Graph Analytics

Modeling Chat Data using a Graph Data Model

This is a Graph Model for chats created by users of the game “To Catch The Pink Flamingo”. The Graph Model has 4 types of nodes (**Users**, **Teams**, **Team Chat Sessions** and **Chat Items**)

Each node has its own unique ID. There are various types of edges, each represents a different kind of interaction between nodes

Edges such as ‘**CreatesSession**’, ‘Joins’ and ‘Leaves’ describes the interaction Users have on the Chat Sessions

Edges such as ‘**CreatesChat**’ describes the interaction Users have on the Chat Items

Edges such as ‘**PartOf**’ describes which Chat Sessions the Chat Items belong to

Edges such as ‘**OwnedBy**’ describes which Teams the Chat Sessions belong to

Edges such as ‘**Mentions**’ and ‘**ResponsesTo**’ describes the interaction ChatItems have towards Users or other Chat Items respectively

Creation of the Graph Database for Chats

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

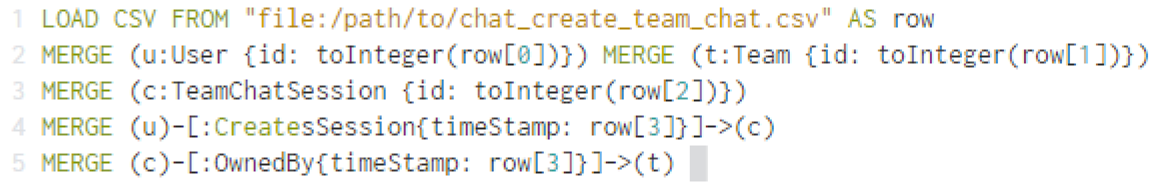
i) Write the schema of the 6 CSV files

Chat_create_team_chat	• userid: ID of user: Int • teamid: ID of team: Int • TeamChatSessionID: ID of chat session: Int • timestamp: Timestamp -- for userid **CREATING** TeamChatSessionID -- for TeamChatSessionID **OWNEDBY** teamid
Chat_item_team_chat	• userid: ID of user: Int • teamchatsessionid: ID of chat session: Int • chatitemid: ID of chat item: Int • timestamp: Timestamp -- for userid **CREATING** chatitemid -- for chatitemid **PARTOF** teamchatsessionid
Chat_join_team_chat	• userid: ID of user: Int • TeamChatSessionID: ID of chat session: Int • timestamp: Timestamp -- for userid **JOINING** TeamChatSessionID

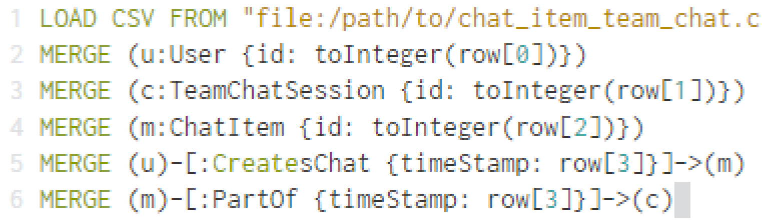
Chat_leave_team_chat	• userd: ID of user: Int • teamchatsessionid: ID of chat session: Int • timestamp: Timestamp -- for userid **LEAVING** teamchatsessionid
Chat_mention_team_chat	• ChatItem: ID of chat item: Int • userid: ID of user: Int • timestamp: Timestamp -- for ChatItem **MENTIONING** userid
Chat_respond_team_chat	• chatid1: ID of chat item: Int • chatid2: ID of chat item: Int • timestamp: Timestamp -- for chatid1 **RESPONDINGTO** chatid2

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

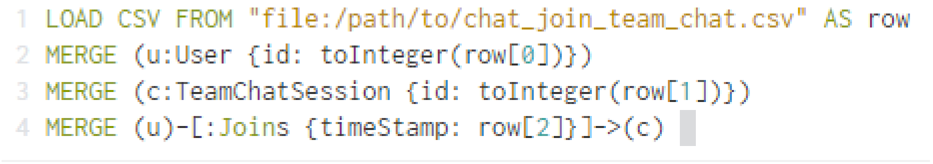
- The loading process determines **which columns** in the CSV file should be **used as integer ids** for the **nodes** and **which single column** should be used as **timestamp of the interaction**.
- Nodes are read as (**nodechar**: **Nodetype** {id: toInteger(row[**row order**])}). Bolded fields are filled in by the Neo4j user
- Edges are read as –[: **Interactiontype** {timestamp: row[**order of timestamp column**]}] –>. Bolded fields are filled in by the Neo4j user
- The CSV **headers** reveal what **type of nodes** that is whereas the **type of interaction** has to be **guessed** via the **file name**. There can be **more than one** type of interaction
- If there are many types of interaction, the timestamp column is shared by them



