Graph Analytics

Modeling Chat Data using a Graph Data Model

(Describe the graph model for chats in a few sentences. Try to be clear and complete.)

The Graph Analysis of chat data between “Catch the Pink Flamingo” users catch the interaction and the behaviour of them. Any user from a team is allowed to create a chat session and other users from the same team can join or leave the chat room. Chat-item is when a user creates or replies to a post. These posts are linearly linked to one another as ‘Response’ and all these posts are of the same team chat-session. The relationships between the entities of the graph model are marked with a timestamp and the entities are identified by a unique id value.

Graph model contains 45463 nodes and 118502 edges.

4 Node Types : User, Team, TeamChatSession, ChatItem

8 Edge Types : CreateChat, CreateSession, Join, Leaves, Mentioned, OwnedBy, PartOf and ResponseTo

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

File	Attributes(Description)
chat_create_team_chat.csv	User, Team, TeamChatSession → ID value timeStamp(CreatesSession), timeStamp(OwnedBy) →Date/Time of creation of chatSession & ownership assignment of chatSession to team
chat_join_team_chat.csv	User, TeamChatSession → ID value timestamp(Join) → Date/time of user joining team chat-session
chat_item_team_chat.csv	User, TeamChatSession, ChatItem → ID value timeStamp(CreateChat), timeStamp(Part of) → Date/Time of creation of chatSession,
chat_leave_team_chat.csv	User, TeamChatSession → ID value timeStamp (Leaves) → Date/Time of user leaving the chat group
chat_mention_team_chat.csv	chatItem, User → ID value timestamp(Mentioned) → Date/time of user mentioning a particular post
chat_respond_team_chat.csv	chatItem, chatItem →Post1 ID value, Post2 ID value TimeStamp(ResponseTo) → Date/Time of post1 created as a response to

	post2

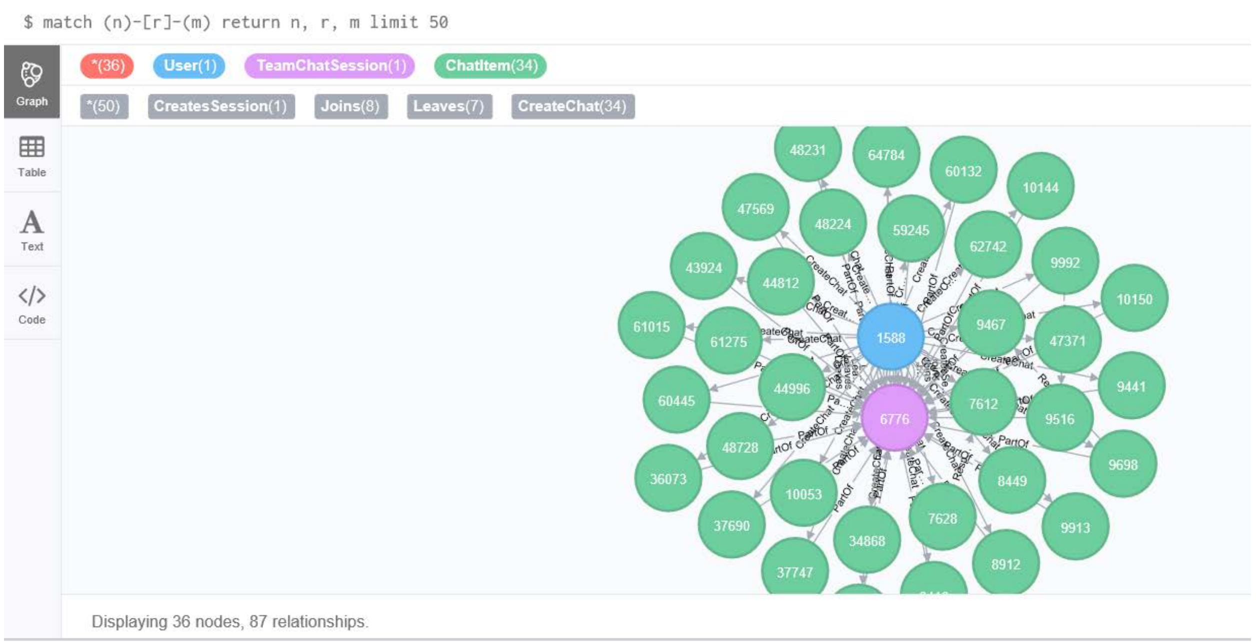
ii) Explain the loading process and include a sample LOAD command

