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 describes the following network. When one User creates a TeamChatSession, it is then owned by team. Users can join and leave the TeamChatSession. In TeamChatSession, users can create ChatItem that is part of TeamChatSession. ChatItem could also be mentioned by Users. And User could respond to User as well. All the relationships are recorded with timestamp.
Four types of nodes:

1. User
2. Team
3. TeamChatSession
4. ChatItem.

Seven types of relationships:

1. User creates TeamChatSession with timestamp
2. Team owns TeamChatSession with timestamp
3. User joins TeamChatSession with timestamp
4. User leaves TeamChatSession with timestamp
5. User creates ChatItem with timestamp
6. ChatItem is part of TeamChatSession with timestamp
7. ChatItem is mentioned by User with timestamp
8. ChatItem responses to ChatItem with 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
1. chat_create_team_chat.csv
userid, teamid, TeamChatSessionID, timestamp
2. chat_item_team_chat.csv
userid, TeamChatSessionID, chatitemid, timestamp
3. chat_join_team_chat.csv
userid, TeamChatSessionID, timestamp
4. chat_leave_team_chat.csv
userid, TeamChatSessionID, timestamp
5. chat_mention_team_chat.csv
chatitemid, userid, timestamp
6. chat_respond_team_chat.csv
chatitemid1, chatitemid2,timestamp

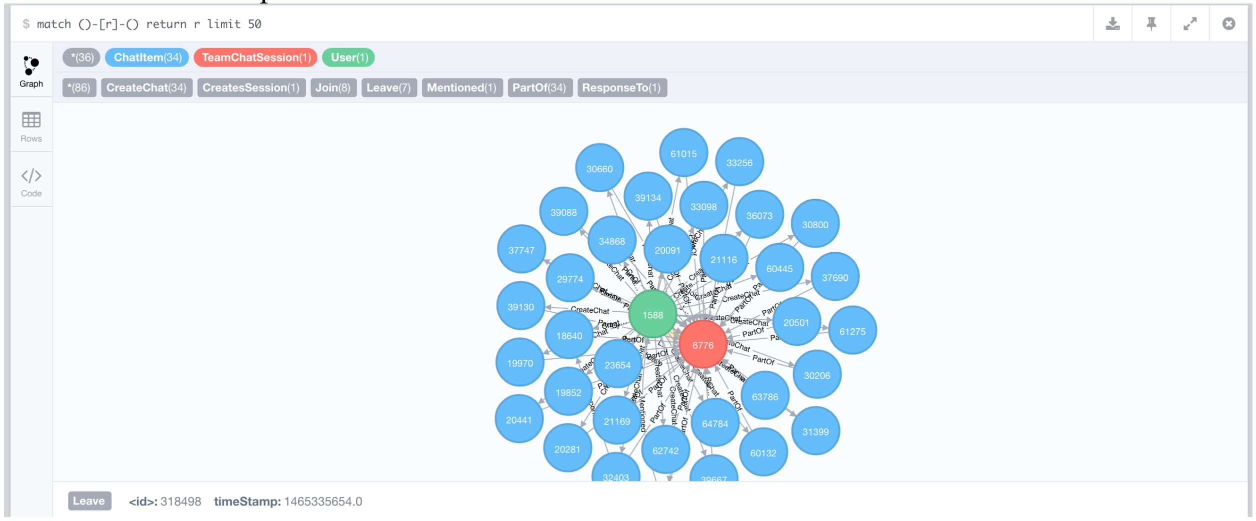
ii) Explain the loading process and include a sample LOAD command

```
# clear out the database
MATCH (n)
```

```
OPTIONAL MATCH (n)-[r]-()
DELETE n,r
 # create the constraint on nodes' primary key
CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE;
CREATE CONSTRAINT ON (t:Team) ASSERT t.id IS UNIQUE;
CREATE CONSTRAINT ON (c:TeamChatSession) ASSERT c.id IS UNIQUE;
CREATE CONSTRAINT ON (i:ChatItem) ASSERT i.id IS UNIQUE;
# load chat_create_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_create_team_chat.csv" AS row MERGE (u:User {id: toInt(row[0])})
MERGE (t:Team {id: toInt(row[1])}) MERGE (c:TeamChatSession {id:
toInt(row[2])}) MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c) MERGE
(c)-[:OwnedBy{timeStamp: row[3]}]->(t)

# load chat_join_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_join_team_chat.csv" AS row MERGE (u:User {id: toInt(row[0])}) MERGE
(c:TeamChatSession {id: toInt(row[1])}) MERGE (u)-[:Join{timeStamp: row[2]}]-
>(c)

# load chat_leave_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_leave_team_chat.csv" AS row MERGE (u:User {id: toInt(row[0])})
MERGE (c:TeamChatSession {id: toInt(row[1])}) MERGE (u)-[:Leave{timeStamp:
row[2]}]->(c)

# chat_item_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_item_team_chat.csv" AS row MERGE (u:User {id: toInt(row[0])})
MERGE (c:TeamChatSession {id: toInt(row[1])}) MERGE (i:ChatItem {id:
toInt(row[2])}) MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i) MERGE (i)-
[:PartOf{timeStamp: row[3]}]->(c)

# chat_mention_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_mention_team_chat.csv" AS row MERGE (i:ChatItem {id: toInt(row[0])})
MERGE (u:User {id: toInt(row[1])}) MERGE (i)-[:Mentioned {timeStamp:
row[2]}]->(u)



# chat_respond_team_chat.csv
LOAD CSV FROM "file:///Users/iBowen/Desktop/chat-
data/chat_respond_team_chat.csv" AS row MERGE (i:ChatItem {id: toInt(row[0])})
MERGE (j:ChatItem {id: toInt(row[1])}) MERGE (i)-[:ResponseTo {timeStamp:
row[2]}]->(j)
```