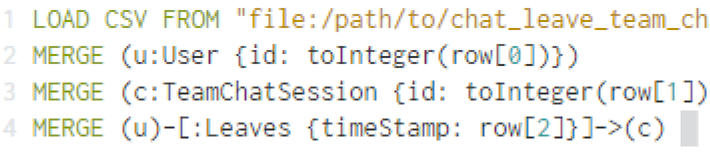
```
LOAD CSV FROM "file:/path/to/chat_create_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])}) MERGE (t:Team {id: toInteger(row[1])})
MERGE (c:TeamChatSession {id: toInteger(row[2])})
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c)
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)
```



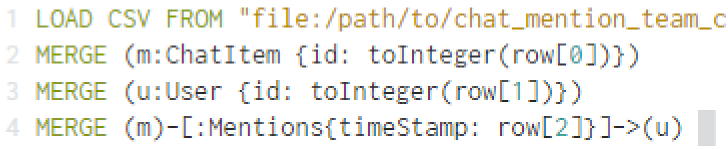
```
LOAD CSV FROM "file:/path/to/chat_item_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (m:ChatItem {id: toInteger(row[2])})
MERGE (u)-[:CreatesChat {timeStamp: row[3]}]->(m)
MERGE (m)-[:PartOf {timeStamp: row[3]}]->(c)
```



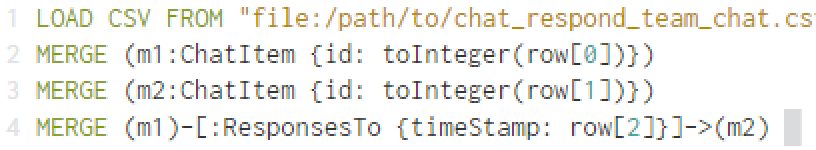
```
LOAD CSV FROM "file:/path/to/chat_join_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Joins {timeStamp: row[2]}]->(c)
```



```
LOAD CSV FROM "file:/path/to/chat_leave_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Leaves {timeStamp: row[2]}]->(c)
```

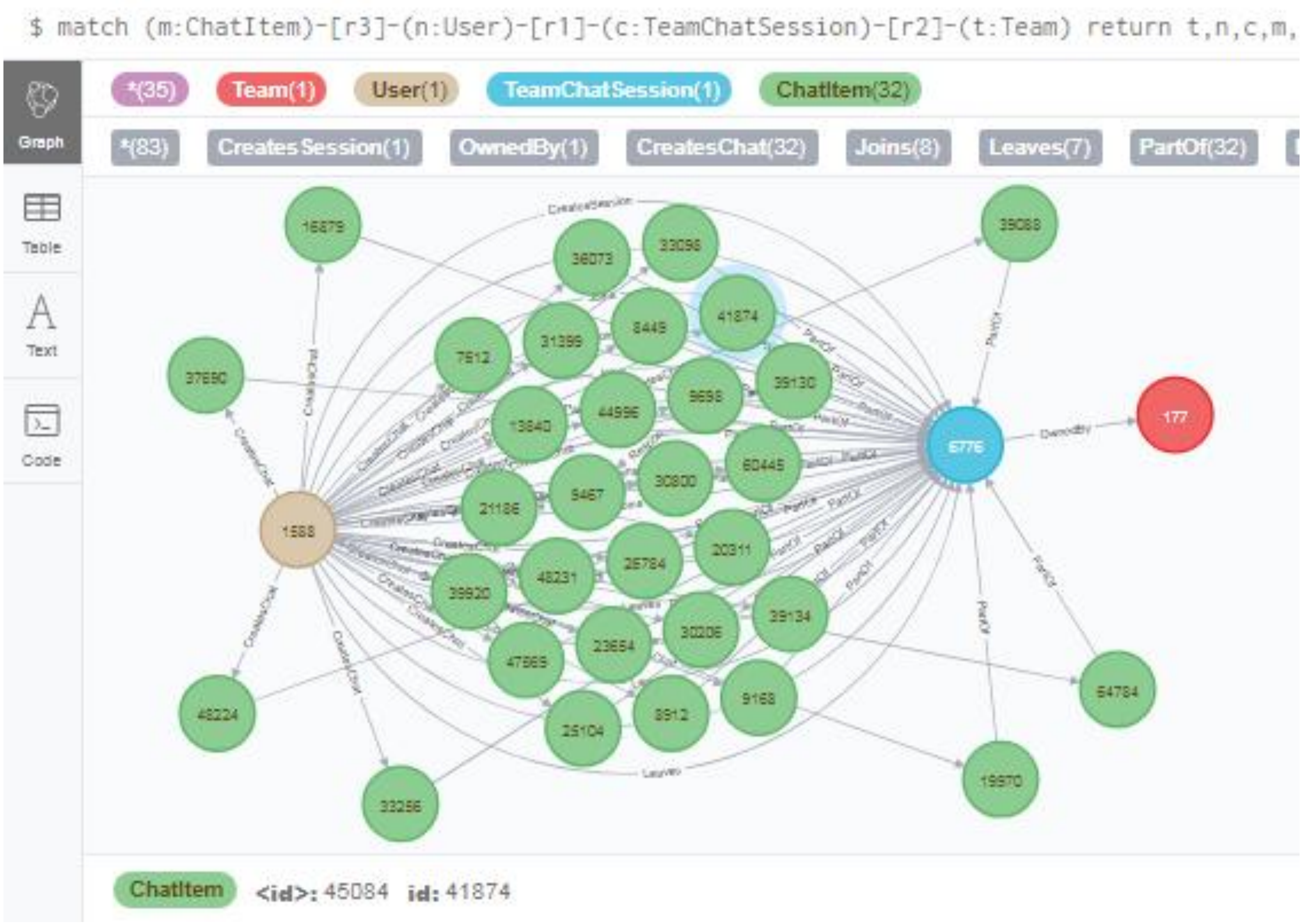


```
LOAD CSV FROM "file:/path/to/chat_mention_team_chat.csv" AS row
MERGE (m:ChatItem {id: toInteger(row[0])})
MERGE (u:User {id: toInteger(row[1])})
MERGE (m)-[:Mentions{timeStamp: row[2]}]->(u)
```



```
LOAD CSV FROM "file:/path/to/chat_respond_team_chat.csv" AS row
MERGE (m1:ChatItem {id: toInteger(row[0])})
MERGE (m2:ChatItem {id: toInteger(row[1])})
MERGE (m1)-[:ResponsesTo {timeStamp: row[2]}]->(m2)
```

