Graph Analytics with Neo4J (Catch the Pink Flamingo Game)

Capstone Project | Week 4 | Nyan Lynn Htet | Sept 8 2024

Modeling Chat Data using a Graph Data Model

The graph data model is utilized to represent user interactions through chat data. Users can initiate chat sessions and add messages within those sessions. A user may be mentioned in a chat message, and one message can reply to another. Additionally, users have the ability to join or exit existing team chat sessions.

Creation of the Graph Database for Chats

Describe the steps you took for creating the graph database. As part of these steps

i) Write the schema of the 6 CSV files
ii) Explain the loading process and include a sample LOAD command
iii) Present a screenshot of some part of the graph you have generated. The graphs must include clearly visible examples of most node and edge types.

schema of the 6 CSV files

chat_create_team_chat.csv	userid: the user id assigned to the user teamid: the id of the team teamChatSessionID: a unique id for the chat session timestamp: a timestamp denoting when the chat session created
chat_item_team_chat.csv	userid: the user id assigned to the user teamchatsessionid: a unique id for the chat session chatitemid: a unique id for the chat item timestamp: a timestamp denoting when the chat item was created
chat_join_team_chat.csv	userid: the user id assigned to the user teamChatSessionID: a unique id for the chat session timestamp: a timestamp denoting when the user joined a chat session

chat_leave_team_chat.csv	userid: the user id assigned to the user teamChatSessionID: a unique id for the chat session timestamp: a timestamp denoting when the user left a chat session
chat_mention_team_chat.csv	ChatItemId: the id of the ChatItem userid: the user id assigned to the user timestamp: a timestamp denoting when the user was mentioned by a chat item
chat_respond_team_chat.csv	chatid1: the id of the chat post 1 chatid2: the id of the chat post 2 timestamp: a timestamp denoting when the chat post 1 responds to the chat post 2

Explain the loading process and include a sample LOAD command

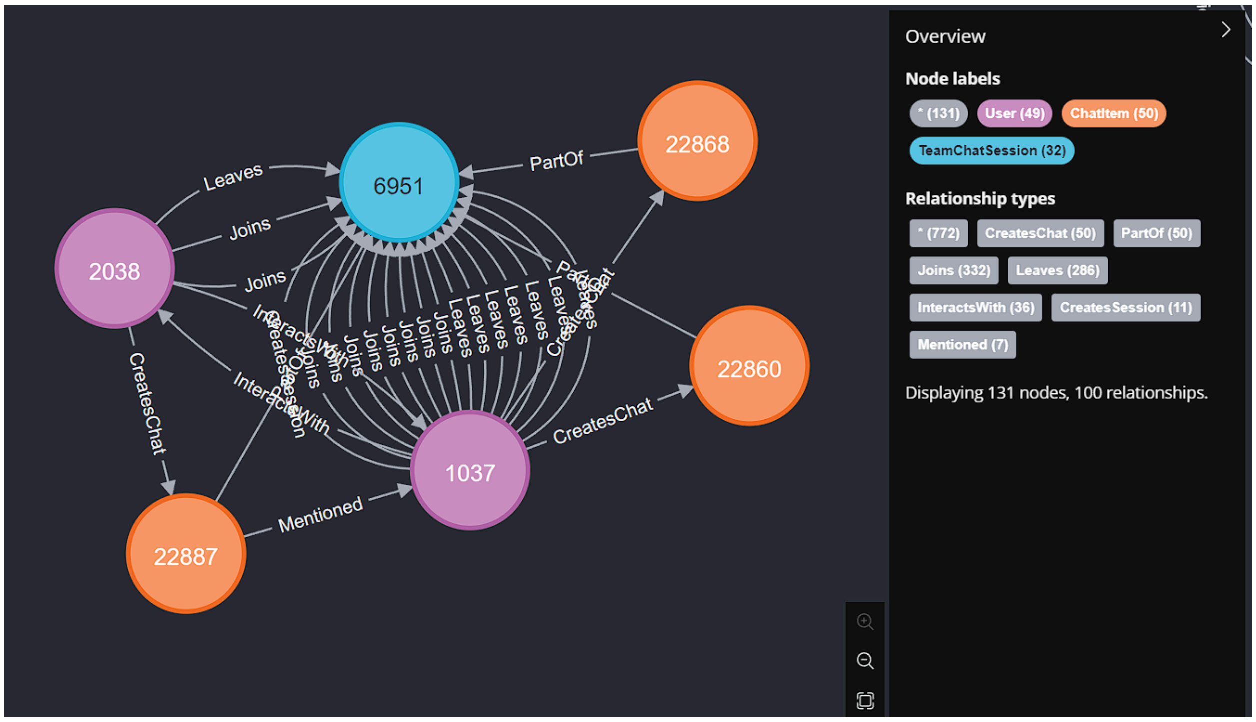
The first line loads the CSV file from a designated location, processing one row at a time. Lines two through four generate nodes for User, Team, and TeamChatSession, with a specific column converted to an integer, which is then assigned to the id attribute. Lines five and six establish the CreatesSession and OwnedBy edges, connecting the previously created nodes. These edges include a timestamp property, which is populated from the fourth column of the schema.

```
CREATE CONSTRAINT FOR (u:User) REQUIRE u.id IS UNIQUE;
CREATE CONSTRAINT FOR (t:Team) REQUIRE t.id IS UNIQUE;
CREATE CONSTRAINT FOR (c:TeamChatSession) REQUIRE c.id IS UNIQUE;
CREATE CONSTRAINT FOR (i:ChatItem) REQUIRE i.id IS UNIQUE;

LOAD CSV FROM "file:///D:/chat_csv/chat_create_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (t:Team {id: toInteger(row[1])})
MERGE (c:TeamChatSession {id: toInteger(row[2])})
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c)
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)
```

