Graph Analytics

Modeling Chat Data using a Graph Data Model

The graph consists of Users who create ChatItems that are part of TeamChatSessions. These TeamChatSessions can be crated by Users and owned by Teams. The ChatItems can mention Users or respond to other ChatItems. Users can join or leave TeamChatSessions during time. All these relations include a timestamp.

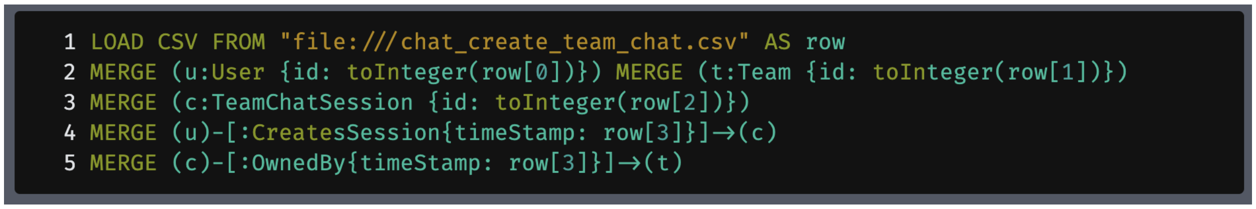
Creation of the Graph Database for Chats

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

i) Write the schema of the 6 CSV files