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.

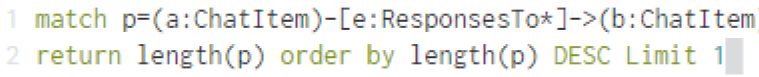


```
$ match (m:ChatItem)-[r3]-(n:User)-[r1]-(c:TeamChatSession)-[r2]-(t:Team) return t,n,c,m,
```

ChatItem <id>: 45084 id: 41874

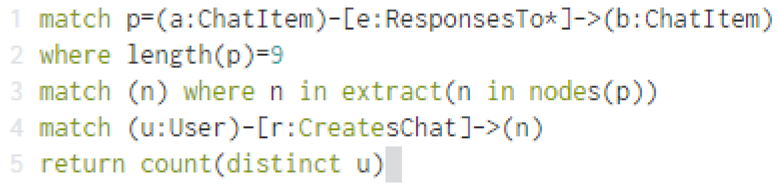
Finding the longest conversation chain and its participants

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.



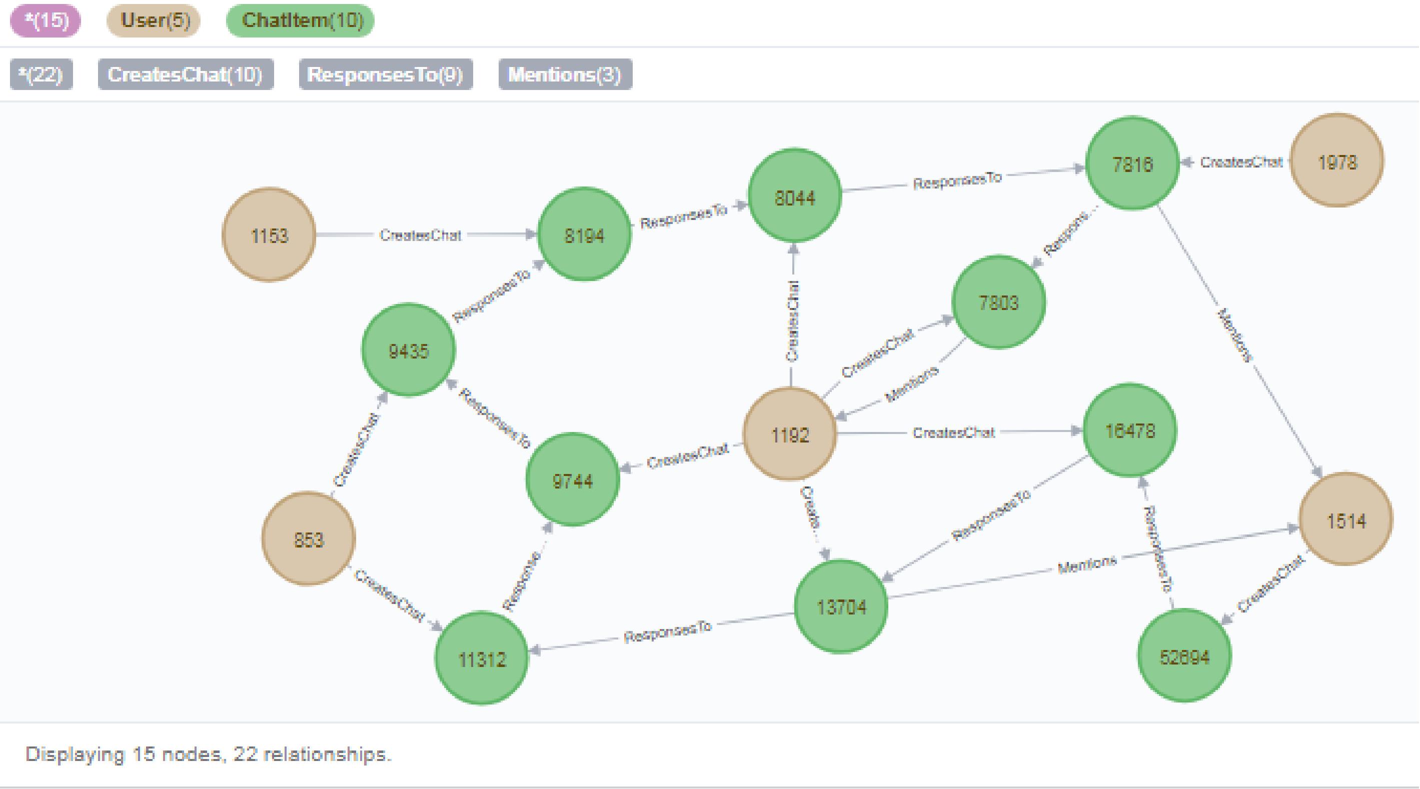
```
match p=(a:ChatItem)-[e:ResponsesTo*]->(b:ChatItem)
return length(p) order by length(p) DESC Limit 1
```

Longest conversation chain's length = 9



```
match p=(a:ChatItem)-[e:ResponsesTo*]->(b:ChatItem)
where length(p)=9
match (n) where n in extract(n in nodes(p))
match (u:User)-[r:CreatesChat]->(n)
return count(distinct u)
```

Number of unique users in this chain = 5 (1153, 1192, 1514, 1978, 853)



Displaying 15 nodes, 22 relationships.

