Graph Analytics

Modeling Chat Data using a Graph Data Model

A graph is used to represent the chat data model because its composed of several entities that relationships among them, for example: 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.

- Vertices (Entities)
    - User
    - Team
    - TeamChatSession
    - ChatItem
- Edges (Relationships)
    - User creates TeamChatSession with timestamp
    - Team owns TeamChatSession with timestamp
    - User joins TeamChatSession with timestamp
    - User leaves TeamChatSession with timestamp
    - User creates ChatItem with timestamp
    - ChatItem is part of TeamChatSession with timestamp
    - ChatItem is mentioned by User with timestamp
    - 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

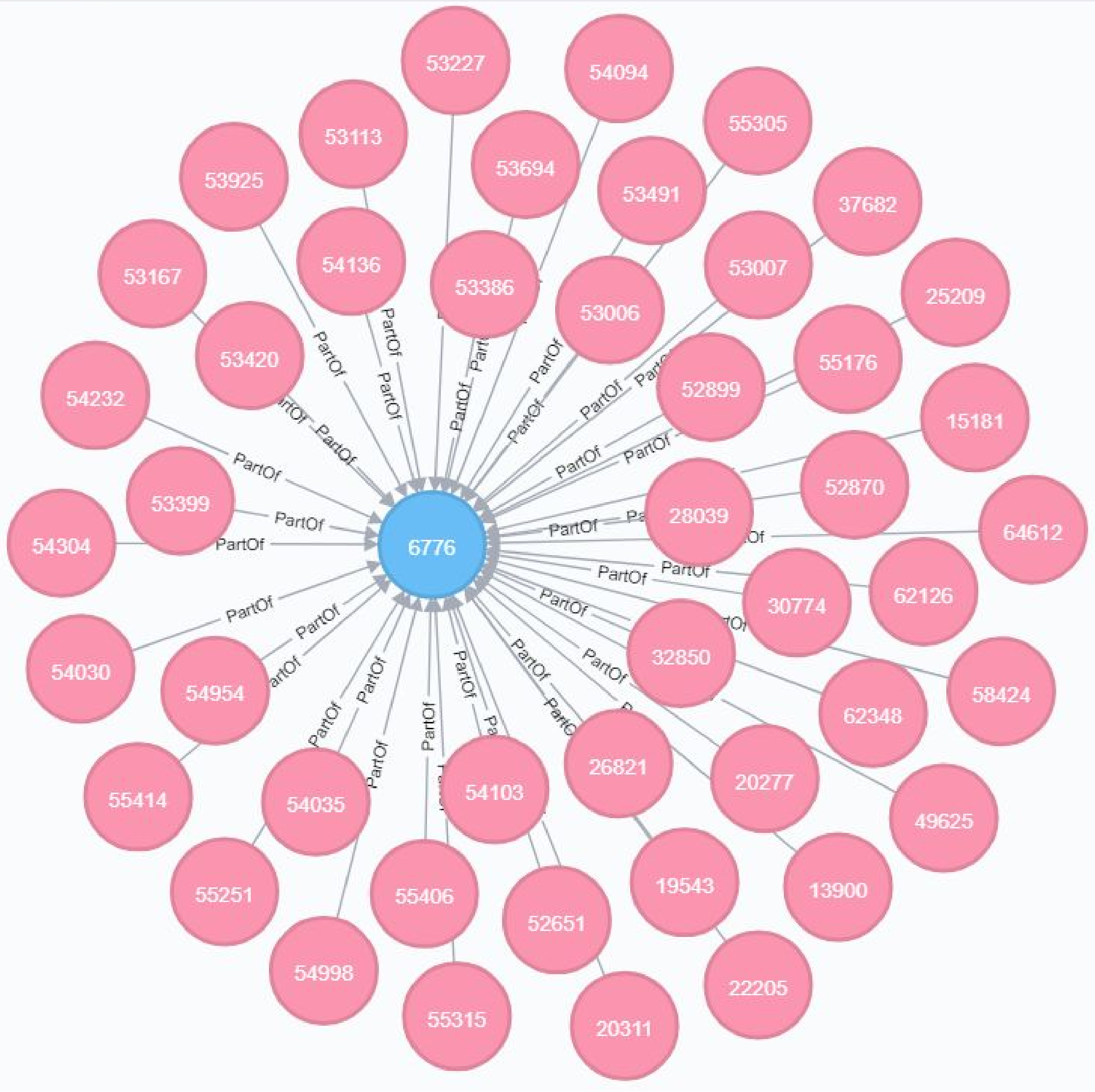
File Name	Fields	Description
chat_create_team_chat.csv	userID	the user id assigned to the user
	teamID	the id of the team
	teamChatSessionID	a unique id for the chat session
	timestamp	a timestamp denoting when the chat session created