Present a screenshot of some part of the graph you have generated. The graphs must include clearly visible examples of most node and edge types.

```
$ MATCH (n)-[r]->(m) RETURN n, r, m  LIMIT 100
```



Overview

Node labels

* (131) User (49) ChatItem (50) TeamChatSession (32)

Relationship types

* (772) CreatesChat (50) PartOf (50) Joins (332) Leaves (286) InteractsWith (36) CreatesSession (11) Mentioned (7)

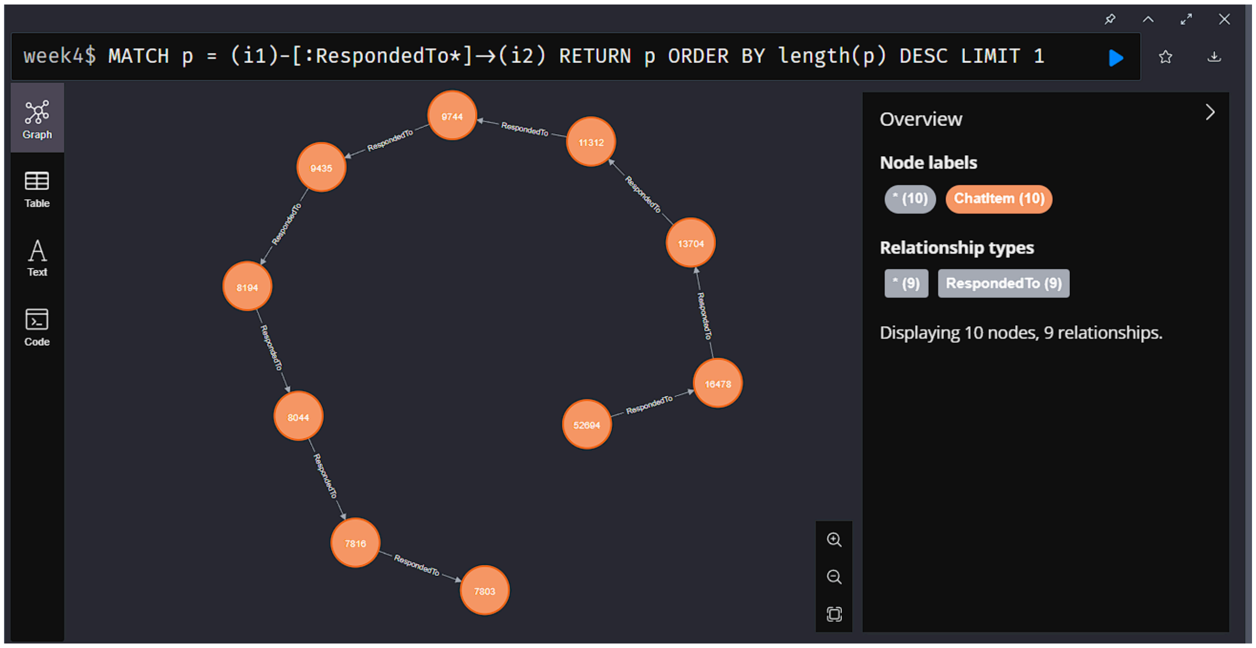
Displaying 131 nodes, 100 relationships.