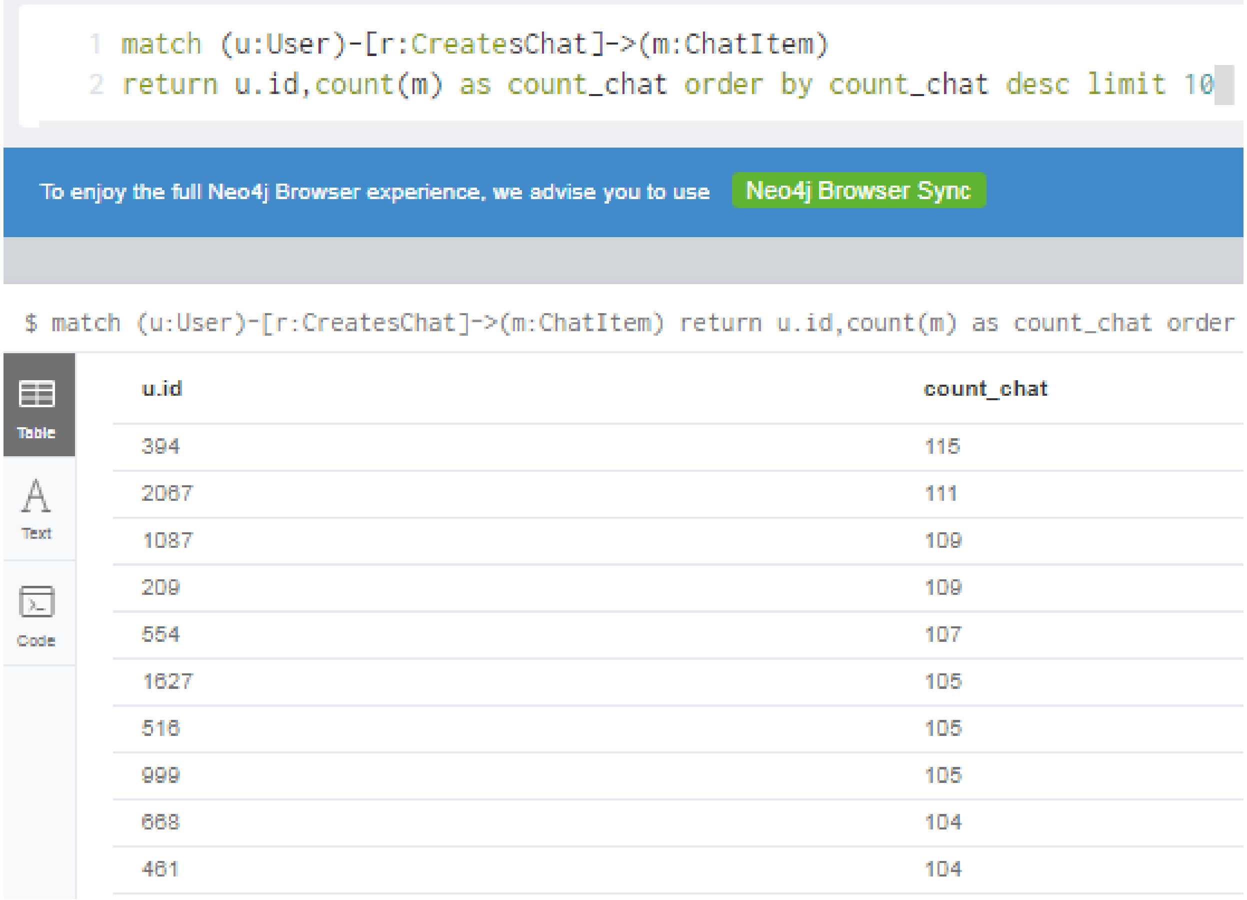
The entire longest conversation chain path, consisting of 5 users and 10 chat items

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

- Find all Users –CreatesChat->ChatItem relationships.
- Then count number of these ChatItems for each User
- Then rank in descending order by this count (limit to top 10)