Dataset	Columns
chat_create_team_chat	userid teamid teamchatsessionid timestamp
chat_item_team_chat	userid teamchatsessionid chatitemid timestamp
chat_join_team_chat	userid teamchatsessionid timestamp
chat_leave_team_chat	userid teamchatsessionid timestamp
chat_mention_team_chat	chatitemid userid timestamp
chat_respond_team_chat	chatid1 chatid2 timestamp

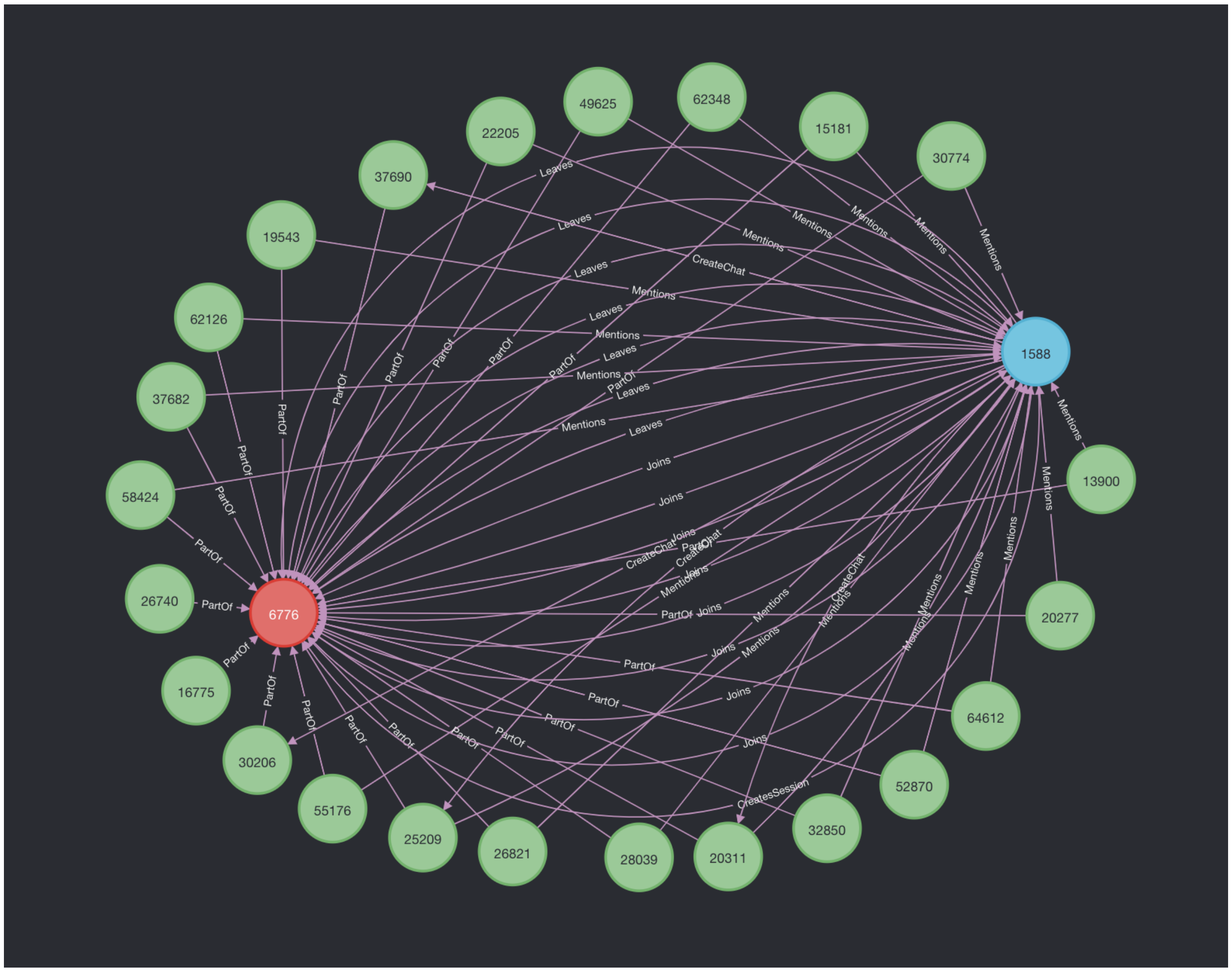
ii) Explain the loading process and include a sample LOAD command



```
LOAD CSV FROM "file:///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)
```