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.

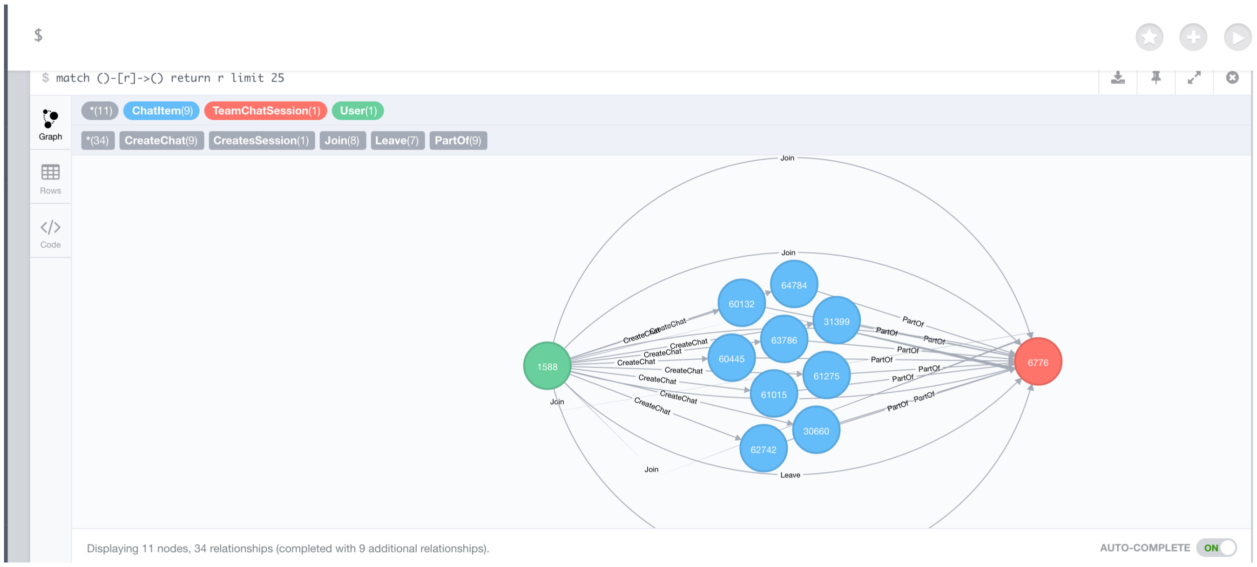
Show 50 relationships without direction



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

Leave <id>: 318498 timeStamp: 1465335654.0

Show 25 relationships with direction



```
$ match ()-[r]->() return r limit 25
```

Displaying 11 nodes, 34 relationships (completed with 9 additional relationships).