```
match (u:User)-[r:CreatesChat]->(m:ChatItem)
return u.id,count(m) as count_chat order by count_chat desc limit 10
```

To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync

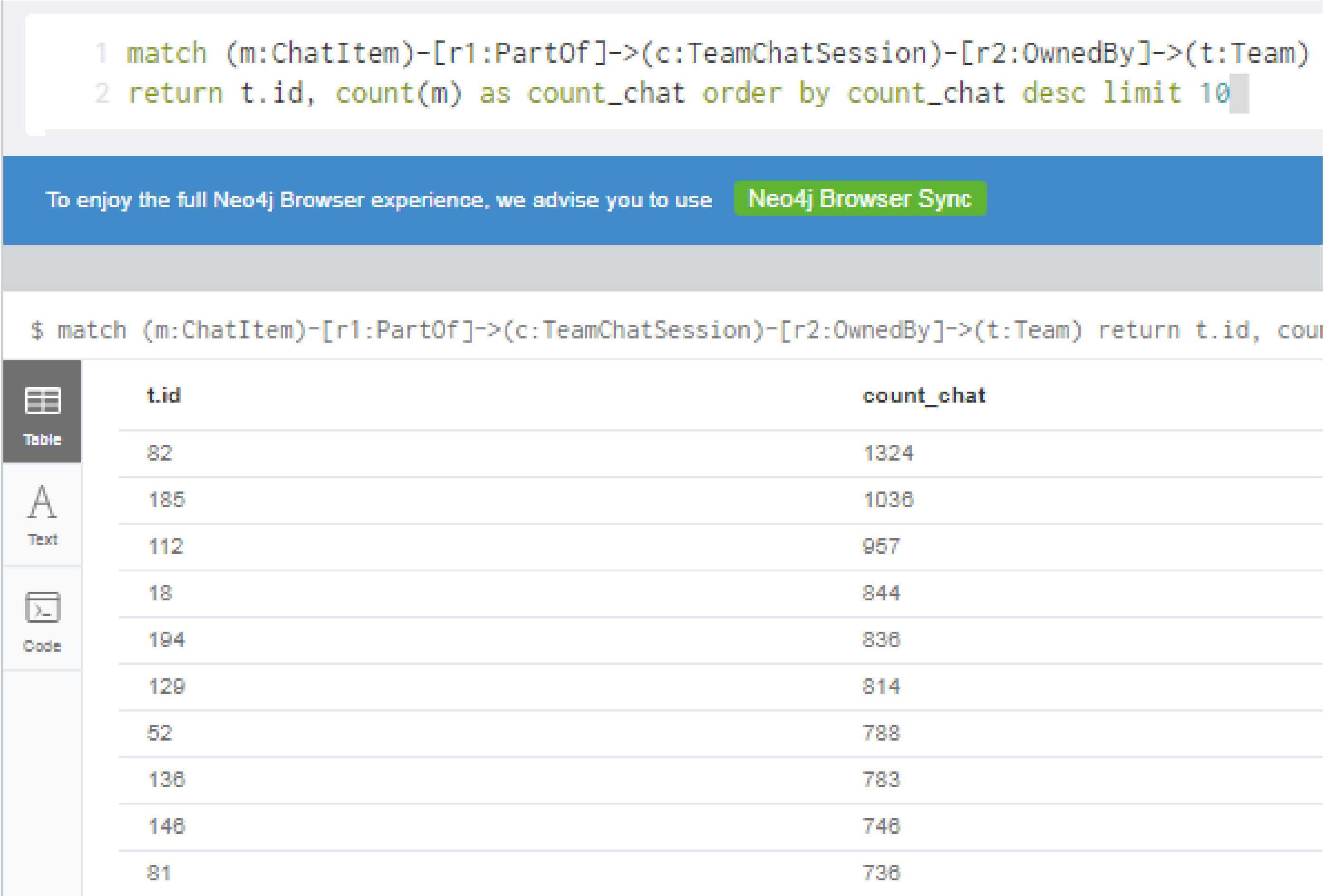
```
$ match (u:User)-[r:CreatesChat]->(m:ChatItem) return u.id,count(m) as count_chat order
```