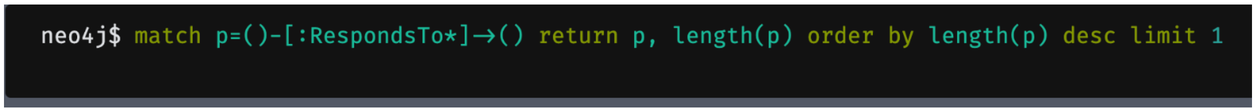
The first line imports the csv archive and reads it row by row. The second line creates User and Team nodes and sets id attribute as an integer from an specific column of each row. The third line does the same process for TeamChatSession nodes. The fourth line creates an edge named CreateSession from Users to TeamChatSession nodes with a timestamp attribute. The fifth line does the same process for edges OwnedBy from TeamChatSession to Team

iii) Present a screenshot of some part of the graph you have generated.

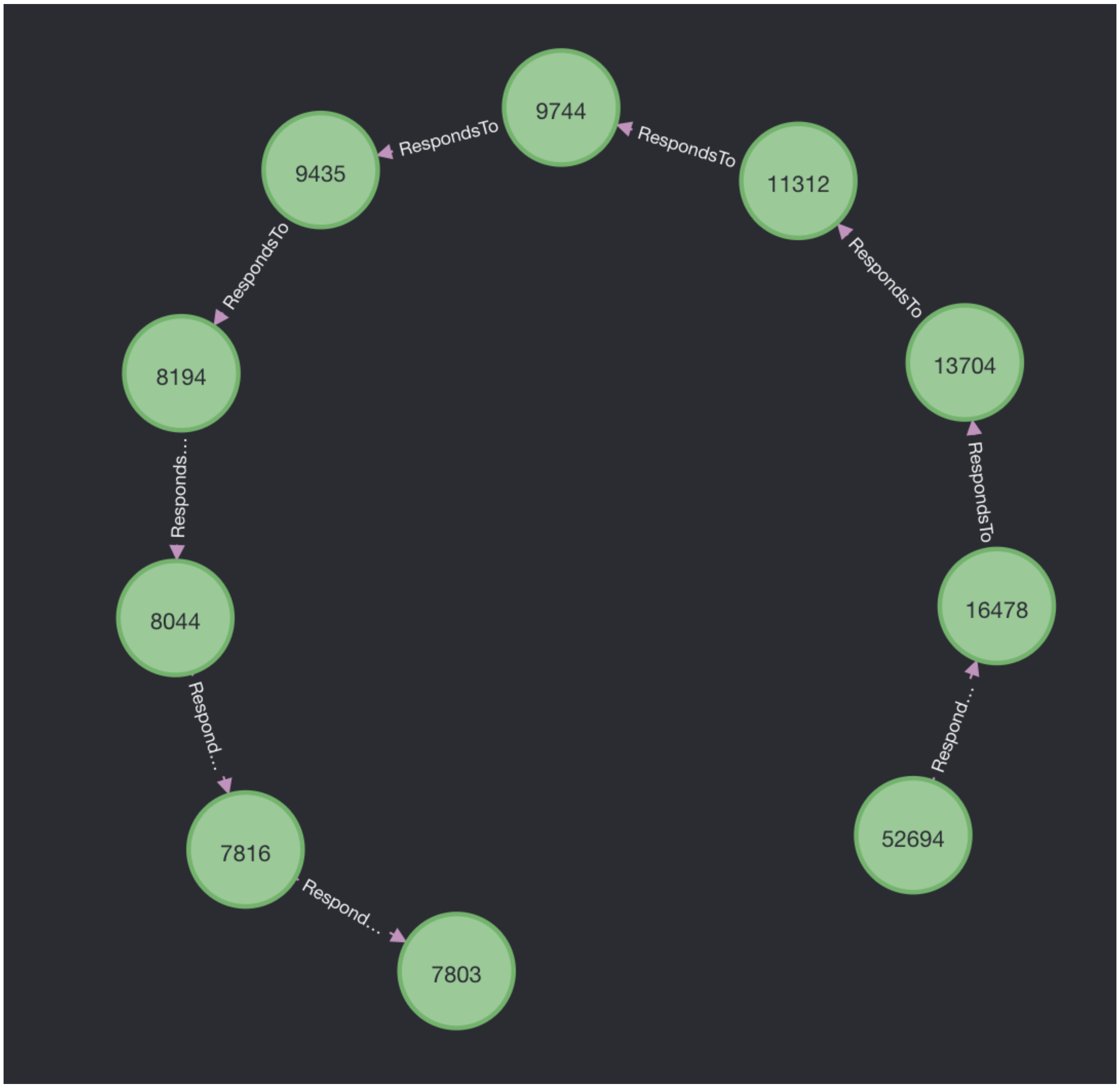


Finding the longest conversation chain and its participants

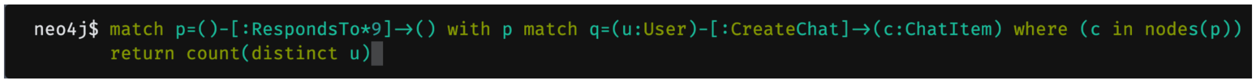
Report the results including the length of the conversation (path length) and how many unique users were part of the conversation chain. Describe your steps. Write the query that produces the correct answer.