When loading a CSV file, each row is parsed for extracting ID values to create nodes, and timestamp values to create edges. Example :

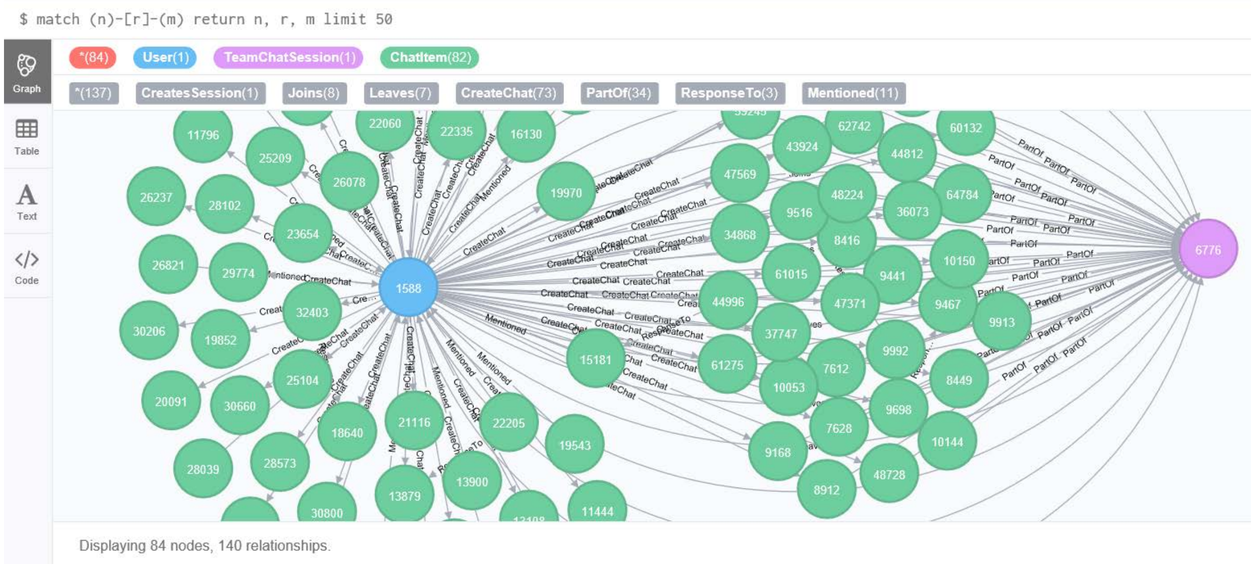
```
LOAD CSV FROM "file:///C:/Users/Sarsiz/Desktop/Capstone%20Project/Week4/chat-
data/chat_item_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (i:ChatItem {id: toInteger(row[2])})
MERGE (u)-[:CreateChat{timestamp: row[3]}]->(i)
MERGE (i)-[:PartOf{timestamp: row[3]}]->(c)
```

1st line loads the csv file from the specified location and parses each line as a 'row' variable.
2nd, 3rd and 4th lines create User, , TeamChatSession and ChatItem nodes, with respective ID values extracted from the 'row' tuple using index values in sequential notation.
4th and 5th lines create the CreateChat and part of edges with respective timestamp values extracted in similar manner, to link up the nodes

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. Below are two acceptable examples. The first example is a rendered in the default Neo4j distribution, the second has had some nodes moved to expose the edges more clearly. Both include examples of most node and edge types.


```
$ match (n)-[r]-(m) return n, r, m limit 50
```