u.id	count_chat
394	115
2067	111
1087	109
209	109
554	107
1627	105
516	105
999	105
668	104
461	104

Users	Number of Chats
394	115
2067	111
1087 ties with 209	109

Chattiest Teams

- Find all ChatItem –PartOf-> TeamChatSession –OwnedBy-> Team relationships.
- Then count number of these ChatItems for each Team
- Then rank in descending order by this count (limit to top 10)



```
match (m:ChatItem)-[r1:PartOf]->(c:TeamChatSession)-[r2:OwnedBy]->(t:Team)
return t.id, count(m) as count_chat order by count_chat desc limit 10
```

To enjoy the full Neo4j Browser experience, we advise you to use
Neo4j Browser Sync

```
$ match (m:ChatItem)-[r1:PartOf]->(c:TeamChatSession)-[r2:OwnedBy]->(t:Team) return t.id, cou
```

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

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.

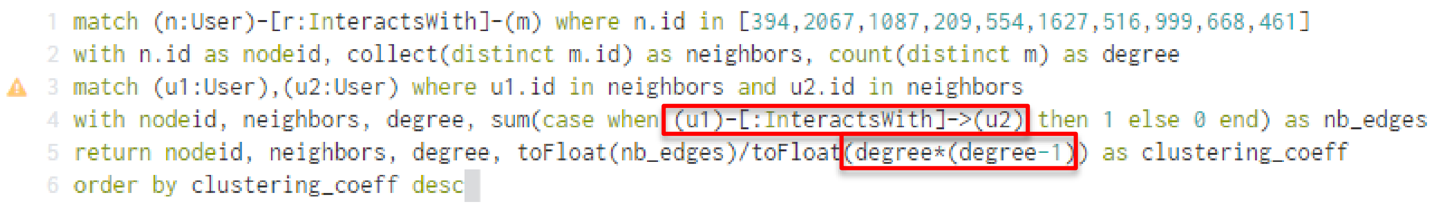
- Out of 10 chattiest users, only **user 999** belongs to the chattiest **team 52**
- **No other relationship observed**

How Active Are Groups of Users?