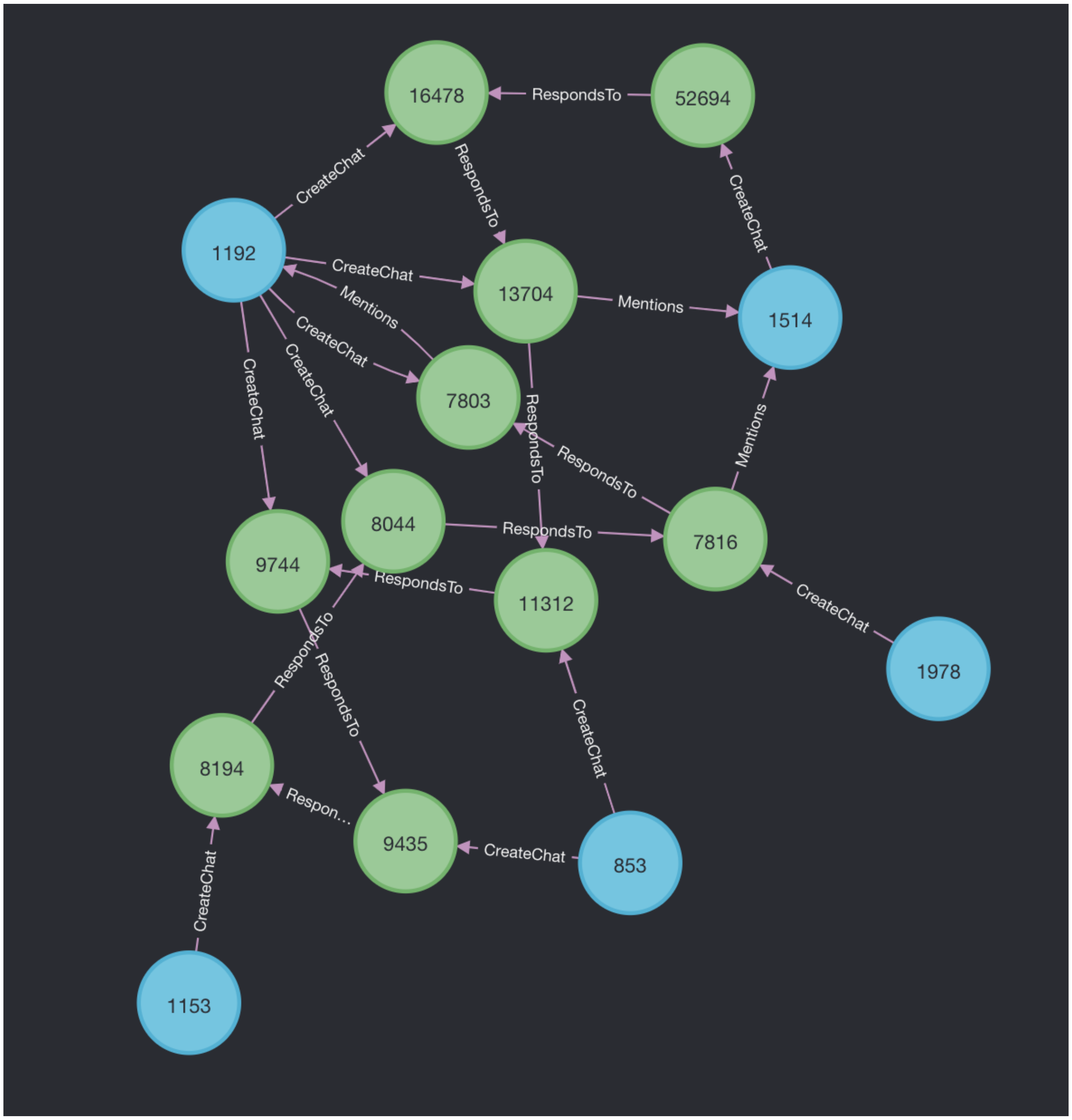
```
neo4j$ match p=()-[:RespondsTo*]→() return p, length(p) order by length(p) desc limit 1
```



The query results in a longest path of length 9, meaning the longest chat chain consists of 10 ChatItems.



```
neo4j$ match p=()-[:RespondsTo*9]→() with p match q=(u:User)-[:CreateChat]→(c:ChatItem) where (c in nodes(p))
       return count(distinct u)
```