Finding the longest conversation chain and its participants

Report the results including the length of the conversation (path length) and how many unique users were part of the conversation chain. Describe your steps. Write the query that produces the correct answer.

the length of the conversation (path length)

```
MATCH p = (i1)-[:RespondedTo*]->(i2)
RETURN p ORDER BY length(p) DESC LIMIT 1
```



```
week4$ MATCH p = (i1)-[:RespondedTo*]→(i2) RETURN p ORDER BY length(p) DESC LIMIT 1
```

Overview

Node labels

* (10) ChatItem (10)

Relationship types

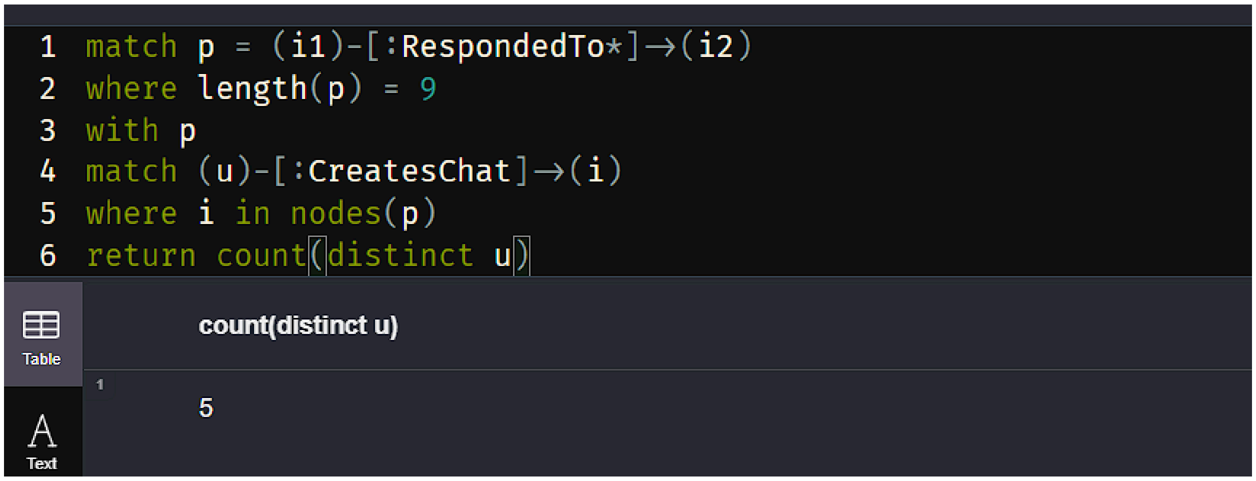
* (9) RespondedTo (9)

Displaying 10 nodes, 9 relationships.

The longest conversation chain in the chat data has path length 9, therefore 10 chats are involved in it.

how many unique users were part of the conversation chain.

```
MATCH p = (i1)-[:RespondedTo*]->(i2)
WHERE length(p) = 9
WITH p
MATCH (u)-[:CreatesChat]->(i)
WHERE i IN nodes(p)
RETURN count(distinct u)
```



```
match p = (i1)-[:RespondedTo*]->(i2)
where length(p) = 9
with p
match (u)-[:CreatesChat]->(i)
where i in nodes(p)
return count(distinct u)
```

