**Graph Analytics**

**Modeling Chat Data using a Graph Data Model**

**The graph model is a network based on chat interactions between users.**

**A chat session can be initiated by a user, other users on the same team are able to join and leave the session.**

**Interactions between users begins when a user create a post.**

**it's possible for a user, mention another user.**

**All relationship between entities are logged with timestamp.**

**Creation of the Graph Database for Chats**

The schema of the 6 CSV files:

|  |  |
| --- | --- |
| **Files** | **Fields** |
| **chat\_create\_team\_chat** | **-userid**  **-teamid**  **-teamChatSessionID**  **-timestamp** |
| **chat\_item\_team\_chat** | **-userid**  **-teamchatsessionid**  **-chatitemid**  **-timestamp** |
| **chat\_join\_team\_chat** | **-userid**  **-teamChatSessionID**  **-teamstamp** |
| **chat\_leave\_team\_chat** | **-userid**  **-teamchatsessionid**  **-timestamp** |
| **chat\_mention\_team\_chat** | **-chatItem**  **-userid**  **-timeStamp** |
| **chat\_respond\_team\_chat** | **-chatid1**  **-chatid2**  **-timestamp** |

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)

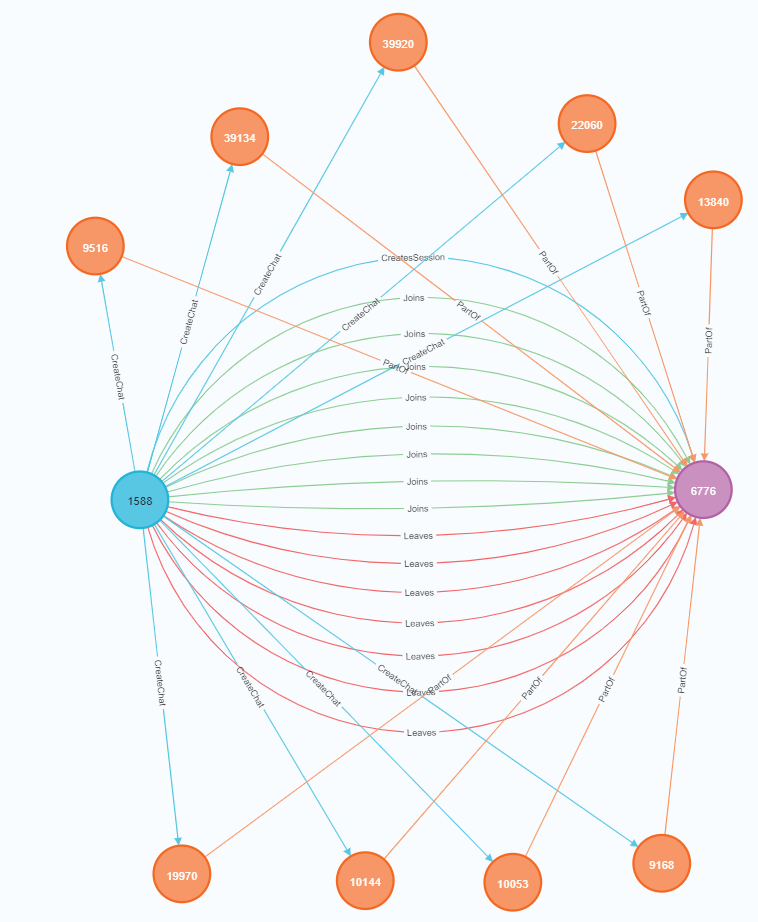
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)

**The first line imports the csv file from specific path. second, third and fourth lines creates nodes with different labels (User, Team, TeamChatSession) and have id properties. fifth and sixth lines create a relatiosship under lables specified like this:**

**Between User nodes and TeamChatSessionnodes called CreateSession**

**Between TeamChatSession nodes and Team nodes called OwnedBy.**

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 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:



**Finding the longest conversation chain and its participants**

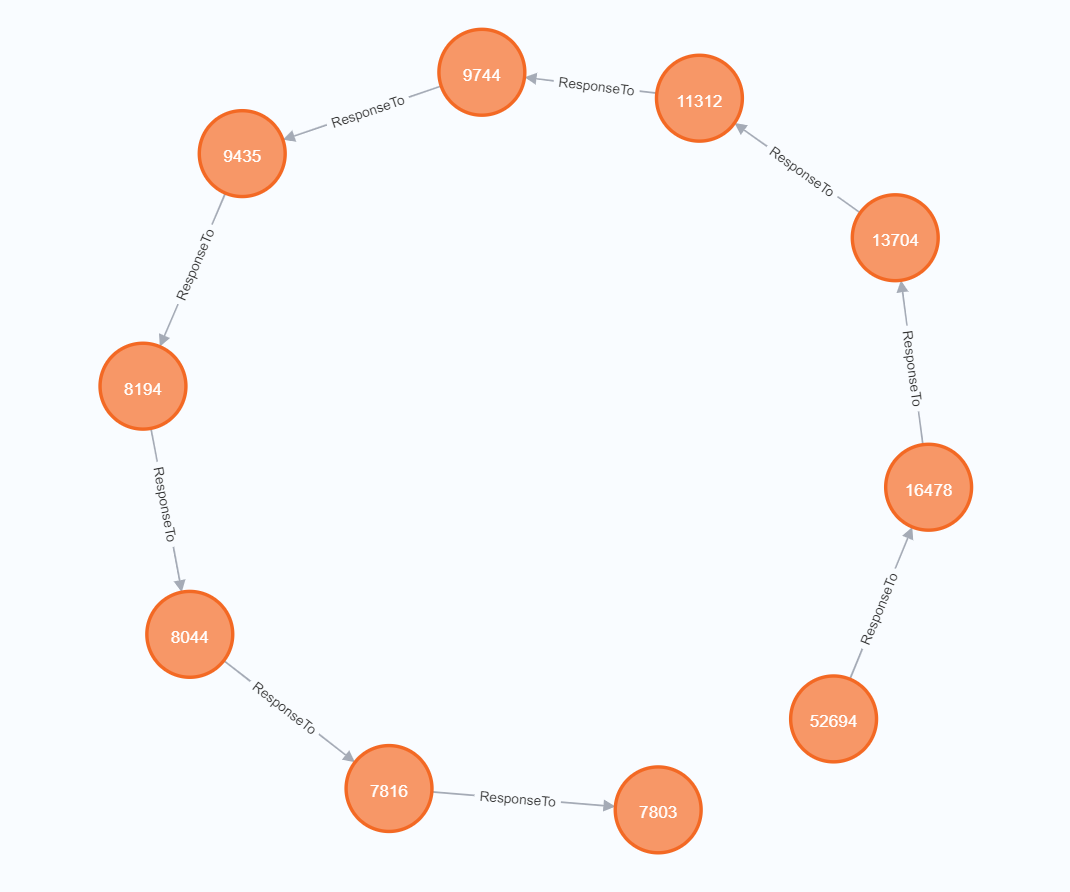
Report the results including the length of the conversation (path length) and how many unique users were part of the conversation chain. Describe your steps. Write the query that produces the correct answer:

**How many chats are involved between tow nodes?**

match p=(a)-[:ResponseTo\*]->(b)

return p,length(p)

order by length(p) desc limit 1



**The longest conversation chain in the chat data has path length (9) and there is (10) chats are involved on it.**

**How many Users are participated in this chain?**

match p=(c:ChatItem)-[:ResponseTo\*]->(j:ChatItem)

where length(p) = 9

with p

match q=(u:User)-[:CreateChat]-(c:ChatItem)

where (c in NODES(p))

return count(DISTINCT u)

**With (9) as longest path, count the number of distinct Users who create a ChatItem in the longest path and the query result is (5) 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.

**Chattiest Users:**

match (u:User)-[:CreateChat]-(i:ChatItem)

return u.id as Users , count(u.id) as Num\_Chats

order by count(u.id) desc limit 10

|  |  |
| --- | --- |
| **Users** | **Number of Chats** |
| **394** | **115** |
| **2067** | **111** |
| **1087** | **109** |
| **209** | **109** |
| **554** | **107** |
| **1627** | **105** |
| **999** | **105** |
| **516** | **105** |
| **668** | **104** |
| **461** | **104** |

**Chattiest Teams:**

match (:ChatItem)-[:PartOf]-(:TeamChatSession)-[:OwnedBy]-(t:Team)

return t.id as Teams , count(t.id) as Num\_Chats

order by count(t.id) desc limit 10

|  |  |
| --- | --- |
| **Teams** | **Number of Chats** |
| **82** | **1324** |
| **185** | **1036** |
| **112** | **957** |
| **18** | **844** |
| **194** | **836** |
| **129** | **814** |
| **52** | **788** |
| **136** | **783** |
| **146** | **746** |
| **81** | **736** |

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

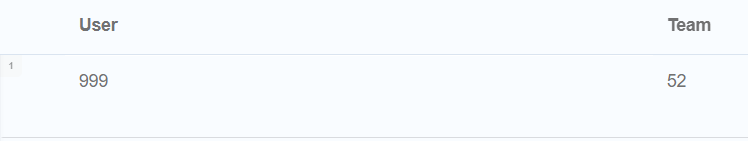
match (u:User)-[:CreateChat]-(:ChatItem)-[:PartOf]-(:TeamChatSession)-[:OwnedBy]-(t:Team)

where u.id in [394,2067,1087,209,554,1627,999,516,668,461]

and t.id in [82,185,112,18,194,129,52,136,146,81]

return distinct u.id as User , t.id as Team

**The result be like:**



**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.

**connect mention Users:**

match (u1:User)-[:CreateChat]->(:ChatItem)-[:Mentioned]->(u2:User)

merge (u1)-[:InteractsWith]->(u2)

**connect Users response with the chat creator:**

match (u1:User)-[:CreateChat]->(:ChatItem)-[:ResponseTo]-(:ChatItem)<-[:CreateChat]-(u2:User)

merge (u1)-[:InteractsWith]->(u2)

**Eliminate all self-interactions:**

match (u1)-[r:InteractsWith]->(u2) where u1.id=u2.id detach delete r

**Most Active Users (based on Cluster Coefficients)**

match (u1:User{id:394})-[:InteractsWith]->(u2:User)

with collect(u2.id) as neighbours, count(u2) as k

match (u3:User)-[iw:InteractsWith]->(u4:User)

where (u3.id in (neighbours)) and (u4.id in (neighbours))

return count(iw)/(k \* (k - 1) \* 1.0) as ClusterCoefficient

|  |  |
| --- | --- |
| **User ID** | **Coefficient** |
| **394** | **0.9167** |
| **2067** | **0.7679** |
| **209** | **0.9524** |