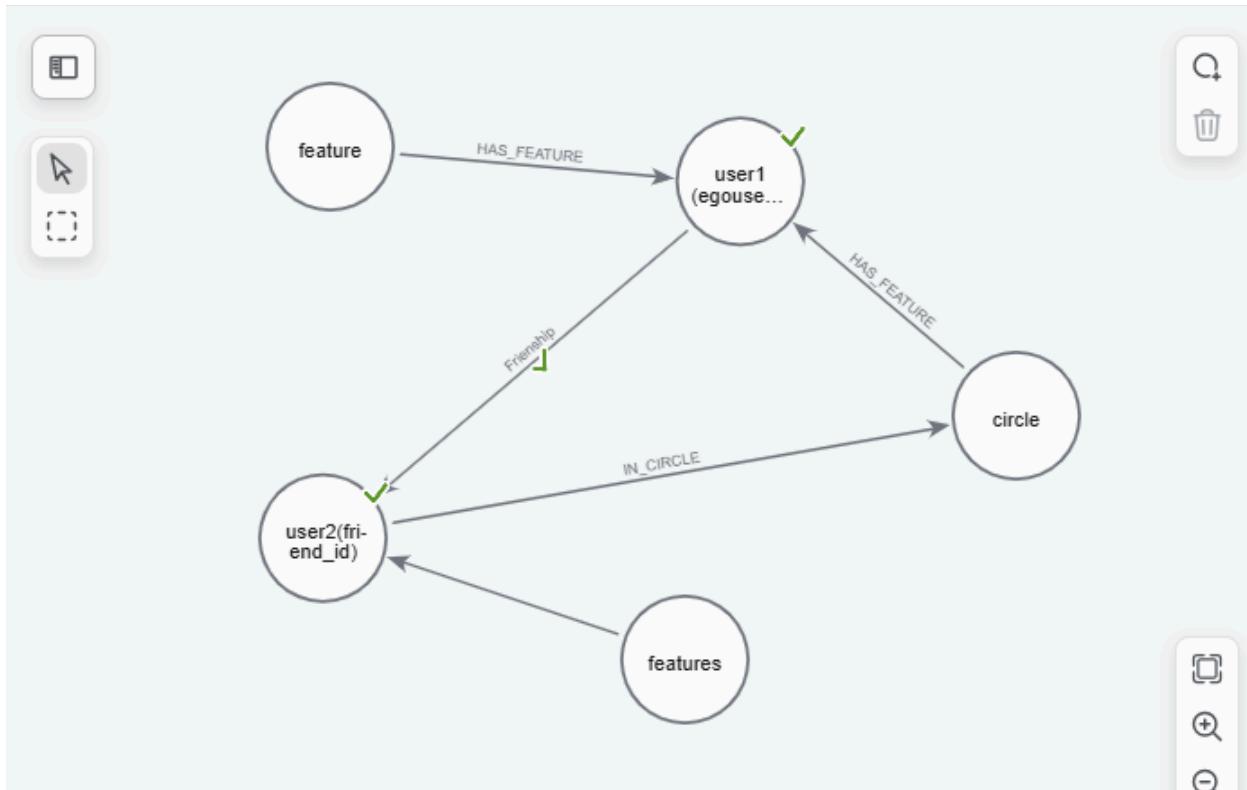
3.Data Modeling

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, which 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 :

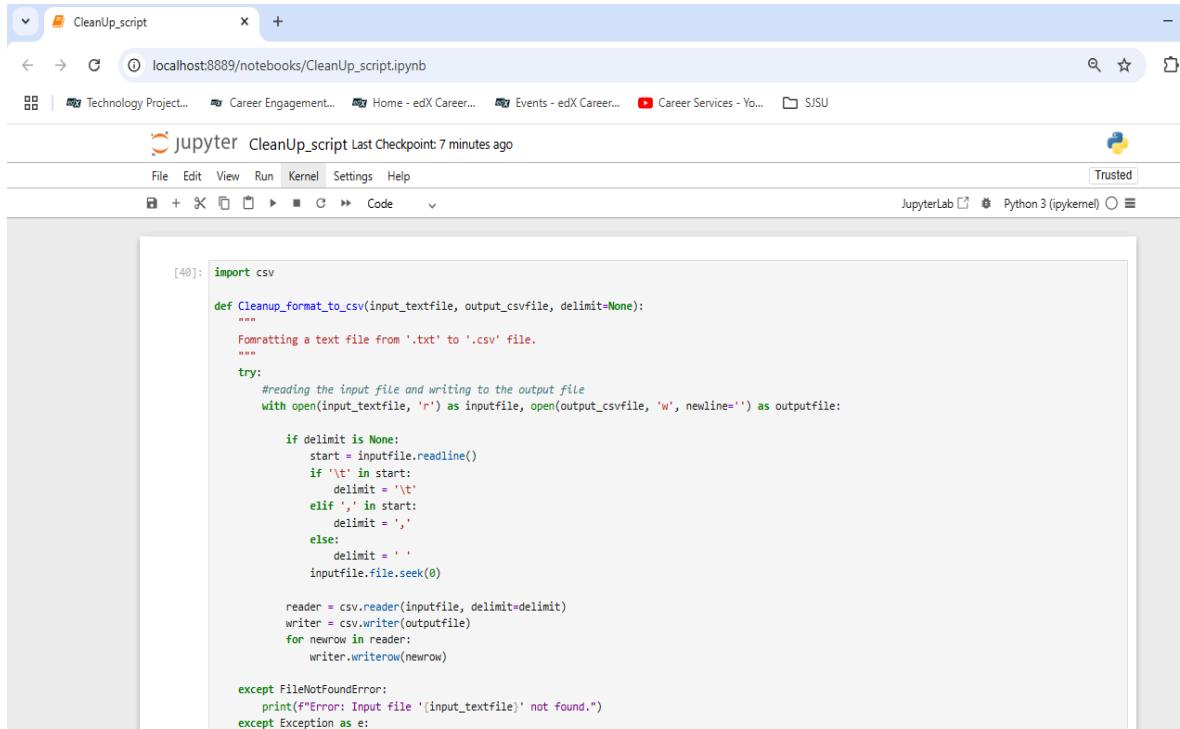
Neo4j 4.0 has a huge update, named Fabric. Fabric comes with its own database that acts as an entry point to the Neo4j environment. A driver will connect to a proxy server or cluster of proxy servers with a set of configuration on it to give it a picture of each shard. There is then a new Cypher USE 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 .