	count(distinct u)
1	5

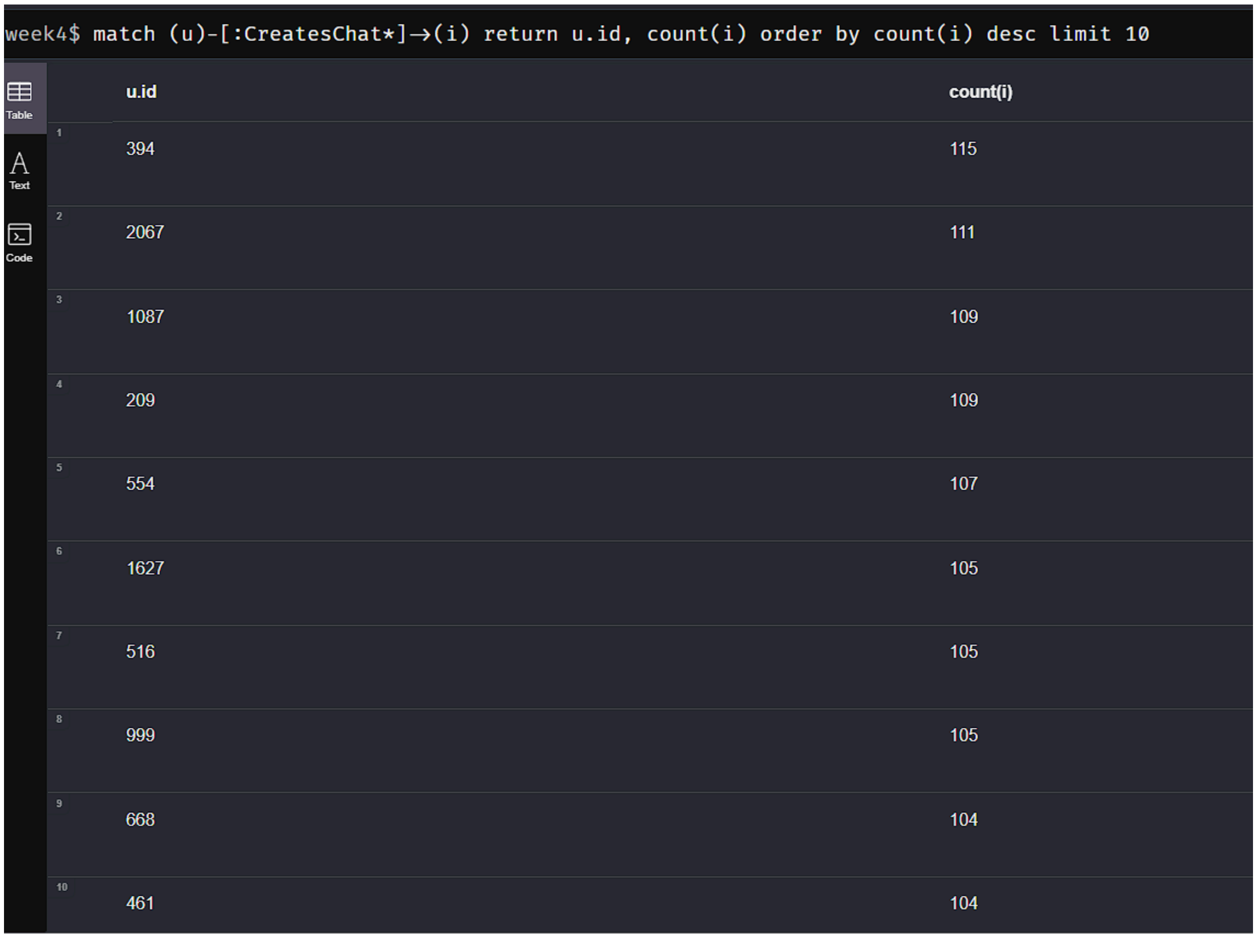
The unique user count is 5

Analyzing the relationship between top 10 chattiest users and top 10 chattiest teams

Chattiest Users

Query the number of chats created by a user from the CreateChat edge

```
MATCH (u)-[:CreatesChat*]->(i)
RETURN u.id, count(i)
ORDER BY count(i) desc limit 10
```



```
week4$ match (u)-[:CreatesChat*]→(i) return u.id, count(i) order by count(i) desc limit 10
```

	u.id	count(i)
1	394	115
2	2067	111
3	1087	109
4	209	109
5	554	107
6	1627	105
7	516	105
8	999	105
9	668	104
10	461	104