Displaying 36 nodes, 87 relationships.


```
$ match (n)-[r]-(m) return n, r, m limit 50
```

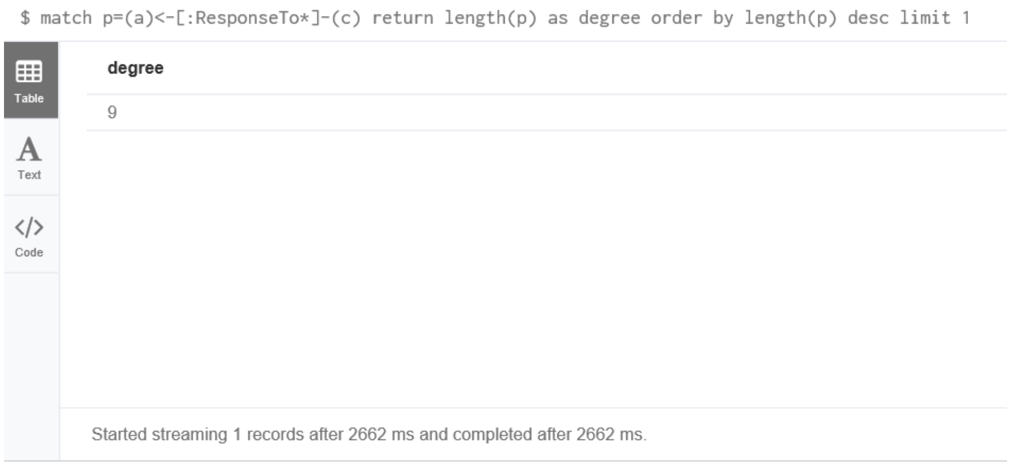
Displaying 84 nodes, 140 relationships.

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 relationship between ChatItems is "ResponseTo". So we will use this to find the longest conversation chain. **The returned result is 9, i.e 10 chat item nodes**

```
match p=(a)<-[:ResponseTo*]-(c)
```
return length(p) as degree order by length(p) desc limit 1



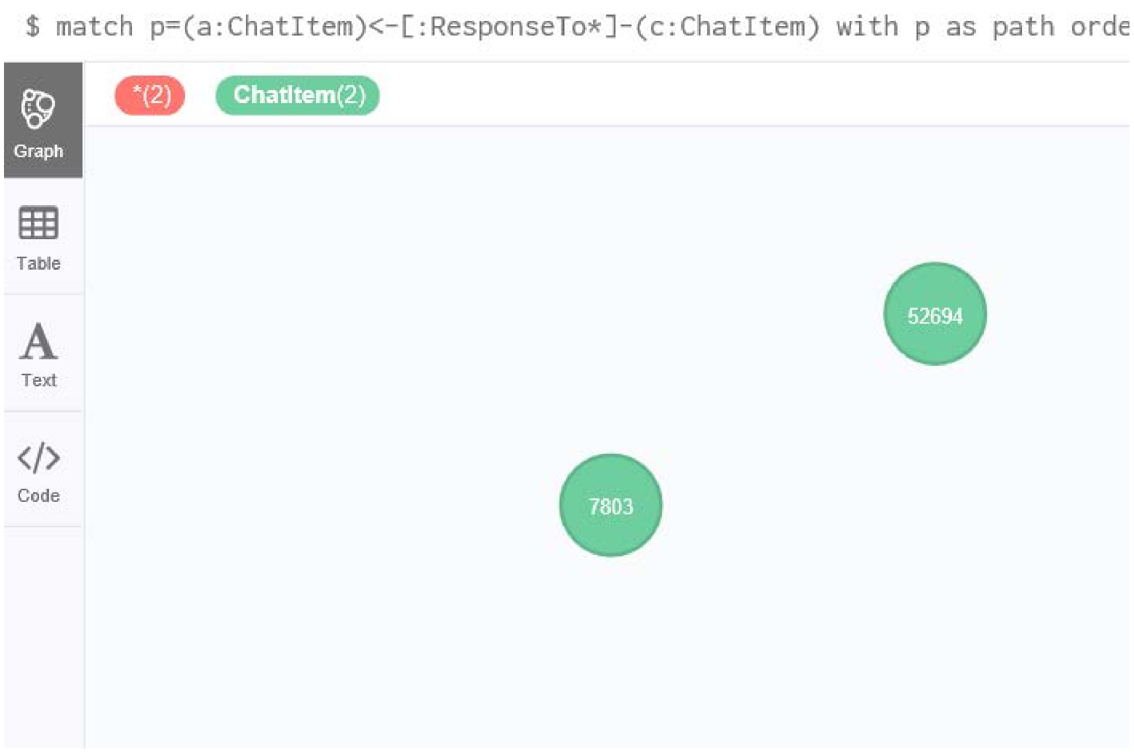
```
$ match p=(a)<-[:ResponseTo*]-(c) return length(p) as degree order by length(p) desc limit 1
```

