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.)

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	Description	Fields
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

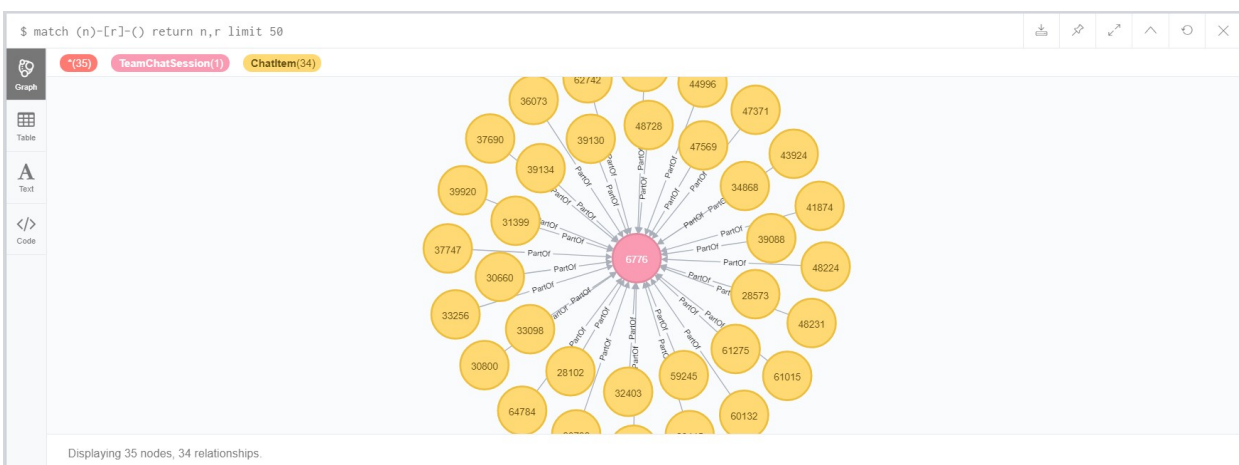
ii) Explain the loading process and include a sample LOAD command

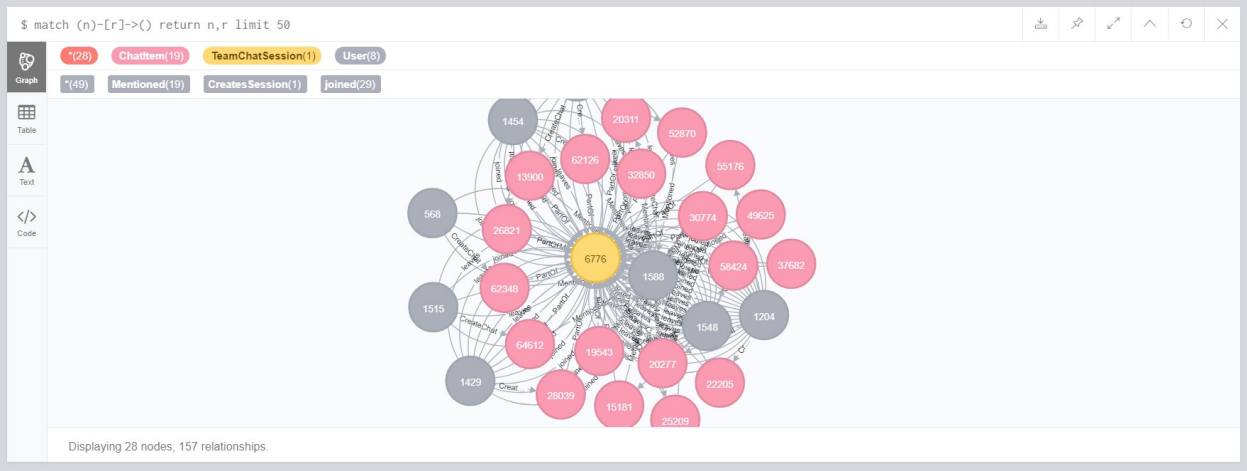
Copy data into the import folder inside the database folder the run queries like the following for each file

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

Line1: loads data from chat_join_team_chat .csv file
Line2: creates node tagged as User and has attribute id from csv file data column 0
Line3: creates nodes tagged as TeamChatSession and has attribute id from csv file data column 1
Line4: links User node to TeamChatSession with relation called joined with timestamp attribute loaded from csv column 2

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 has had some nodes moved to expose the edges more clearly. Both include examples of most node and edge types.





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

  Explanation:

  Get the longest path connect chatItem by getting all paths sort the in descending order the return the first (longest one).

- 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)

Explanation:

We know the longest conversation chain length the get the count of unique users.

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

- **Explanation:**

  Match users that connected to chats using CreateChat relation then return user id and count of chats created by each user id in descending order limited by 10.

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

"u.id"	"count(i)"
"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 371 ms and completed after 371 ms.