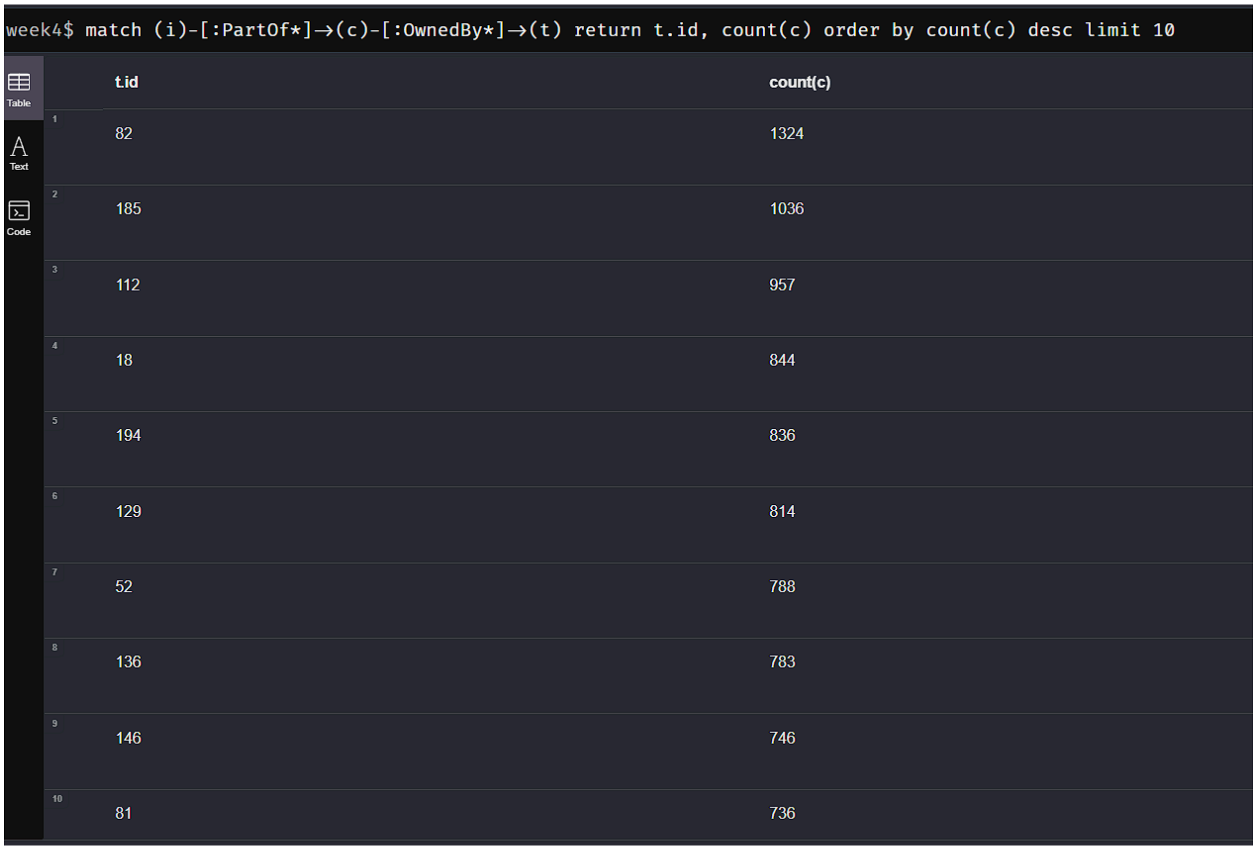
Chattiest Users

Users	Number of Chats
394	115
2067	111
1087	109

.

Chattiest Teams

```
match (i)-[:PartOf*]->(c)-[:OwnedBy*]->(t)
return t.id, count(c)
order by count(c) desc limit 10
```



```
week4$ match (i)-[:PartOf*]→(c)-[:OwnedBy*]→(t) return t.id, count(c) order by count(c) desc limit 10
```

t.id	count(c)
82	1324
185	1036
112	957
18	844
194	836
129	814
52	788
136	783
146	746
81	736