degree
9

Started streaming 1 records after 2662 ms and completed after 2662 ms.

TO calculate the Number of Users Participated in the chain, I will find the first and the last node and then all the nodes between them.

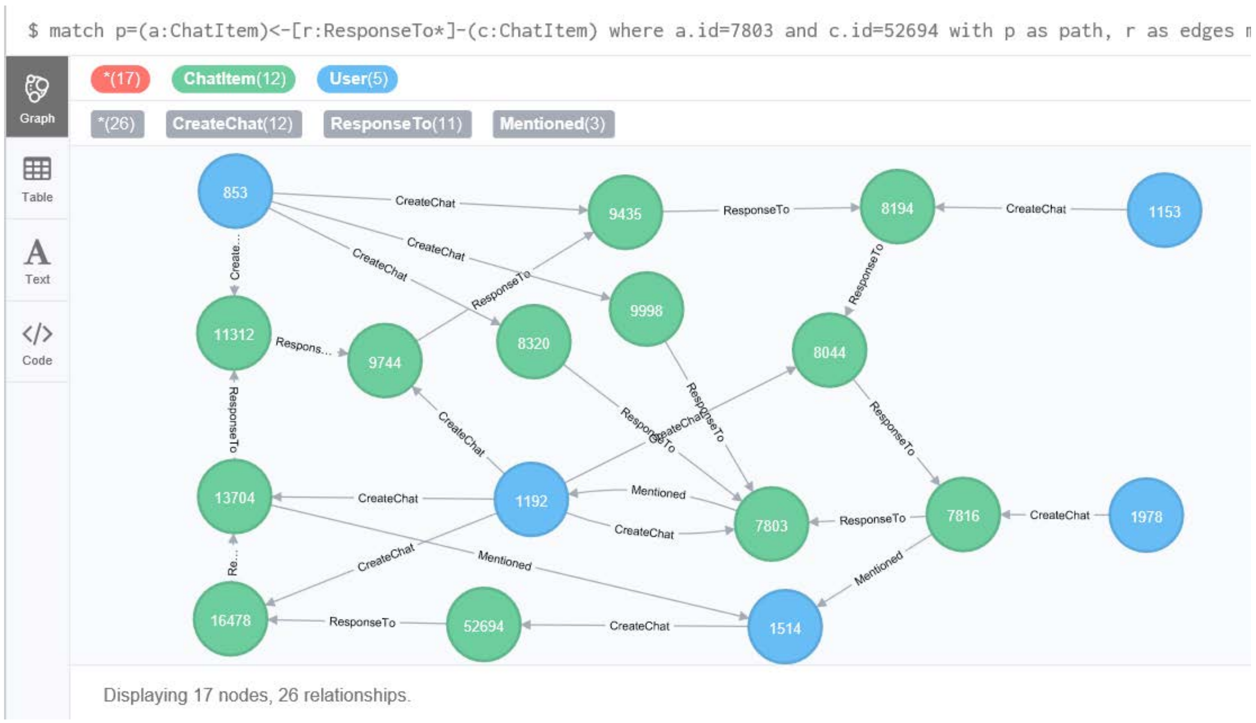
To find the first and last node of the longest conversation chain. Below query finds the longest conversation chain and returns the first and last node of the chain.
match p=(a:ChatItem)<-[:ResponseTo]-(c:ChatItem)*
with p as path order by length(p) desc limit 1
return head(nodes(path)) as first, last(nodes(path)) as last
The first node has an ID of 7803 and the last node has and ID of 52694.



```
$ match p=(a:ChatItem)<-[:ResponseTo*]-(c:ChatItem) with p as path orde
```