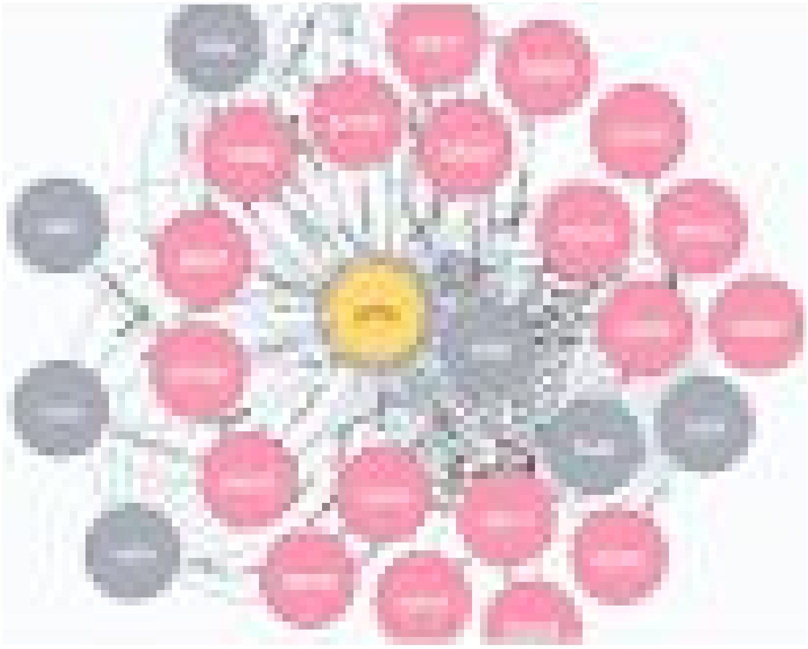
chat_item_team_chat.csv	userID	the user id assigned to the user
	teamChatSessionID	a unique id for the chat session
	chatItemID	a unique id for the chat item
	timestamp	a timestamp denoting when the chat item created
chat_join_team_chat.csv	userID	the user id assigned to the user
	teamChatSessionID	a unique id for the chat session
	timestamp	a timestamp denoting when the user join in 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 leave 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 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

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

```
 # Clear database
MATCH (n)
OPTIONAL MATCH (n)-[r]-()
DELETE n,r

 # Create the constraint 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)

# Load 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)

# Load 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)

# Load 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.


53227
54094
55305
53113
53694
53491
53925
37682
54136
53386
53007
53167
53006
25209
53420
52899
55176
54232
15181
53399
52870
28039
6776
54304
64612
62126
30774
32850
54030
54954
58424
62348
55414
54035
54103
26821
20277
49625
55251
55406
19543
13900
52651
54998
22205
55315
20311
PartOf



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.

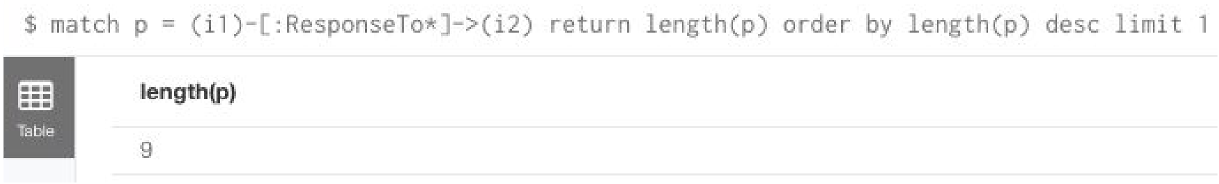
- Finding the longest conversation chain

match p = (i1)-[:ResponseTo*]->(i2)

return length(p)

order by length(p) desc limit 1

    - Answer:



```
$ match p = (i1)-[:ResponseTo*]->(i2) return length(p) order by length(p) desc limit 1
```

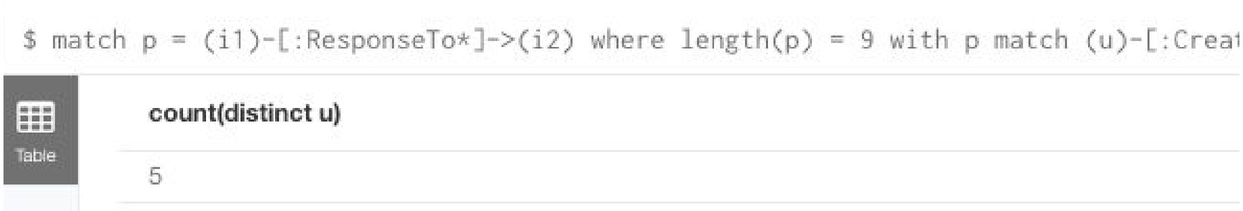
```
length(p)
9
```

- Unique users were part of this chain

match p = (i1)-[:ResponseTo*]->(i2)

where length(p) = 9

```
with p

match (u)-[:CreateChat]->(i)

where i in nodes(p)

return count(distinct u)
```



```
$ match p = (i1)-[:ResponseTo*]->(i2) where length(p) = 9 with p match (u)-[:Creat
```

count(distinct u)
5

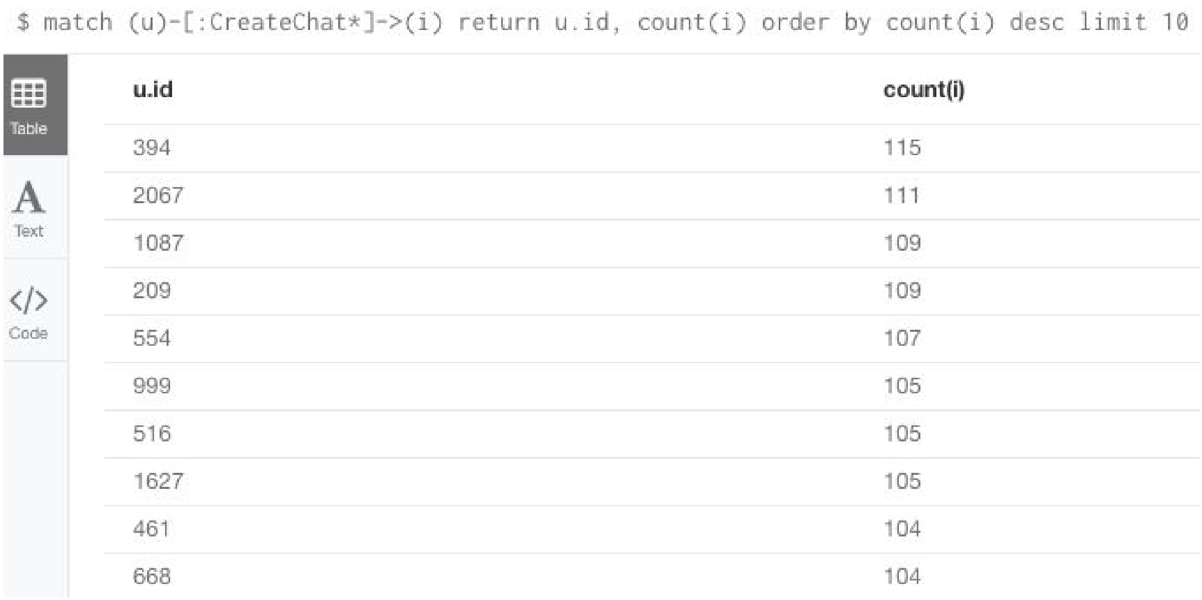
Analyzing the relationship between top 10 chattiest users and top 10 chattiest teams

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.

- Query