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Shows the title "CleanUp_script" and the URL "localhost:8889/notebooks/CleanUp_script.ipynb".
- Toolbar:** Includes standard Jupyter Notebook icons for file operations, run, kernel, settings, and help.
- Header:** Displays the notebook name "jupyter CleanUp_script Last Checkpoint: 7 minutes ago" and a "Trusted" badge.
- Code Cell:** Contains the following Python code in cell [40]:

```
[40]: import csv
def Cleanup_format_to_csv(input_textfile, output_csvfile, delimit=None):
    """
    Formatting a text file from '.txt' to '.csv' file.
    """
    try:
        #reading the input file and writing to the output file
        with open(input_textfile, 'r') as inputfile, open(output_csvfile, 'w', newline='') as outputfile:

            if delimit is None:
                start = inputfile.readline()
                if '\t' in start:
                    delimit = '\t'
                elif ',' in start:
                    delimit = ','
                else:
                    delimit = ''
                    inputfile.seek(0)

            reader = csv.reader(inputfile, delimiter=delimit)
            writer = csv.writer(outputfile)
            for newrow in reader:
                writer.writerow(newrow)

    except FileNotFoundError:
        print("Error: Input file '{input_textfile}' not found.")
    except Exception as e:
```

```
except FileNotFoundError:
    print(f"Error: Input file '{input_textfile}' not found.")
except Exception as e:
    print(f"An error occurred: {e}")
#Function call
input_textfile = '107.feat.txt'
output_csvfile = '107.feat_csv.csv'
Cleanup_format_to_csv(input_textfile, output_csvfile)
```

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

The screenshot shows an IDE (IntelliJ IDEA) interface with a project structure on the left and a code editor on the right. The project structure includes a 'Test' directory containing an 'utilities' folder with a 'resources' subfolder containing 'input' and 'output' folders. Inside 'input' is a file named '107.feat_csv.csv'. Inside 'output' is a file named '107.feat_csv.csv'. The code editor displays 'feature.py' with the following content:

```
import os

def test():
    feature_dir = os.getcwd() + "/resources"
    input_dir = feature_dir + "/input"
    output_dir = feature_dir + "/output"
    for f in os.listdir(input_dir):
        content = ""
        with open(input_dir + "/" + f, 'r') as file:
            lines = file.readlines()
            for line in lines:
                line = line.replace("anonymized feature ", "")
                content = content + line

        with open(output_dir + "/" + f, "w") as file:
            file.write(content)

def test2():
    """
    input:
    896,0,0,0,0,0,0,
    output:
    "user_id,feat_1,feature2"
    896,0,0,0,0,0,0,
    """
    pass
```

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

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help, and Test - feature.py [utilities]. The title bar indicates the current file is feature.py. The left sidebar displays a project structure with a Test folder containing idea and utilities. The utilities folder contains resources, input (with 107.feat_csv.csv), output (with 107.feat_csv.csv), BeerSong.py, Built-inFunc_ex.py, Container.py, Container_Set.py, diceroll.py, feature.py, Func_ex.py, List_ex.py, Masking_g.py, mergeDict.py, MergeSortedArray.py, MergeSortedArray_ll.py, NameBox.py, odd.py, and odd_ll.py. The main code editor shows the following Python script:

```
feature_dir = os.getcwd() + "/resources"
input_dir = feature_dir + "/input"
output_dir = feature_dir + "/output"
file_name = "107.feat_csv.csv"

with open(input_dir + "/" + file_name, 'r') as file:
    lines = file.readlines()
    feature_count = len(lines[0].split(',')) - 1

    header_str = "user_id"
    for i in range(feature_count):
        feature_col_name = "feature_" + str(i)
        header_str = header_str + "," + feature_col_name

    new_content = header_str + "\n" + ''.join(str(x) for x in lines)

    with open(output_dir + "/" + file_name, "w") as file:
        file.write(new_content)

# test()
test2()
```

-Cypher query to load data into Neo4j Aura :

```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ArpanaSinghSJSU/DataWarehouse/refs/heads/main/facebook_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)
```

a/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/107feat
names_distinct.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

The screenshot shows the Neo4j Aura account profile page. The URL in the browser is `console-preview.neo4j.io/account/profile`. The top navigation bar includes links for 'Send feedback', 'Learn', and a user profile for 'Srilakshmi Narayana M...'. On the left, a sidebar titled 'Account settings' lists 'Profile' (which is selected and highlighted in blue), 'API Keys', and 'Preferences'. The main content area is titled 'Profile' and displays a circular profile picture with a letter 'S' on it. To the right of the picture, the name 'Srilakshmi Narayana Murthy' and email address 'srilakshmi.narayananmurthy@sjtu.edu' are shown.