Used the IDs of the first and last node in the following query to find the number of users participated in this chain. Below query finds the conversation chain given the id of the first and last node of the chain, then looks for users who created each chat item.

match p=(a:ChatItem)<-[r:ResponseTo]-(c:ChatItem)*
where a.id=7803 and c.id=52694
with p as path, r as edges
match (u)-[r:CreateChat]-(n)
where n in nodes(path)
return u, r, n, edges



```
$ match p=(a:ChatItem)<-[r:ResponseTo*]-(c:ChatItem) where a.id=7803 and c.id=52694 with p as path, r as edges
```

Displaying 17 nodes, 26 relationships.

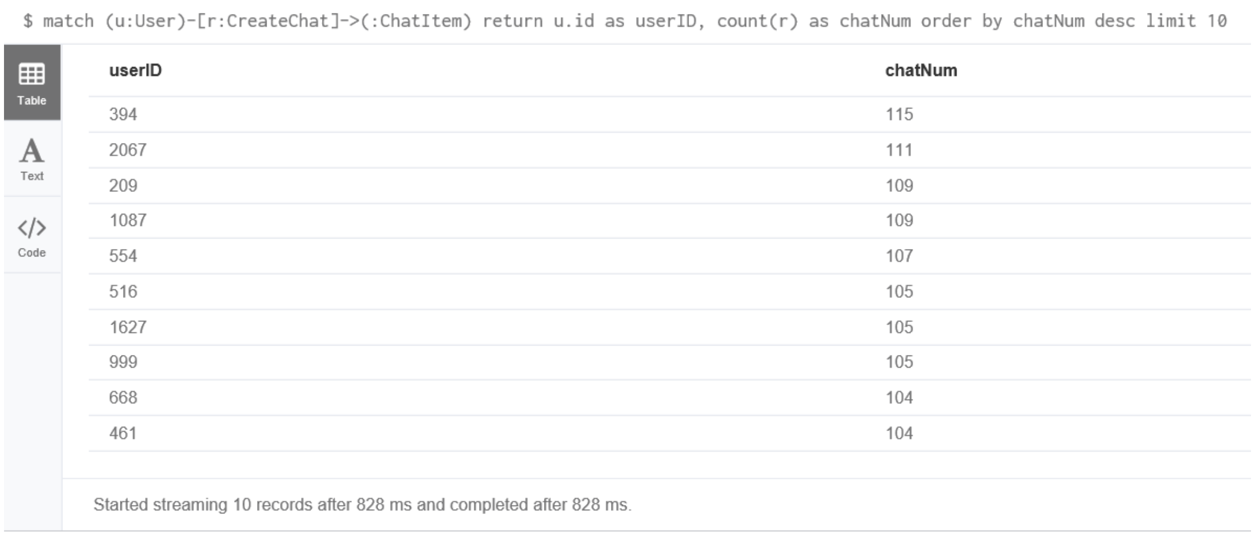
Number of users participating in this chain : 5

UserIDs of 5 users : 1153, 1978, 1514, 1192, 853

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

Describe your steps from Question 2. In the process, create the following two tables. You only need to include the top 3 for each table. Identify and report whether any of the chattiest users were part of any of the chattiest teams.