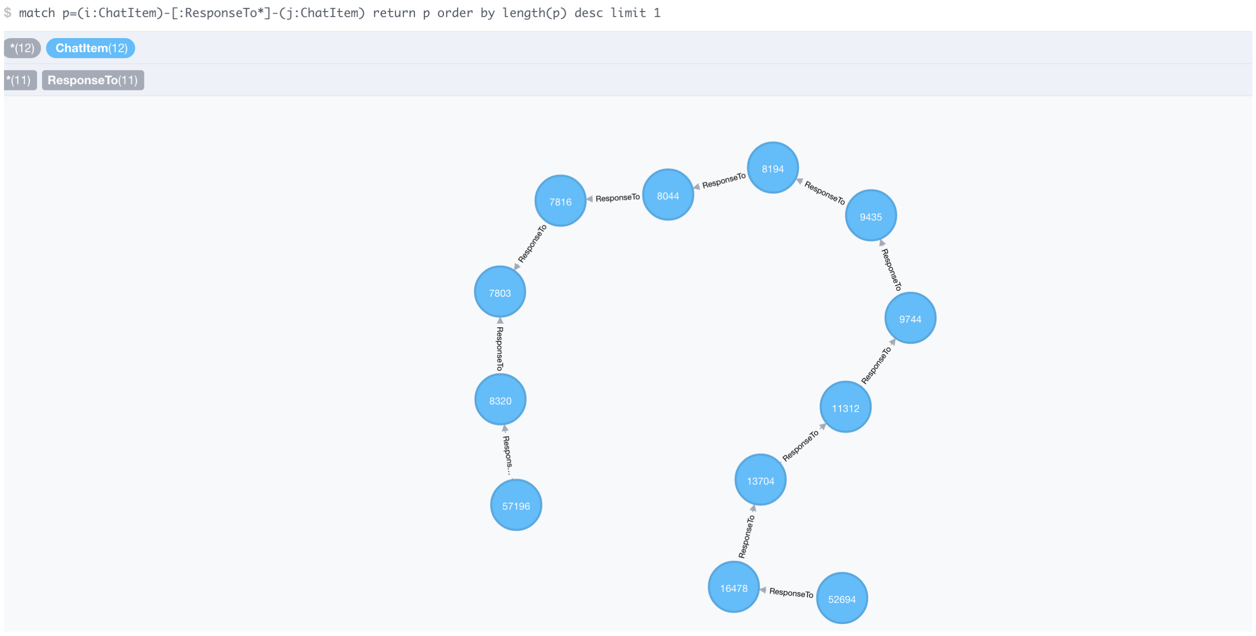
Finding the longest conversation chain and its participants

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.

# step 1, find the longest path with edge of "ResponseTo"

```
match p=(i:ChatItem)-[:ResponseTo*]-(j:ChatItem) return length(p) order by length(p) desc limit 1
# step 2, count the number distinct users who create ChatItem in the longest path
match p=(i:ChatItem)-[:ResponseTo*]-(j:ChatItem)
with p order by length(p) desc limit 1
match (u:User)-[:CreateChat]-(i)
where i in nodes(p)
return count(distinct u)
```

The longest path:



```
$ match p=(i:ChatItem)-[:ResponseTo*]-(j:ChatItem) return p order by length(p) desc limit 1
```

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

Users	Number of Chats
394	115
2067	111
209	109

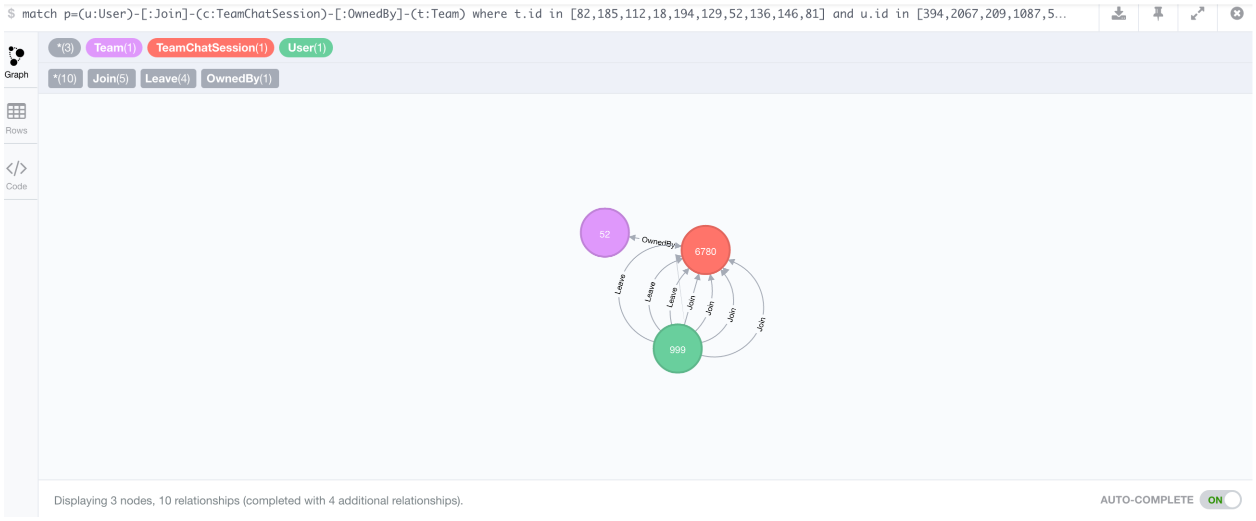
Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

Finally, present your answer.

match p=(u:User)-[:Join]-(c:TeamChatSession)-[:OwnedBy]-(t:Team) where t.id in [82,185,112,18,194,129,52,136,146,81] and u.id in [394,2067,209,1087,554,516,1627,999,668,461] return p

There is one chattest user id 999 who was part of one chattest team id 52



```
$ match p=(u:User)-[:Join]-(c:TeamChatSession)-[:OwnedBy]-(t:Team) where t.id in [82,185,112,18,194,129,52,136,146,81] and u.id in [394,2067,209,1087,5…
```

Displaying 3 nodes, 10 relationships (completed with 4 additional relationships).