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

The screenshot shows the neo4j console interface. On the left, a sidebar lists various sections: Instances, Import, Data APIs (Beta), Tools, Query, Explore, Operations, Metrics, Logs, Project, Users, Billing, and Settings. The main area displays a table titled 'Instances' with one row. The table columns are Name, ID, Status, Type, Region, Memo, Storage, CPU, and Actions. The single instance listed is 'Free instance' with ID '93abe0de', status 'RUNNING', and type 'AuraDB Fr...'. Below the table, it says 'Showing 1 of 1 instances'. At the bottom right of the main area, there are 'Show' and '10' dropdowns. The top right of the screen shows a user profile for 'Srilakshmi Narayana M...' and a 'Create instance' button. The top bar also includes 'Send feedback', 'Learn', and a search bar.

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

The screenshot shows the Neo4j Aura console interface. On the left, a sidebar menu includes 'Data services', 'Tools', 'Operations', 'Project', and 'Billing'. The 'Import' option is selected and highlighted in blue. The main workspace displays a graph model with a single node labeled 'user2'. A legend indicates that a green arrow represents the 'friends' relationship. To the right of the graph, there are sections for 'Definition' (Label, Table, Properties) and 'Constraints & Indexes (2)'. The 'Label' section shows 'Name: user1'. The 'Table' section shows 'Name: facebook_combined_csv.csv' and has a 'Filter table' toggle switch. The 'Properties' section has a 'Map from table' button. At the top right, there are buttons for 'Send feedback', 'Learn', 'Run import', and more. The top bar also shows the instance type ('Free instance'), database ('neo4j'), and user ('Srilakshmi Narayana M...'). The bottom of the screen features a taskbar with various icons and a system status bar showing the date (08-04-2025), time (12:41), and weather (19°C).

Creating User1 Node and mapping the user1 node with imported dataset

The screenshot shows the Neo4j Aura interface for importing data. On the left, a sidebar lists 'Data services', 'Tools' (with 'Import' selected), and 'Operations'. The main area displays a graph model with two nodes, 'user1' and 'user2', connected by a 'Friends' relationship. The 'Definition' section shows a node named 'user1'. The 'Table' section lists 'facebook_combined_csv.csv' twice. The 'Properties' section includes fields for Name, Type, Column, and ID. A 'Map from table' dialog is open, showing a mapping for 'user1' with 'user1' checked and 'user2' unchecked. The status bar at the bottom indicates it's 19°C, 12:43, and the date is 08-04-2025.

This screenshot is identical to the one above, but the 'Map from table' dialog is now open for the 'user2' node. It shows 'user1' checked and 'user2' unchecked. The 'Column' dropdown is set to 'user1'. The status bar at the bottom indicates it's 19°C, 12:45, and the date is 08-04-2025.

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 ⓘ S Srilakshmi Narayana M...

Data services

- Instances
- Import **Beta**

Tools

- Query
- Explore

Operations

- Metrics
- Logs

Project

- Users

Billing

Type here to search

Graph models / Untitled model 8

Instance: Free instance Database: neo4j User: Aura (srilakshmi.narayananamurthy@sjsu.edu)

Generate model Run import ...

Definition Constraints & Indexes (2)

user2

Table ⓘ

Name facebook_combined_csv.csv

Filter table

Properties Map from table +

Name	Type	Column	ID
user2	integer	user2	key

19°C ENG 12:46 08-04-2025

```
graph LR; user1((user1)) -- Friends --> user2((user2))
```

← → ⌂ ⌂ console-preview.neo4j.io/tools/import/model/67f6fe32-2d81-42c9-889f-fe34e42b5e56

neo4j aura SJSU / New project ▾ Send feedback 📢 ⓘ Learn ⓘ S Srilakshmi Narayana M...

Data services

- Instances
- Import **Beta**

Tools

- Query
- Explore

Operations

- Metrics
- Logs

Project

- Users

Billing

Type here to search

Graph models / Untitled model 8

Instance: Free instance Database: neo4j User: Aura (srilakshmi.narayananamurthy@sjsu.edu)

Generate model Run import ...

Definition Constraints & Indexes (2)

user2

Table ⓘ

Name facebook_combined_csv.csv

Map from table Select all

user1	0
user2	1

Cancel Confirm

Column ID

user2 key

19°C ENG 12:46 08-04-2025

```
graph LR; user1((user1)) -- Friends --> user2((user2))
```

Creating relationship FRIENDS between User1 and User2

The screenshot shows the neo4j Aura interface for importing data. On the left, a sidebar includes sections for Data services, Tools (with Import selected), Operations, Project, and Billing. The main area displays a graph model with two nodes, 'user1' and 'user2', connected by a blue arrow labeled 'Friends'. To the right, configuration panels show the Relationship type set to 'Friends' and the Table set to 'facebook_combined_csv.csv'. Below these are sections for Node ID mapping and a summary of the import process.

Importing from csv file

The screenshot shows the neo4j Aura interface after a successful import job. The main area displays a summary of the import results for two users: 'user1' and 'user2'. For user1, the summary includes: Time taken 00:00:01, File size 920.5 KiB, File rows 88,234, Nodes created 3,663, Properties set 3,663, Labels added 3,663, and Query time 00:00:01. A green box indicates 'Run import completed' successfully. A 'Show Cypher' button is available for each user entry. The sidebar on the left remains the same as in the previous screenshot.

neo4j aura

Total time: 00:00:08

Run import completed
Import completed successfully.

Time taken	File size	File rows	Nodes created	Properties set	Labels added	Query time
00:00:00	920.5 KiB	88,234	4,037	4,037	4,037	00:00:00

Friends facebook_combined_csv.csv Show Cypher

Time taken	File size	File rows	Relationships created	Properties set	Query time
00:00:05	920.5 KiB	88,234	88,234	0	00:00:05

Close Explore results

Type here to search

UNH 13:01 08-04-2025

COLOR REPRESENTATIONS

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

neo4j aura SJSU / New project

Send feedback Learn Srilakshmi Narayana M...