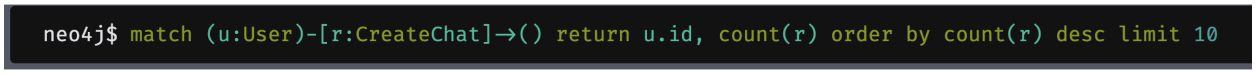


The query return a total of 5 distinct users involved in the longest chat chain

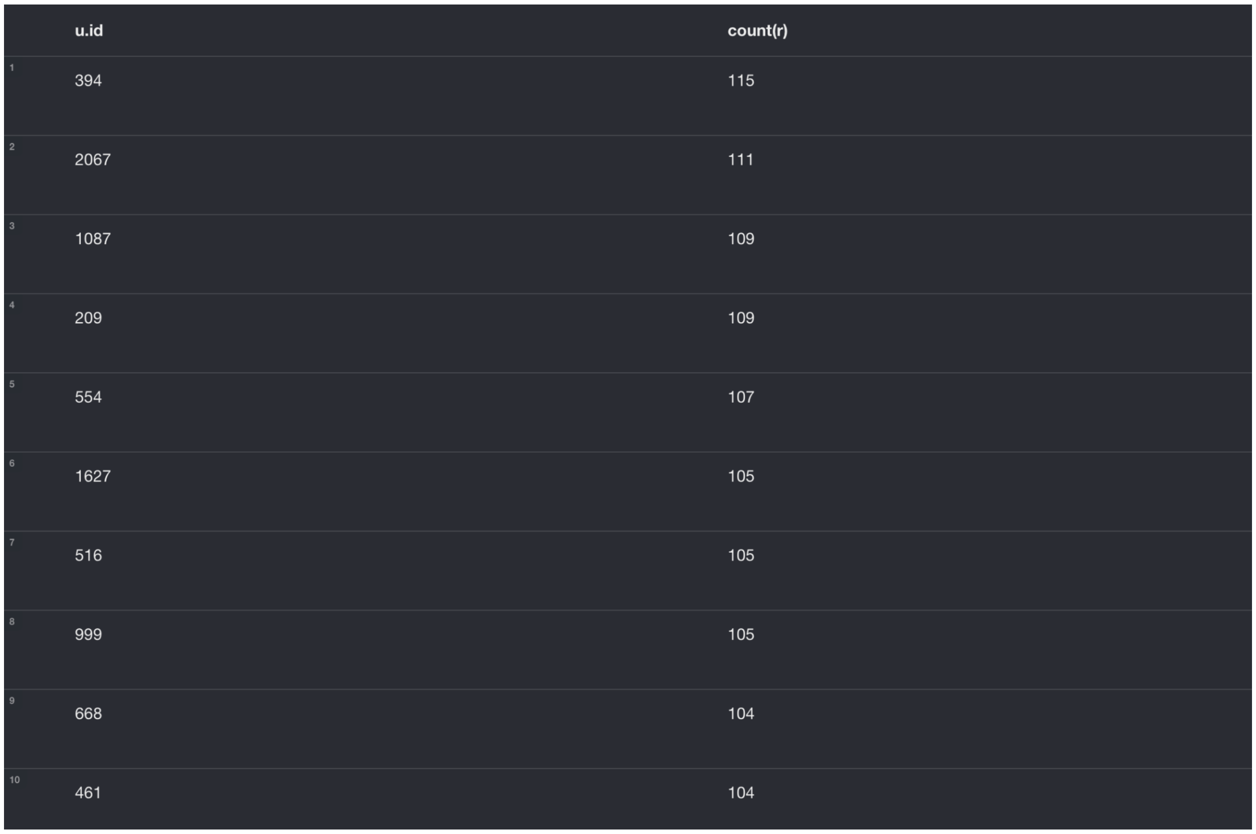
Analyzing the relationship between top 10 chattiest users and top 10 chattiest teams

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.