Chattiest Users

Users	Number of Chats
394	115
2067	111
209	109

- **Query:**

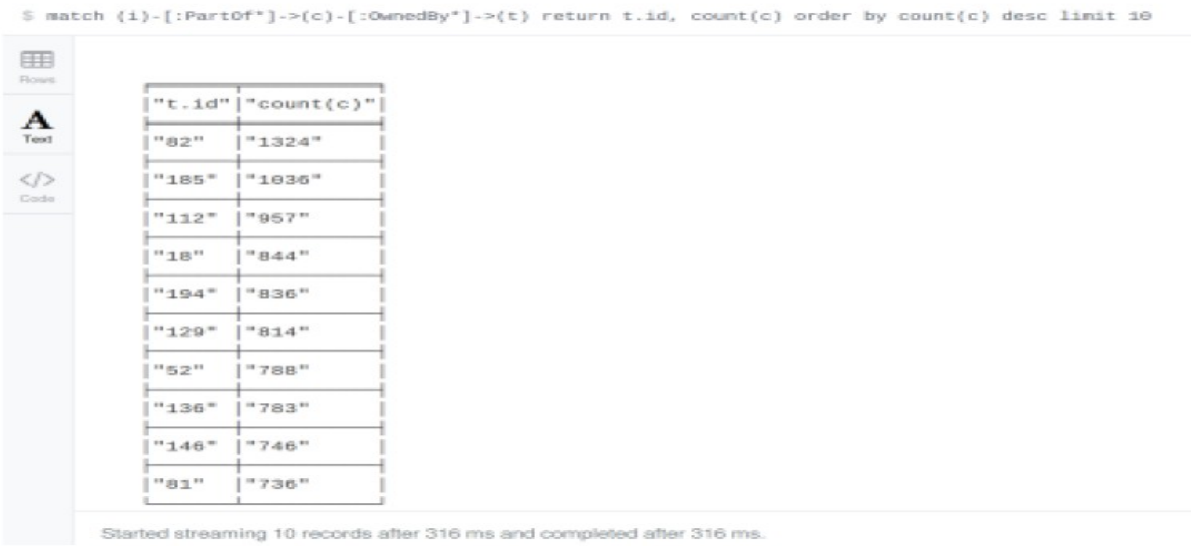
  match (i)-[:PartOf*]->(c)-[:OwnedBy*]->(t)

  return t.id, count(c)

  order by count(c) desc limit 10

- **Explanation:**

  Match chatItem that is part of TeamChatSession then match TeamChatSession that is ownedBy t then return team id and count of TeamChatSession in descending order limited to 10.



```
$ 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"

Started streaming 10 records after 316 ms and completed after 316 ms.

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

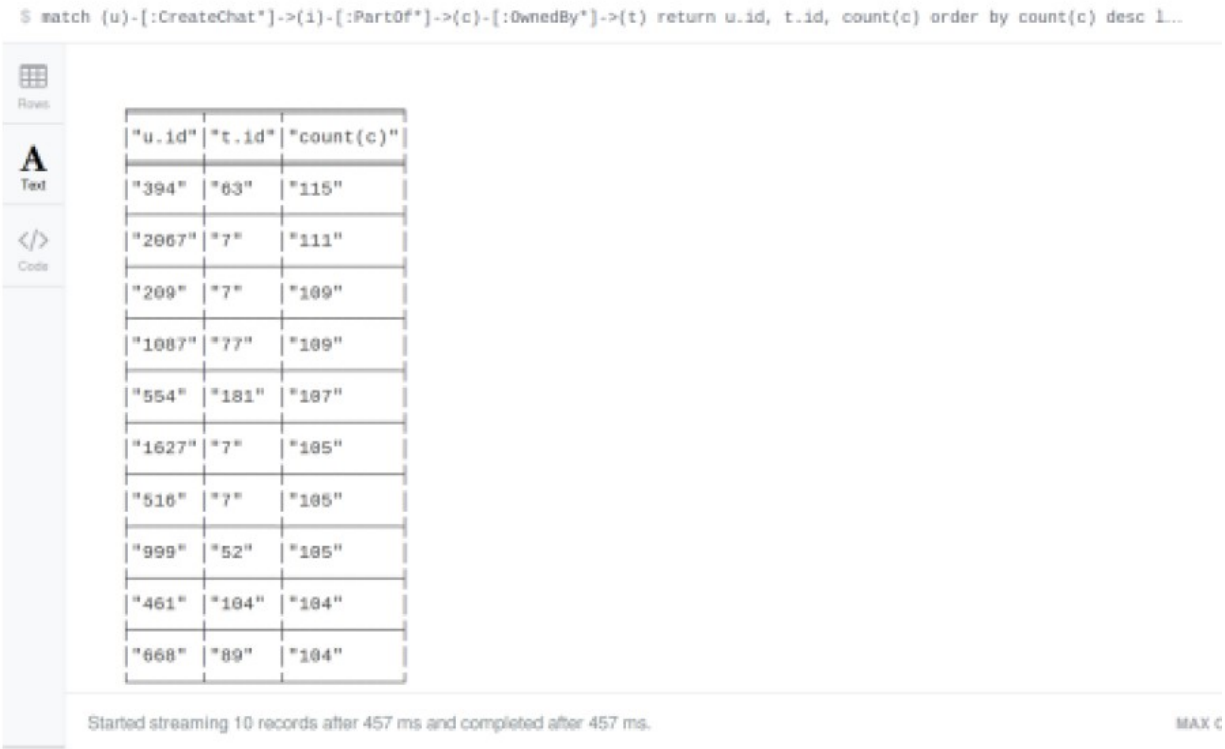
Finally, present your answer, i.e. whether or not any of the chattiest users are part of any of the chattiest teams.

By using the following query

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

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

order by count(c) desc limit 10



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

Users	Team	Count
394	63	115
2067	7	111
209	7	109

User 999 in team 52 is part of top 10 but the rest aren’t .

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.

a) We will construct the neighborhood of users. In this neighborhood, we will connect two users if
   - One mentioned another user in a chat
   - One created a chatItem in response to another user's chatItem
     - Queries:
       - 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)

b) The above scheme will create an undesirable side effect if a user has responded to her own chatItem, because it will create a self-loop between two users. So, after the first two steps we need to eliminate all self-loops involving the edge "Deal".

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

   remove them by the previous query

   - The following query will return the number of edges

     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)

- If one member has multiple edges with another member we need to count it as 1 because we care only if the edge exists or not.

```
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
```

c) The rest is to get the 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
```

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	.9524
554	.9048
1087	.8