Default Perspective Export

Instance: Free instance Database: neo4j User: Aura (srilakshmi.narayananamurthy@sjsu.edu)

Default Perspective

user1 — (any) — user2

All (10477) Selected (0) Force-based layout

Circle 9 Feature 15 user1 1001 user2 2017

19°C 12:51 08-04-2025

Type here to search

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_CIRCLE relation has been created between user and circle

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

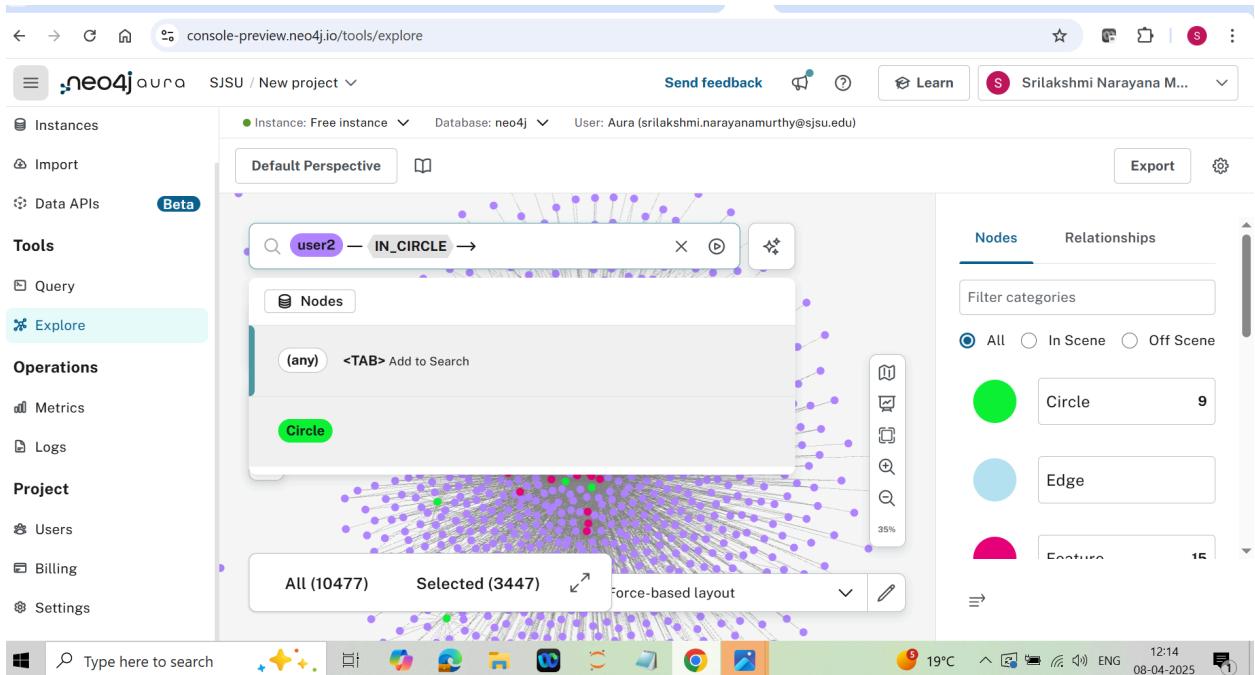
The screenshot shows the Neo4j Aura web interface. On the left, there's a sidebar with sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore), Operations (Metrics, Logs), and Project (Users). The 'Query' tab is selected. In the main area, under 'Saved Cypher', there are several pre-defined queries with checkboxes. One of these, 'Creating circles with cypher', is checked. To its right, the Cypher editor contains the following code:

```
1 LOAD CSV WITH HEADERS FROM
  'https://raw.githubusercontent.com/ArpanaSinghSJSU/DataWarehouse/refs/heads/main/facebook_107circles_reformatted.csv' AS row
2 MERGE (ego:user1 {id: 107})
3 MERGE (u:user2 {id: toInteger(row.friend_id)})
4 MERGE (c:Circle {name: row.circle_name, owner: 107})
5 MERGE (u)-[:IN_CIRCLE]->(c)
6
```

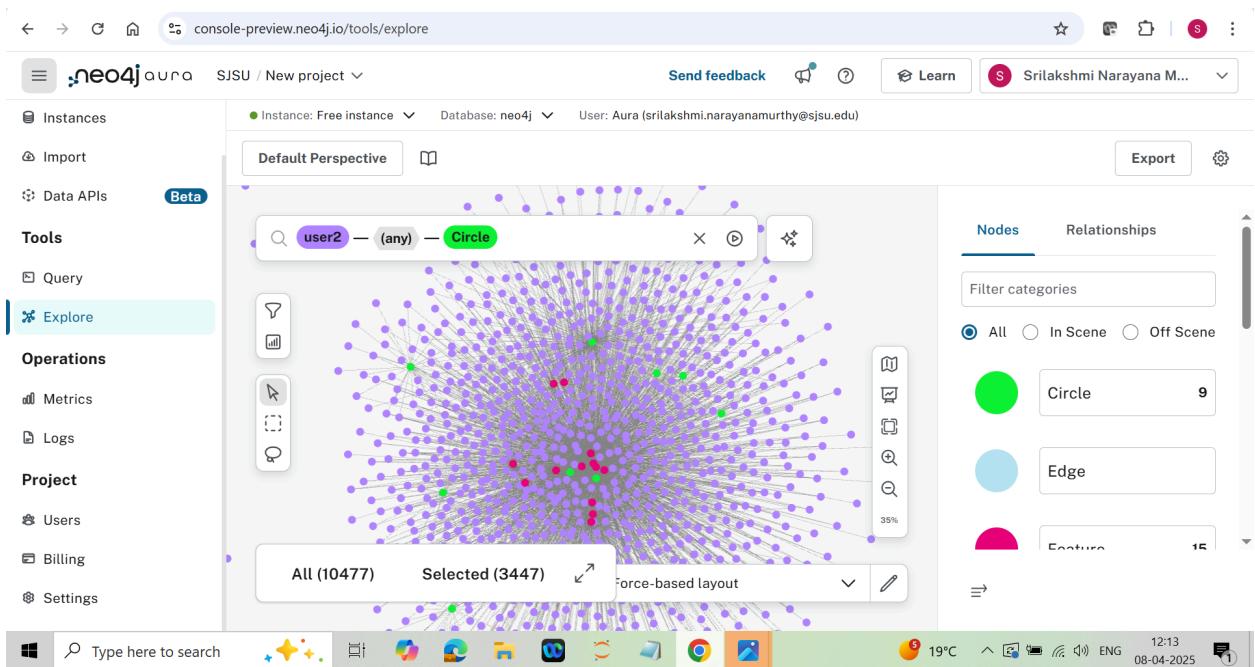
Below the code, a message indicates: 'Created 491 nodes, created 501 relationships, set 500 properties, added 491 labels' and 'Completed after 1,864 ms'. There are also other query logs listed below, such as 'Deleted 9331 nodes, deleted 100551 relationships' and 'MATCH (c:Circle {name: "circle5"})<-[::IN_CIRCLE]->(c)'.