Chattiest Users



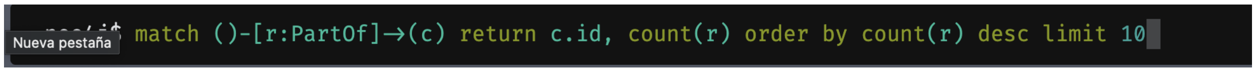
```
neo4j$ match (u:User)-[r:CreateChat]→() return u.id, count(r) order by count(r) desc limit 10
```



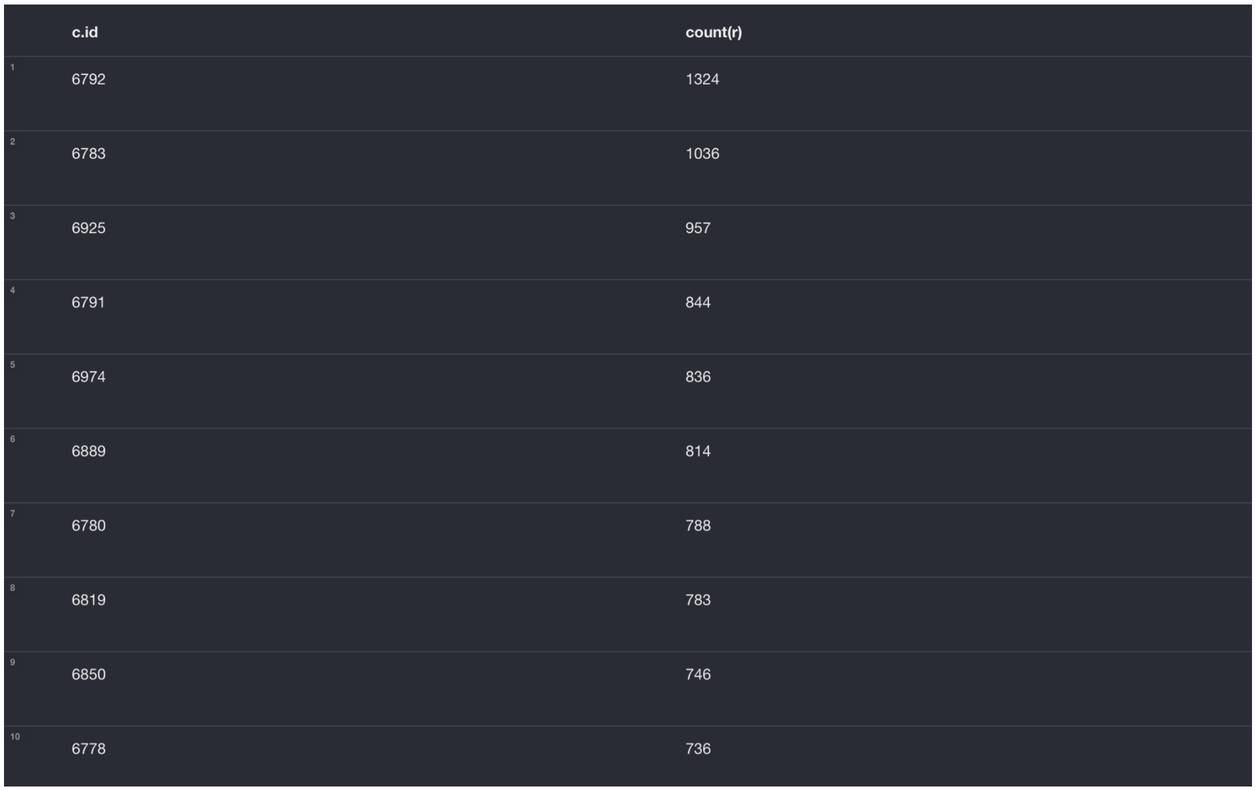
```
   u.id                                   count(r)
1  394                                    115
2  2067                                   111
3  1087                                   109
4  209                                    109
5  554                                    107
6  1627        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

Users	Number of Chats
394	115
2067	111
1087	109

Chattiest Teams



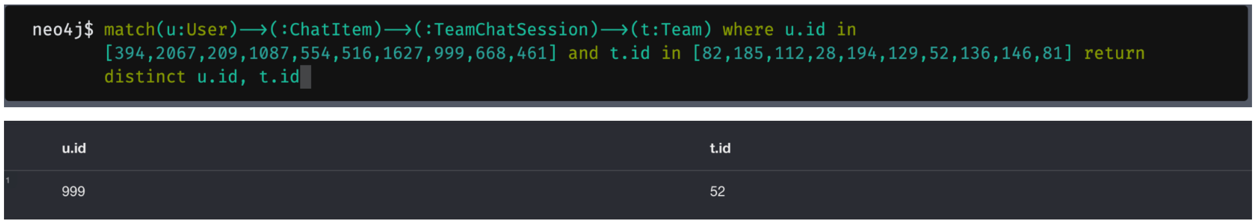
```
neo4j$ match ()-[r:PartOf]→(c) return c.id, count(r) order by count(r) desc limit 10
```



	c.id	count(r)
1	6792	1324
2	6783	1036
3	6925	957
4	6791	844
5	6974	836
6	6889	814
7	6780	788
8	6819	783
9	6850	746
10	6778	736

Teams	Number of Chats
6792	1324
6783	1036
6925	957

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