Here I count the number of chat item created by each user, then sorts the result in descending order and return the top 10 results.
match (u:User)-[r:CreatesChat]->(:ChatItem)
return u.id as userID, count(r) as chatNum order by chatNum desc limit 10


```
$ match (u:User)-[r:CreateChat]->(:ChatItem) return u.id as userID, count(r) as chatNum order by chatNum desc limit 10
```

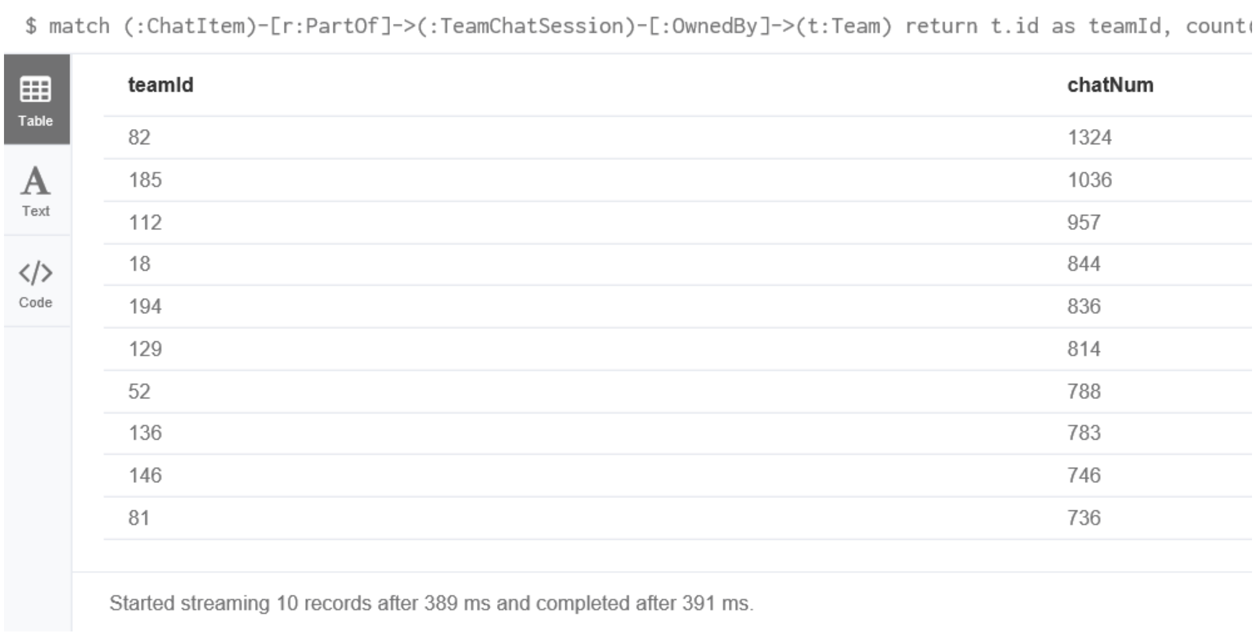
```
userID    chatNum
394       115
2067      111
209       109
1087      109
554       107
516       105
1627      105
999       105
668       104
461       104

Started streaming 10 records after 828 ms and completed after 828 ms.
```

Chattiest Users

Users	Number of Chats
394(1st)	115
2067(2nd)	111
209(3rd)	109
1087(3rd)	109

To find the chattiest team, I applied the query :

match (:ChatItem)-[r:PartOf]->(:TeamChatSession)-[:OwnedBy]->(t:Team)
return t.id as teamId, count(r) as chatNum order by chatNum desc limit 10



```
$ match (:ChatItem)-[r:PartOf]->(:TeamChatSession)-[:OwnedBy]->(t:Team) return t.id as teamId, count(
```

```
teamId                                                              chatNum
82                                                                  1324
185                                                                 1036
112                                                                 957
18                                                                  844
194                                                                 836
129                                                                 814
52                                                                  788
136                                                                 783
146                                                                 746
81                                                                  736

Started streaming 10 records after 389 ms and completed after 391 ms.
```