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



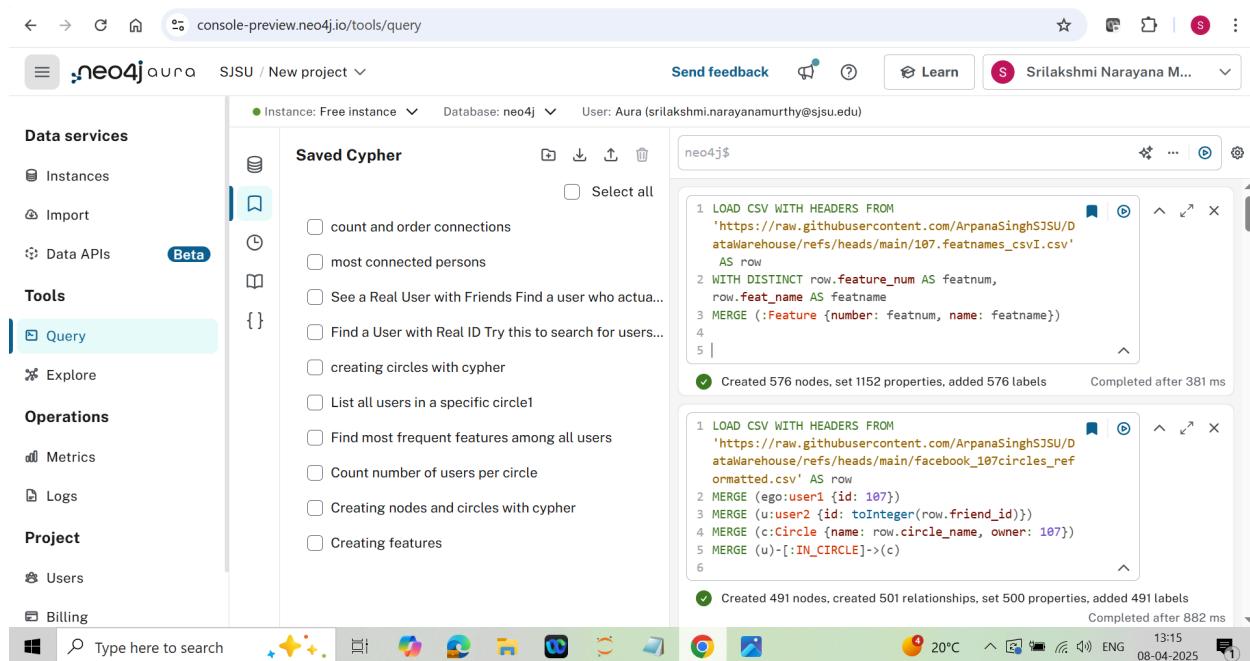
Creating Features

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_csv.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 Beta

Send feedback 📣 ⓘ Learn S Srilakshmi Narayana M...

Data services

- Instances
- Import
- Data APIs

Tools

- Query Selected
- Explore

Operations

- Metrics
- Logs

Project

- Users
- Billing

Type here to search

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_dat e"
7 "166"	"workstart_dat e"
8 "200"	"workstart_dat e"
9 "725"	"workstart_dat e"

20°C 20°C ENG 13:18 08-04-2025

This screenshot shows the neo4j Aura interface with the 'Query' tool selected. A Cypher query is run against a free instance, returning four rows of data. The columns are 'feature_num' and 'feature_name'. The data includes rows for feature numbers 165, 166, 200, and 725, all associated with the name 'workstart_dat e'.

← → ⌂ ⌂ console-preview.neo4j.io/tools/query

neo4j aura SJSU / New project Beta

Send feedback 📣 ⓘ Learn S Srilakshmi Narayana M...

Data services

- Instances
- Import
- Data APIs

Tools

- Query Selected
- Explore

Operations

- Metrics
- Logs

Project

- Users
- Billing

Type here to search

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_dat e"
2 "723"	"workstart_dat e"
3 "724"	"workstart_dat e"
4 "164"	"workstart_dat e"

20°C 20°C ENG 13:18 08-04-2025

This screenshot shows the neo4j Aura interface with the 'Query' tool selected. A Cypher query is run against a free instance, returning four rows of data. The columns are 'feature_num' and 'feature_name'. The data includes rows for feature numbers 162, 723, 724, and 164, all associated with the name 'workstart_dat e'.