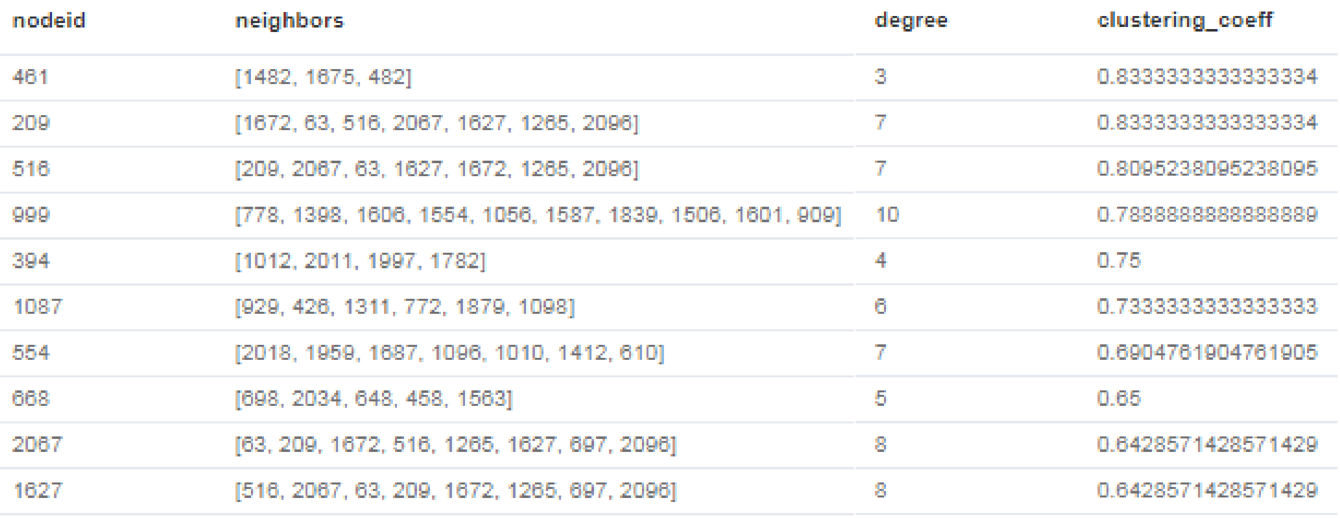
Describe your steps for performing this analysis. Be as clear, concise, and as brief as possible. Finally, report the top 3 most active users in the table below.

There are two ways to consider connection between neighbors' nodes:

1) Consider the edges as **directed edges**. That means if there are 2 nodes interacting with each other that connection should be **counted twice** & maximum number of connections **= k*(k-1)**



```
match (n:User)-[r:InteractsWith]-(m) where n.id in [394,2067,1087,209,554,1627,516,999,668,461]
with n.id as nodeid, collect(distinct m.id) as neighbors, count(distinct m) as degree
match (u1:User),(u2:User) where u1.id in neighbors and u2.id in neighbors
with nodeid, neighbors, degree, sum(case when (u1)-[:InteractsWith]->(u2) then 1 else 0 end) as nb_edges
return nodeid, neighbors, degree, toFloat(nb_edges)/toFloat(degree*(degree-1)) as clustering_coeff
order by clustering_coeff desc
```



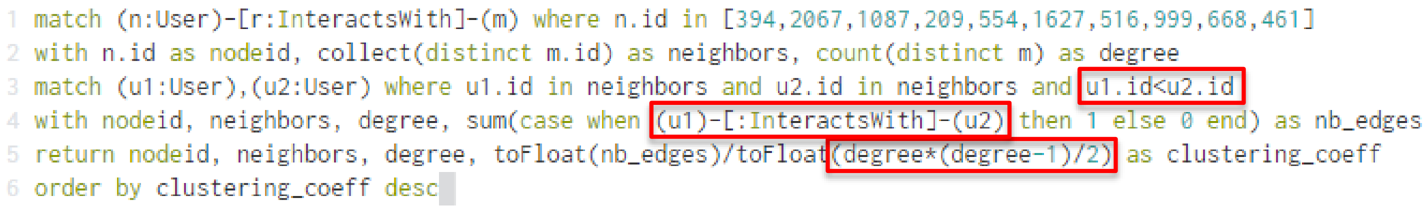
nodeid	neighbors	degree	clustering_coeff
461	[1482, 1675, 482]	3	0.8333333333333334
209	[1672, 63, 516, 2067, 1627, 1265, 2096]	7	0.8333333333333334
516	[209, 2067, 63, 1627, 1672, 1265, 2096]	7	0.8095238095238095
999	[778, 1398, 1606, 1554, 1056, 1587, 1839, 1506, 1601, 909]	10	0.7888888888888889
394	[1012, 2011, 1997, 1782]	4	0.75
1087	[929, 426, 1311, 772, 1879, 1098]	6	0.7333333333333333
554	[2018, 1959, 1687, 1096, 1010, 1412, 610]	7	0.6904761904761905
668	[698, 2034, 648, 458, 1563]	5	0.65
2067	[63, 209, 1672, 516, 1265, 1627, 697, 2096]	8	0.6428571428571429
1627	[516, 2067, 63, 209, 1672, 1265, 697, 2096]	8	0.6428571428571429

THIS METHOD SEEMS TO BE THE ONE **IMPLIED BY THE ASSIGNMENT INSTRUCTION**. HOWEVER THE RESULTING COEFFICIENT VALUES **DON'T MATCH THE GRADING RUBRIC**