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)

UserId	Coefficient
461	1
209	0.9523809523809523
516	0.9523809523809523

1. connect two users if One mentioned another user in a chat

    Match (u1:User)-[:CreateChat]->(i:ChatItem)-[:Mentioned]->(u2:User) merge (u1)-[:InteractsWith]->(u2)

2. connect two users if One created a chatItem in response to another user's chatItem

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

3. delete self-loop InteractsWith relationship

    match (u1)-[r:InteractsWith]->(u1) delete r

4. get neighbors and set degree based on the number of neighbors on the "InteractsWith" edge.

    match (u1:User)-[r:InteractsWith]-(u2:User) with u1, count(distinct u2) as degree set u1.deg = degree

return u1.id, u1.degreturn u1.id, u1.deg

5. get the number of links amongst neighbors, it includes the following steps:

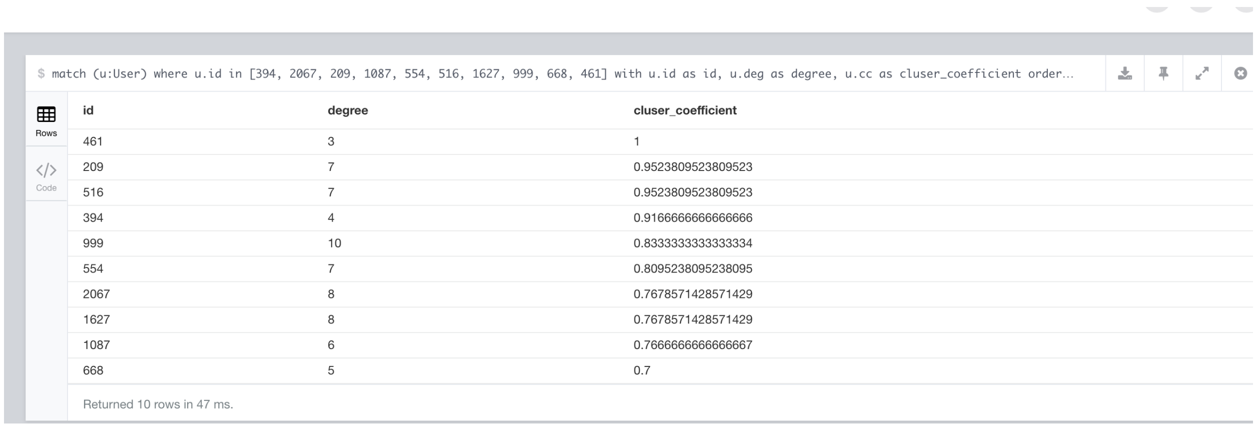
# find the neighbors users of one user node, and collect all distinct neighbor ids.

# find all links amongst neighbors

# count 1 if there are one or more relationships between two neighbors.

# show the user degree and add all links together
# calculate the clustering efficient

```
match (u1:User)-[:InteractsWith]-(u2:User)
with u1, collect(distinct u2.id) as neighbors
match (n:User)-[r:InteractsWith]->(m:User)
where (n.id in neighbors) and (m.id in neighbors)
with u1, case when (n)-->(m) then 1
else 0 end as value
with u1, u1.deg as deg, sum(value) as links
set u1.cc = toFloat(links) / (deg * (deg - 1))
return u1.id, u1.deg, u1.cc

match (u:User) where u.id in [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461]
with u.id as id, u.deg as degree, u.cc as cluser_coefficient order by cluser_coefficient desc
return id, degree, cluser_coefficient
```



```
$ match (u:User) where u.id in [394, 2067, 209, 1087, 554, 516, 1627, 999, 668, 461] with u.id as id, u.deg as degree, u.cc as cluser_coefficient order…
```

id	degree	cluser_coefficient
461	3	1
209	7	0.9523809523809523
516	7	0.9523809523809523
394	4	0.9166666666666666
999	10	0.8333333333333334
554	7	0.8095238095238095
2067	8	0.7678571428571429
1627	8	0.7678571428571429
1087	6	0.7666666666666667
668	5	0.7

Returned 10 rows in 47 ms.