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)
```

The screenshot shows the Neo4j Aura console interface. On the left, there's a sidebar with sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore), Operations (Metrics, Logs), Project (Users, Billing), and a search bar. The main area has tabs for 'Saved Cypher' and 'neo4j\$'. The 'neo4j\$' tab contains the Cypher query shown above. Below the query, the results are displayed: 'Created 564 nodes, created 14015 relationships, set 564 properties, added 564 labels Completed after 7,996 ms'. A note indicates that the execution plan contains an Eager operator. At the bottom, there's another Cypher snippet: '1 MATCH (f:Feature) 2 RETURN f.number AS feature_num, f.name AS feature_name 3 LIMIT 20 4'. The status bar at the bottom right shows the date (08-04-2025), time (13:23), and system information (20°C, ENG).

Features are represented in pink

The screenshot shows the Neo4j Explore interface. On the left, a sidebar menu includes sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore, highlighted in green), Operations (Metrics, Logs), Project (Users, Billing), and a search bar. The main area displays a network graph with a central node labeled "user2" connected to many other nodes. A search bar at the top of the graph area shows the query "user2 - (any) - Feature". A warning message indicates "1000 nodes found! The node query limit has been reached." To the right of the graph, there are tabs for Nodes and Relationships, and a sidebar for filtering categories: All (selected), In Scene, and Off Scene. Under Nodes, there are three categories: Circle (9 nodes), Edge (0 nodes), and Feature (101 nodes). The Feature category is highlighted in pink, matching the color of the nodes in the graph. The bottom of the screen shows a taskbar with various icons and system status information.

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
```

The screenshot shows the neo4j console interface. On the left, there's a sidebar with various tools like Instances, Import, Data APIs, Tools (with Query selected), Explore, Operations, Project, Users, Billing, and Settings. The main area has tabs for 'Saved Cypher' and 'neo4j\$'. The 'neo4j\$' tab contains the Cypher query shown above. Below the query, there's a 'Table' tab which displays the results in a table format:

user_id
1 1410
2 1144
3 950
4 1385
5 1174
6 1087

The status bar at the bottom shows the date and time as 11:51 08-04-2025.

The screenshot shows the neo4j aura web interface. On the left, there's a sidebar with various tools like Instances, Import, Data APIs, Tools (with a Beta button), Query (which is selected and highlighted in blue), Explore, Operations, Metrics, Logs, Project, Users, Billing, and Settings. The main area is titled "Saved Cypher" and contains a list of 16 queries. The queries are numbered 7 through 22 and include:

- 7 563
- 8 1684
- 9 1676
- 10 1555
- 11 1549
- 12 389
- 13 1013
- 14 1251
- 15 1031
- 16 1421

At the bottom right of the main area, it says "Started streaming 16 records after 57 ms and completed after 58 ms." The status bar at the bottom shows system information: UNH, 11:52, 08-04-2025.

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
```

This screenshot shows the same neo4j aura interface as the previous one, but now the results of the executed query are displayed. The results are presented in a table format:

circle	member_count
1 "circle0"	10
2 "circle1"	16
3 "circle2"	19
4 "circle3"	39
5 "circle4"	10
6 "circle5"	37
7 "circle6"	308

The status bar at the bottom shows system information: 18°C, 11:38, 08-04-2025.

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
```

The screenshot shows the neo4j console interface. On the left, there's a sidebar with sections for Data services, Tools (with 'Query' selected), Operations, Project, and Billing. A search bar at the bottom left contains the placeholder 'Type here to search'. The main area has tabs for 'Table' (selected) and 'RAW'. The query 'MATCH (:user2)-[:HAS_FEATURE]->(f:Feature)' is entered in the RAW tab. The results table shows the following data:

feature	frequency
"educationtype"	1892
"educationschoo lid"	1454
"educationyeari d"	1427
"locale"	1037
"gender"	1033
"workemployeri d"	918

The screenshot shows the neo4j Aura web interface. On the left, there's a sidebar with sections for Data services, Tools (Query selected), and Operations. The main area has a 'Saved Cypher' section with a list of pre-defined queries. Below it is a code editor window with the following Cypher query:

```
neo4j$ MATCH (:user2)-[:HAS_FEATURE]->(f:Feature)
```

The results table shows the following data:

feature	frequency
"educationyearid"	1427
"locale"	1037
"gender"	1033
"workemployerid"	918
"locationid"	806
"worklocationid"	778
"workstart_date"	715

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
```

The screenshot shows the neo4j Aura web interface. The sidebar is identical to the previous one. The main area displays the specific Cypher query:

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

The results table shows the following data:

f.name
"birthday"
"educationconcentrationid"
"educationschoolid"
"educationschoolid"
"educationtype"
"educationtype"

The screenshot shows the neo4j Aura interface. On the left, there's a sidebar with sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore, Metrics, Logs), Operations (Metrics, Logs), Project (Users, Billing), and a search bar at the bottom. The main area has tabs for Data services and Tools, with 'Query' selected. A 'Beta' badge is visible next to the Tools tab. The central workspace shows a 'Saved Cypher' section with a list of recent queries and a text editor containing a Cypher query:

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

The results pane below shows the names of features from the query:

Table	RAW
f.name	
5	"educationschoolid"
6	"educationtype"
7	"educationtype"
8	"educationtype"
9	"educationyearid"
10	"educationyearid"
11	"gender"

At the bottom, there's a taskbar with icons for Windows, Start, Task View, File Explorer, Edge, File, Task Manager, and a system status bar showing the date, time, battery level, and network connection.

The screenshot shows the Neo4j Aura interface. On the left, there's a sidebar with sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore, Metrics, Logs), Operations (Metrics, Logs), Project (Users, Billing), and a bottom section for console-preview.neo4j.io/tools/query. The main area has tabs for 'Saved Cypher' and 'Recent'. The 'Saved Cypher' tab is active, displaying a list of pre-defined Cypher queries with checkboxes next to them. One query is selected and expanded, showing the Cypher code:

```
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
```

Below the code, there are two tabs: 'Table' and 'RAW'. The 'Table' tab is selected, showing a table with three columns: User1, User2, and SharedFeatures. The data is as follows:

User1	User2	SharedFeatures
930	1549	24
1491	1845	21
1300	1744	20
1201	1488	20

The screenshot shows the neo4j aura web interface. On the left, there's a sidebar with sections for Data services, Tools (Query selected), Operations, Project, and Billing. The main area has a 'Saved Cypher' section with a list of pre-defined queries. Below it is a query editor window titled 'neo4j\$' containing a table with three columns: User1, User2, and SharedFeatures. The table lists 10 rows of data. At the bottom, there's a search bar and a toolbar with various icons.

User1	User2	SharedFeatures
930	1549	24
1491	1845	21
1300	1744	20
1201	1488	20
921	1491	18
1645	1744	18
1305	1491	18
1549	1625	18
641	1549	18

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;
```

The screenshot shows the neo4j aura web interface. The sidebar and layout are identical to the first screenshot. The query editor window now contains the Cypher code from the previous block. Below it is a table with three columns: Circle, Feature, and Usage. The table lists 5 rows of data. The interface includes a search bar and a toolbar at the bottom.

Circle	Feature	Usage
"circle0"	"educationtype"	16
"circle0"	"workstart_dae"	13
"circle0"	"educationyearid"	11
"circle0"	"gender"	10
"circle0"	"locale"	10

The screenshot shows the Neo4j Aura interface with a sidebar containing 'Data services' (Instances, Import, Data APIs), 'Tools' (Query, Explore, Operations, Project, Users, Billing), and a 'Saved Cypher' section with various saved queries. The main area displays a Cypher query and its results:

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

Circle	Feature	Usage
circle1	gender	16
circle1	locale	16
circle1	workemployerid	16
circle1	worklocationid	11
circle1	locationid	10

The screenshot shows the Neo4j Aura interface with a sidebar containing 'Data services' (Instances, Import, Data APIs), 'Tools' (Query, Explore, Operations, Project, Users, Billing), and a 'Saved Cypher' section with various saved queries. The main area displays a Cypher query and its results:

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

Circle	Feature	Usage
circle2	locationid	21
circle2	workemployerid	21
circle2	worklocationid	20
circle2	gender	19

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;

```

The screenshot shows the Neo4j Aura interface. On the left, there's a sidebar with sections for Data services, Tools (selected), Operations, Project, and Billing. The Tools section has a sub-section for Query, which is highlighted with a teal background. In the main area, there's a 'Saved Cypher' section with a list of pre-defined queries like 'most connected persons' and 'Find a User with Real ID'. Below this is a code editor window titled 'neo4j\$' containing the Cypher query from the question. To the right of the code editor is a table result set with columns 'User1', 'RecommendedFrier', and 'SharedFeatures'. The table contains five rows of data. At the bottom of the interface, there's a search bar, a toolbar with various icons, and a system tray showing the date and time.

User1	RecommendedFrier	SharedFeatures
930	1549	24
1491	1845	21
1201	1488	20
1300	1744	20
641	1549	18

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;
```

The screenshot shows the neo4j Aura web interface. On the left, there's a sidebar with sections for Data services, Tools (Query is selected), and Operations. The main area has a 'Saved Cypher' section with a list of saved queries and a 'neo4j\$' query editor containing the Cypher code from above. Below the editor is a table showing the results of the query:

InfluencerID	CircleCount
1684	3
1197	2
1384	2
1144	2
950	2

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;
```

The screenshot shows the neo4j console interface. On the left, there's a sidebar with sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore, Operations, Metrics, Logs, Project, Users, Billing), and a search bar. The main area has tabs for Saved Cypher and Query. A query is being run in the Query tab:

```

1 MATCH (:user2)-[:HAS_FEATURE]->(f:Feature)
2 RETURN f.name AS FeatureName, COUNT(*) AS UsageCount
3 ORDER BY UsageCount DESC
4 LIMIT 10;
5
6

```

The results are displayed in a table:

FeatureName	UsageCount
educationtype*	1892
educationschoo lid*	1454
educationyeari d*	1427
locale*	1037
gender*	1033

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;

```

The screenshot shows the neo4j Aura interface. On the left, there's a sidebar with sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore), Operations (Metrics, Logs), Project (Users, Billing), and a search bar. The main area has tabs for 'Saved Cypher' and 'Recent'. A query editor window is open with the following Cypher code:

```
[: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 u2) AS Users
RETURN Circle, ROUND(1.0 * SharedFeatures / (Users * Users), 2) AS HomogeneityScore
ORDER BY HomogeneityScore DESC;
```

The results table shows the following data:

Circle	HomogeneityScore
1 "circle3"	5.63
2 "circle2"	5.39
3 "circle5"	5.37
4 "circle8"	4.64
5 "circle6"	4.31

This screenshot shows the same neo4j Aura interface. The sidebar and recent queries are identical. The query editor window now contains this Cypher code:

```
* Users), 2) AS HomogeneityScore
5 ORDER BY HomogeneityScore DESC;
```

The results table shows the following data:

Circle	HomogeneityScore
1 "circle3"	5.63
2 "circle2"	5.39
3 "circle5"	5.37
4 "circle8"	4.64
5 "circle6"	4.31
6 "circle1"	2.94
7 "circle4"	2.74
8 "circle7"	2.69

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;

The screenshot shows the neo4j console interface. On the left, there's a sidebar with sections for Data services, Tools (Query is selected), and Operations. The main area has a 'Saved Cypher' section with a list of saved queries and a text input field containing the Cypher code. Below it is a table showing the results of the executed query.