```
match (u)-[:CreateChat*]->(i)

return u.id, count(i)

order by count(i) desc limit 10
```

- Result



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

u.id	count(i)
394	115
2067	111
1087	109
209	109
554	107
999	105
516	105
1627	105
461	104
668	104

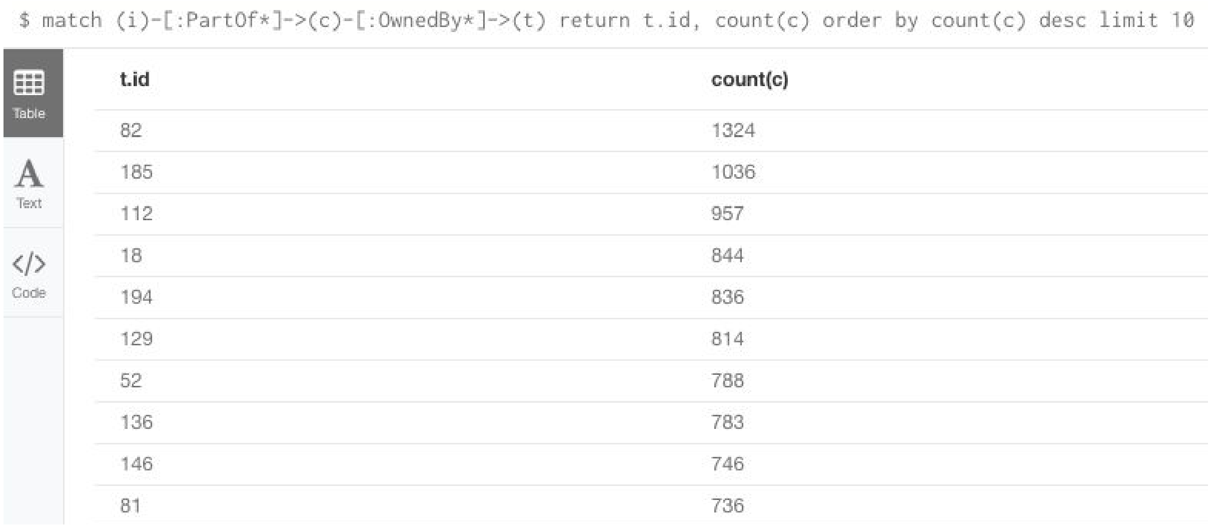
Chattiest Users

Users	Number of Chats
394	115
2067	111
209	109

- Query

```
match (i)-[:PartOf*]->(c)-[:OwnedBy*]->(t)

return t.id, count(c)

order by count(c) desc limit 10
```

- Result



```
$ 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

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.