Chattiest Teams

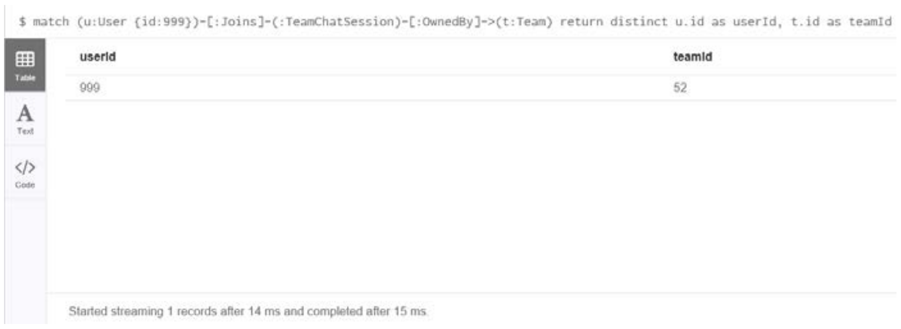
Teams	Number of Chats
82	1324
185	1036
112	957

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

I find teamID of each top 10 chattiest users. Below query takes in a userID (replace XXXX by userID), and uses the Joins and OwedBy relationship to find the chat session a user have joined, then finds the team that owns the chat session

match (u:User {id:XXXX})-[:Joins]-(:TeamChatSession)-[:OwnedBy]->(t:Team)
return distinct u.id as userId, t.id as teamId

It turned out that only one of the top 10 chattiest users is from the top 10 chattiest Team. This user has a userID **of 999 (8th in top 10 chattiest user), and the team for this user has a teamID of 52.**



```
$ match (u:User {id:999})-[:Joins]-(:TeamChatSession)-[:OwnedBy]->(t:Team) return distinct u.id as userId, t.id as teamId
```

```
userId                                              teamId
999                                                 52

Started streaming 1 records after 14 ms and completed after 15 ms.
```

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.

First I will create relations between the users who interact with each other.

match (u1:User)-[:CreateChat]->(i:ChatItem)-[:Mentioned]->(u2:User)
create (u1)-[:InteractsWith]->(u2)

Above query is used to create relations for users who mentioned another user in chat.

Now, I will be creating relations for users who created a chatItem in response to another users' chatitem.

match (u1:User)-[:CreateChat]->(i1:ChatItem)-[:ResponseTo]->(i2:ChatItem)<-[:CreateChat]-(u2:User)
create (u1)-[:InteractsWith]->(u2)

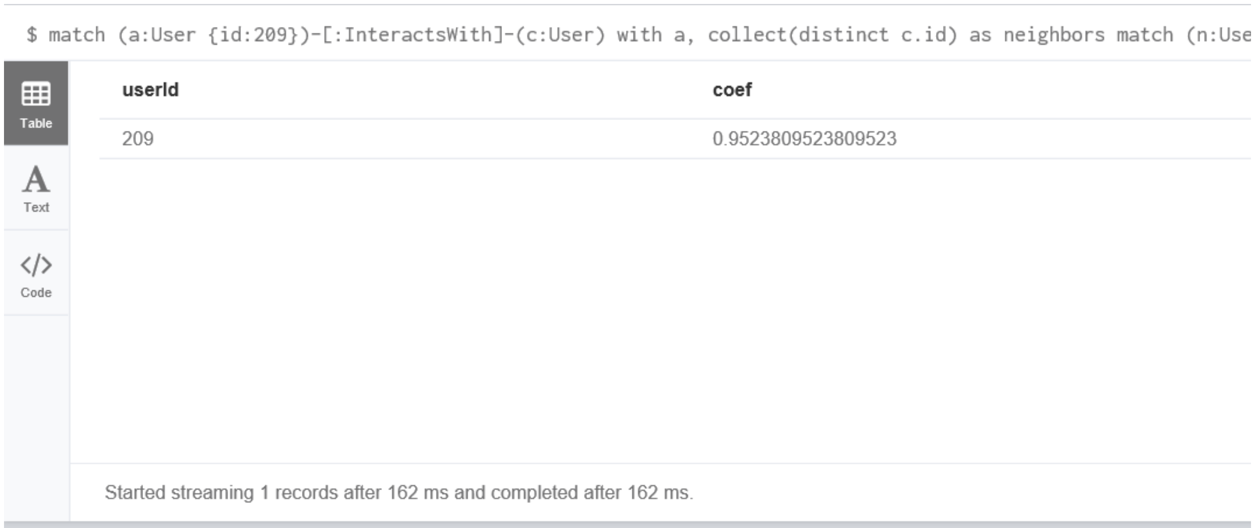
I deleted any loop relations that connects a user to himself or herself