Saved Cypher:

```
1 MATCH (c:Circle)-[:IN_CIRCLE]->(u:user2)-[:HAS_FEATURE]->(f:Feature)
2 WITH c.name AS CircleName, COUNT(DISTINCT f) AS FeatureDiversity
3 RETURN CircleName, FeatureDiversity
4 ORDER BY FeatureDiversity DESC;
5
```

Table:

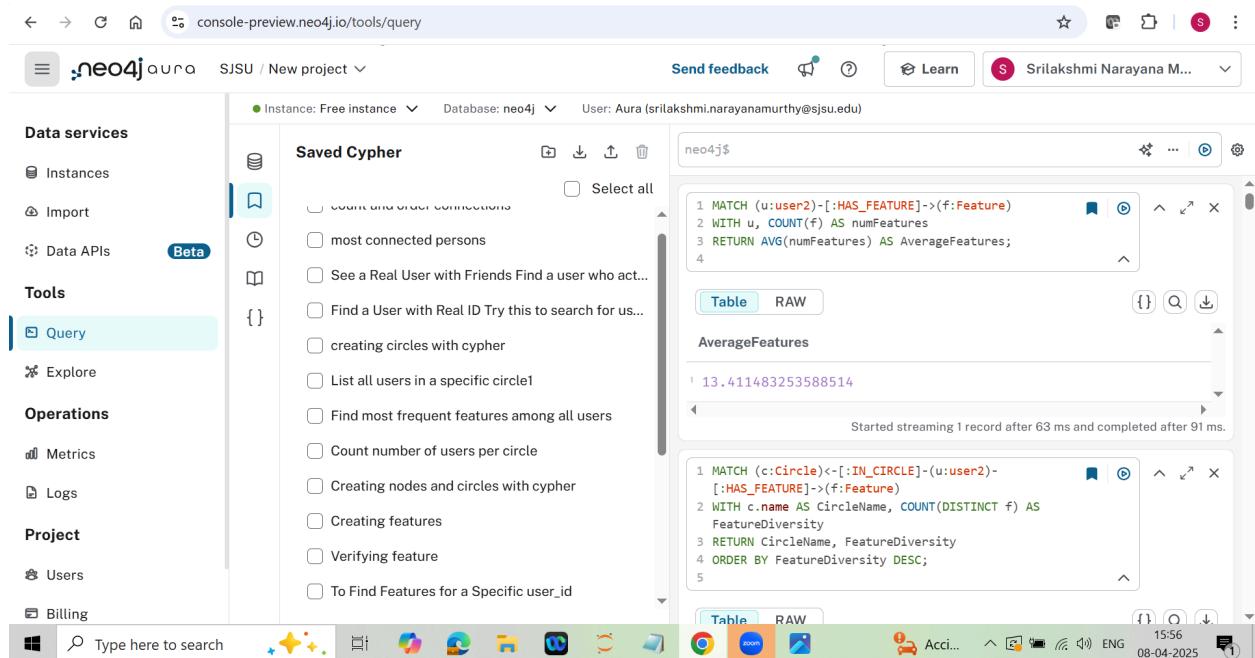
CircleName	FeatureDiversity
circle6	391
circle3	172
circle2	155
circle8	140
circle5	119

This screenshot is identical to the one above, showing the same Cypher code and results table. The table data is as follows:

CircleName	FeatureDiversity
circle6	391
circle3	172
circle2	155
circle8	140
circle5	119
circle7	118
circle1	92
circle0	60
circle4	49

Average Number of Features Per User

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



The screenshot shows the neo4j console interface. On the left, there's a sidebar with sections for Data services, Tools (Query is selected), and Operations. The main area has two tabs: 'Saved Cypher' and 'neo4j\$'. The 'neo4j\$' tab contains the Cypher query shown above. Below the query, the result '13.411483253588514' is displayed. A note says 'Started streaming 1 record after 63 ms and completed after 91 ms.' At the bottom, there's a search bar and a row of icons.

```
neo4j$  
1 MATCH (u:user2)-[:HAS_FEATURE]->(f:Feature)  
2 WITH u, COUNT(f) AS numFeatures  
3 RETURN AVG(numFeatures) AS AverageFeatures;  
4  
AverageFeatures  
1 13.411483253588514  
Started streaming 1 record after 63 ms and completed after 91 ms.
```

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;
```

The screenshot shows the neo4j browser interface. On the left, there's a sidebar with sections for Data services, Tools (selected), and Operations. Under Tools, the Query tab is active. In the main area, there's a 'Saved Cypher' section with a list of saved queries and a 'Select all' checkbox. Below it is the query editor with the Cypher code. To the right is the results table.

User1	User2	Circle
1030	1043	"circle0"
1043	1252	"circle0"
1030	1254	"circle0"
1252	1254	"circle0"
1174	1549	"circle1"

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;
```

The screenshot shows the Neo4j Aura query console interface. On the left, there's a sidebar with sections for Data services (Instances, Import, Data APIs), Tools (Query, Explore, Operations, Metrics, Logs, Project, Users, Billing), and a search bar. The main area has tabs for 'Saved Cypher' and 'neo4j\$'. Under 'Saved Cypher', there are several queries listed with checkboxes. One query is selected and shown in the code editor:

```

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

The results are displayed in a table:

User1	User2	sharedFeatureCount
930	1549	24

Below the table, a message says "Started streaming 1 record after 98 ms and completed after 2,114 ms." Another query is partially visible below it.

HW Blog(Neo4j) :

[Canvassing 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 won't 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.