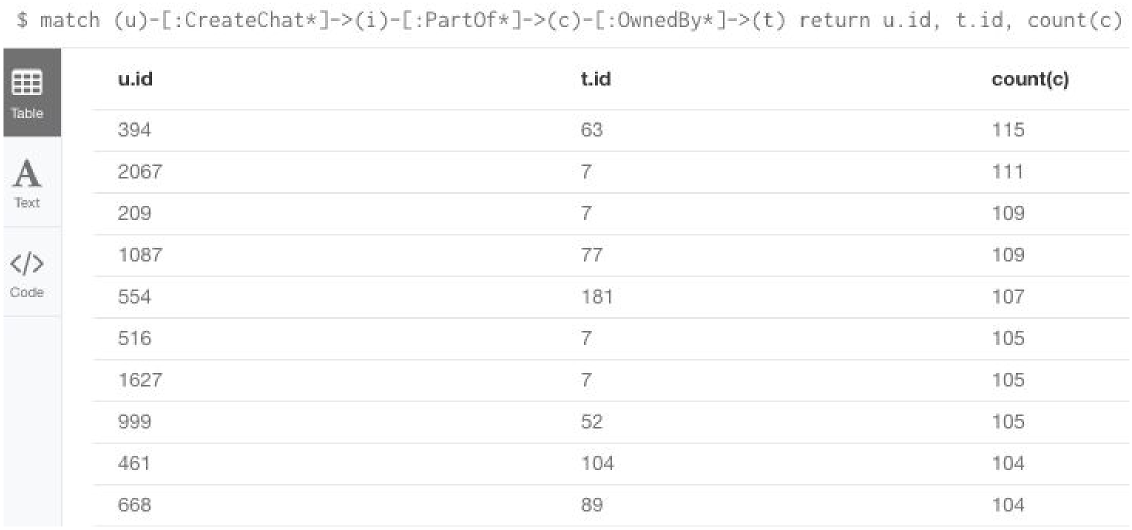
- Query

match (u)-[:CreateChat*]->(i)-[:PartOf*]->(c)-[:OwnedBy*]->(t)

return u.id, t.id, count(c)

order by count(c) desc limit 10

- Result



```
$ match (u)-[:CreateChat*]->(i)-[:PartOf*]->(c)-[:OwnedBy*]->(t) return u.id, t.id, count(c)
```

u.id	t.id	count(c)
394	63	115
2067	7	111
209	7	109
1087	77	109
554	181	107
516	7	105
1627	7	105
999	52	105
461	104	104
668	89	104

- Explanation

The user 999, which is in the team 52 is part of the top 10 chattiest teams, but other 9 users are not part of the top 10 chattiest teams. This demonstrates that most of the chattiest users are not in the 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.

- Constructing neighborhood of users using the following criteria: one mentioned another in a chat and one created a chatItem in response to another
    - Query:

```
match (u1:User)-[:CreateChat]->(i)-[:Mentioned]- >(u2:User) create (u1)-[:Deal]->(u2)
```

```
match (u1:User)-[:CreateChat]->(i1:ChatItem)- [:ResponseTo]- (i2:ChatItem) with u1, i1, i2 match (u2)-[:CreateChat]-(i2) create (u1)-[:Deal]->(u2)
```

- Removing self-loop
    - Query:

```
match (u1)-[r:Deal]->(u1) delete r
```

        - Removing self-loop from first query

```
match (u1:User)-[r1:Deal]->(u2:User) where u1.id <> u2.id with u1, collect(u2.id) as neighbors, count(distinct(u2)) as neighborAmount match (u3:User)-[r2:Deal]->(u4:User) where (u3.id in neighbors) AND (u4.id in neighbors) AND (u3.id <> u4.id) return u3.id, u4.id, count(r2)
```

- Removing duplicated edges count, counting unique entries
    - Query:

```
match (u1:User)-[r1:Deal]->(u2:User) where u1.id <> u2.id with u1, collect(u2.id) as neighbors,
count(distinct(u2)) as neighborAmount match (u3:User)-[r2:Deal]->(u4:User) where (u3.id in neighbors)
AND (u4.id in neighbors) AND (u3.id <> u4.id) return u3.id, u4.id, count(r2), case when count(r2) > 0
then 1 else 0 end as value
```

- Getting coefficient

```
match (u1:User)-[r1:Deal]->(u2:User) where u1.id <> u2.id with u1, collect(u2.id) as neighbors,
count(distinct(u2)) as neighborAmount match (u3:User)-[r2:Deal]->(u4:User) where (u3.id in neighbors)
AND (u4.id in neighbors) AND (u3.id <> u4.id) with u1, u3, u4, neighborAmount, case when (u3)-->(u4)
then 1 else 0 end as value return u1, (sum(value)/(neighborAmount*(neighborAmount-1))) as coeff
order by coeff desc limit 3
```

User ID	Coefficient
209	0,95
554	0,9
1087	0,8