match (u)-[r:InteractsWith]->(u)
delete r

Now I find the clustering coefficient. Equation is *coef=numEdge/(k*(k-1)).* This will take a userID and it will find all direct neighbours and cal. k(no. of neighbours). Now I will calculate numEdge while examining the connectivity of every pair of neighbour nodes.

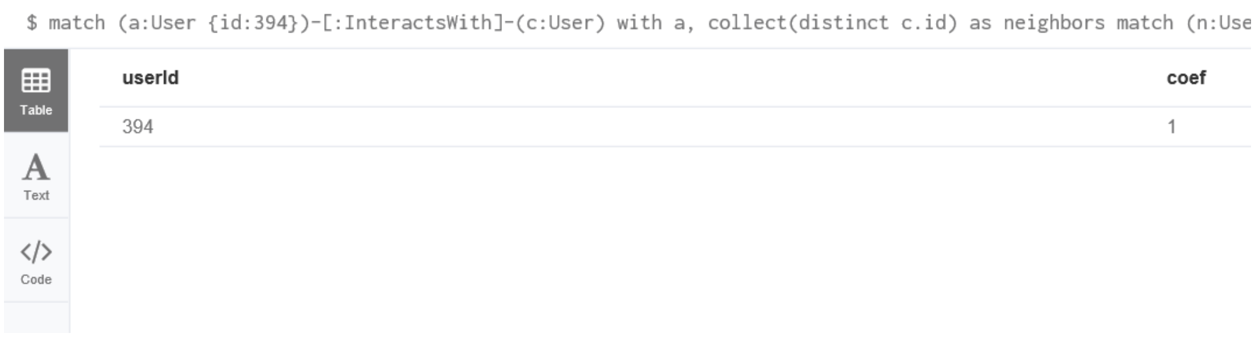
In the end numEdge is divided by *k*(k-1)* to get the clustering coefficient.

match (a:User {id:XXXX})-[:InteractsWith]-(c:User)
with a, collect(distinct c.id) as neighbors
match (n:User), (m:User)
where (n.id in neighbors) and (m.id in neighbors) and (n<>m)
with a, length(neighbors) as k, sum (case when (n)-[:InteractsWith]-(m) then 1 else 0 end) as numEdge
*return a.id as userId, 1.0*numEdge/(k*(k-1)) as coef*


```
$ match (a:User {id:209})-[:InteractsWith]-(c:User) with a, collect(distinct c.id) as neighbors match (n:Use
```

userId	coef
209	0.9523809523809523

Started streaming 1 records after 162 ms and completed after 162 ms.


```
$ match (a:User {id:394})-[:InteractsWith]-(c:User) with a, collect(distinct c.id) as neighbors match (n:Use
```

userId	coef
394	1

Similarly by putting the different ids we can get the coefficients

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
394	1
461	1
209	0.9523809523809523
516	0.9523809523809523
554	0.904761904761904
999	0.866666666666666
1087	0.8
2067	0.785714285714285
1627	0.785714285714285
668	0.7