Chattiest Teams

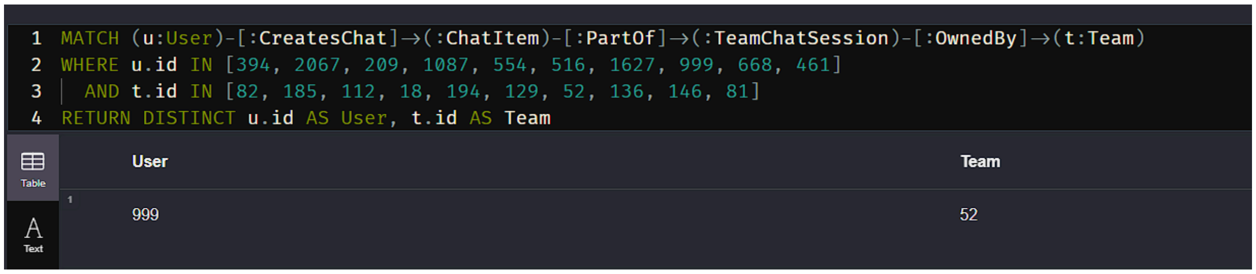
Match all ChatItem with a PartOd edge and connect them with a TeamChatSession node that have an OwnedBy edge connection them with any other node.

Teams	Number of Chats
82	1324
185	1036
112	957

Finally, present your answer, i.e. whether or not any of the chattiest users are part of any of the chattiest teams.

```
MATCH
(u:User)-[:CreatesChat]->(:ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedB
y]->(t:Team)
WHERE u.id IN [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461]
```

```
AND t.id IN [82, 185, 112, 18, 194, 129, 52, 136, 146, 81]
RETURN DISTINCT u.id AS User, t.id AS Team
```



```
MATCH (u:User)-[:CreatesChat]→(:ChatItem)-[:PartOf]→(:TeamChatSession)-[:OwnedBy]→(t:Team)
WHERE u.id IN [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461]
  AND t.id IN [82, 185, 112, 18, 194, 129, 52, 136, 146, 81]
RETURN DISTINCT u.id AS User, t.id AS Team
```

	User	Team
1	999	52

Use this query to investigate if the most chattiest user are part of any chattiest team and it return as: userID 999 is part of teamID 52.

How Active Are Groups of Users?

Describe your steps for performing this analysis. Be as clear, concise, and as brief as possible. Finally, report the top 3 most active users in the table below.

a. Connect mentioned users

```
MATCH (u1:User)-[:CreatesChat]->(:ChatItem)-[:Mentioned]->(u2:User)
MERGE (u1)-[:InteractsWith]->(u2)
```

b. Connect users responses with the chat creator

```
MATCH
(u1:User)-[:CreatesChat]->(:ChatItem)-[:RespondedTo]->(:ChatItem)<-[:Create
sChat]-(u2:User)
MERGE (u1)-[:InteractsWith]->(u2)
```

c. Eliminate all self interaction

```
MATCH (u1)-[r:InteractsWith]->(u1)
DELETE r
```

d. Calculate the cluster coefficient

```
MATCH (u1:User {id: 394})-[:InteractsWith]->(u2:User)
WITH collect(u2.id) AS neighbours, count(u2) AS k
```

```
MATCH (u3:User)-[iw:InteractsWith]->(u4:User)
WHERE u3.id IN neighbours AND u4.id IN neighbours
WITH count(iw) AS triangleCount, k
RETURN CASE WHEN k > 1 THEN triangleCount / (k * (k - 1) * 1.0) ELSE 0 END
AS Clustering_Coefficient

MATCH (u1:User {id: 2067})-[:InteractsWith]->(u2:User)
WITH collect(u2.id) AS neighbours, count(u2) AS k
MATCH (u3:User)-[iw:InteractsWith]->(u4:User)
WHERE u3.id IN neighbours AND u4.id IN neighbours
WITH count(iw) AS triangleCount, k
RETURN CASE WHEN k > 1 THEN triangleCount / (k * (k - 1) * 1.0) ELSE 0 END
AS Clustering_Coefficient

MATCH (u1:User {id: 209})-[:InteractsWith]->(u2:User)
WITH collect(u2.id) AS neighbours, count(u2) AS k
MATCH (u3:User)-[iw:InteractsWith]->(u4:User)
WHERE u3.id IN neighbours AND u4.id IN neighbours
WITH count(iw) AS triangleCount, k
RETURN CASE WHEN k > 1 THEN triangleCount / (k * (k - 1) * 1.0) ELSE 0 END
AS Clustering_Coefficient
```

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
394	0.750
2067	0.933
209	0.999