```
neo4j$ match(u:User)⟶(:ChatItem)⟶(:TeamChatSession)⟶(t:Team) where u.id in
       [394,2067,209,1087,554,516,1627,999,668,461] and t.id in [82,185,112,28,194,129,52,136,146,81] return
       distinct u.id, t.id
```

	u.id	t.id
1	999	52

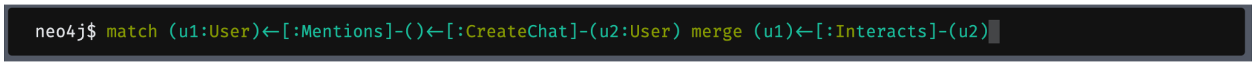
The query searches for the chattiest users that belong to the chattiest teams. It only shows a pair consisting of User 999 belonging to Team 52. It shows that in general the 10 chattiest Users do not belong to the 10 chattiest Teams.

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.

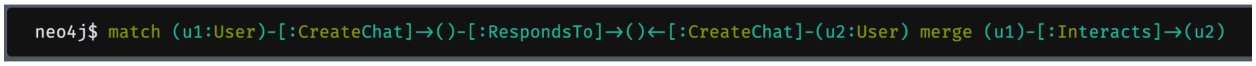
Most Active Users (based on Cluster Coefficients)

This query creates relations Interacts from an User who mention another User in a ChatItem



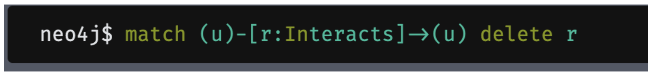
```
neo4j$ match (u1:User)←[:Mentions]-()←[:CreateChat]-(u2:User) merge (u1)←[:Interacts]-(u2)
```

This query creates relations Interacts from an User who responds to another User in a ChatITem



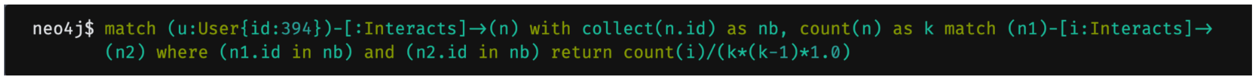
```
neo4j$ match (u1:User)-[:CreateChat]→()-[:RespondsTo]→()←[:CreateChat]-(u2:User) merge (u1)-[:Interacts]→(u2)
```

This query deletes all self loops for Interacts relation



```
neo4j$ match (u)-[r:Interacts]→(u) delete r
```

This query searches for a specific User's neighbours and evaluates how many connections are there between those neighbours. Then computes a coefficient relative to the max amount of possible relations.



```
neo4j$ match (u:User{id:394})-[:Interacts]→(n) with collect(n.id) as nb, count(n) as k match (n1)-[i:Interacts]→
       (n2) where (n1.id in nb) and (n2.id in nb) return count(i)/(k*(k-1)*1.0)
```

User ID	Coefficient
394	0.7500
2067	0.9333
1087	0.7333