Most Active Users (based on Cluster Coefficients) – METHOD 1

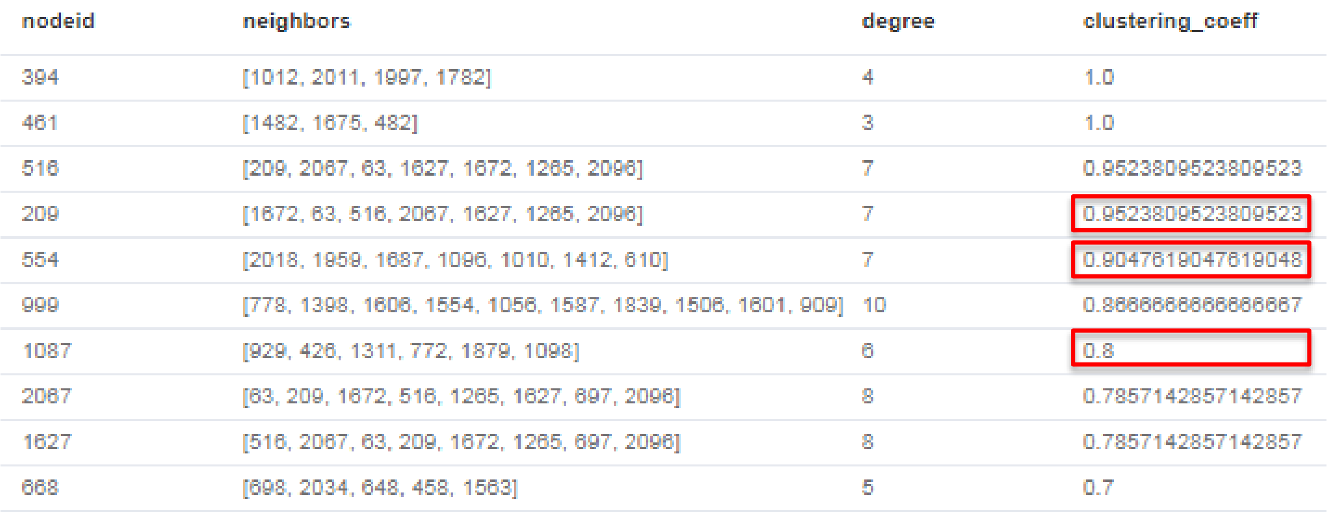
User ID	Coefficient
461	0.8333
209	0.8333
516	0.8095

2) Consider the edges as **un-directed edges**. That means if there are 2 nodes interacting with each other that connection should be **counted exactly once** & maximum number of connections = **k*(k-1)/2**



```
match (n:User)-[r:InteractsWith]-(m) where n.id in [394,2067,1087,209,554,1627,516,999,668,461]
with n.id as nodeid, collect(distinct m.id) as neighbors, count(distinct m) as degree
match (u1:User),(u2:User) where u1.id in neighbors and u2.id in neighbors and u1.id<u2.id
with nodeid, neighbors, degree, sum(case when (u1)-[:InteractsWith]-(u2) then 1 else 0 end) as nb_edges
return nodeid, neighbors, degree, toFloat(nb_edges)/toFloat(degree*(degree-1)/2) as clustering_coeff
order by clustering_coeff desc
```

The u1.id < u2.id is to prevent double counting



nodeid	neighbors	degree	clustering_coeff
394	[1012, 2011, 1997, 1782]	4	1.0
461	[1482, 1675, 482]	3	1.0
516	[209, 2067, 63, 1627, 1672, 1265, 2096]	7	0.9523809523809523
209	[1672, 63, 516, 2067, 1627, 1265, 2096]	7	0.9523809523809523
554	[2018, 1959, 1687, 1096, 1010, 1412, 610]	7	0.9047619047619048
999	[778, 1398, 1606, 1554, 1056, 1587, 1839, 1506, 1601, 909]	10	0.8666666666666667
1087	[929, 426, 1311, 772, 1879, 1098]	6	0.8
2067	[63, 209, 1672, 516, 1265, 1627, 697, 2096]	8	0.7857142857142857
1627	[516, 2067, 63, 209, 1672, 1265, 697, 2096]	8	0.7857142857142857
668	[698, 2034, 648, 458, 1563]	5	0.7

THIS METHOD **DOESN'T** SEEM TO BE THE ONE **IMPLIED BY THE ASSIGNMENT INSTRUCTION**. HOWEVER THE RESULTING COEFFICIENT VALUES **MATCH THE GRADING RUBRIC.** BUT **GOD KNOWS WHY** THEY ARE REGARDED AS **THE TOP 3**

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
394	1.0
461	1.0
516	0.9524