**Graph Analytics**

**Modelling Chat Data using a Graph Data Model**

Graph data model using to illustrate the chatting interaction among users with Chat Data. A user can create a chat session and create chat in the chat session. A user could be mentioned by a chat item and a chat item can response to another chat item. A user can join in an existed team chat session or leave it.

**Creation of the Graph Database for Chats**

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

1. **Detail for 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 |

1. **Explain the loading process and include a sample LOAD command**

Load process of a file in Neo4j starts with a LOAD command where you specify the file to load. Each row of the file is process by MERGE commands that creates nodes and relations, assigning values for attributes of these entities with the values of the columns of the row.

*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 CSV FROM "file:///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)*

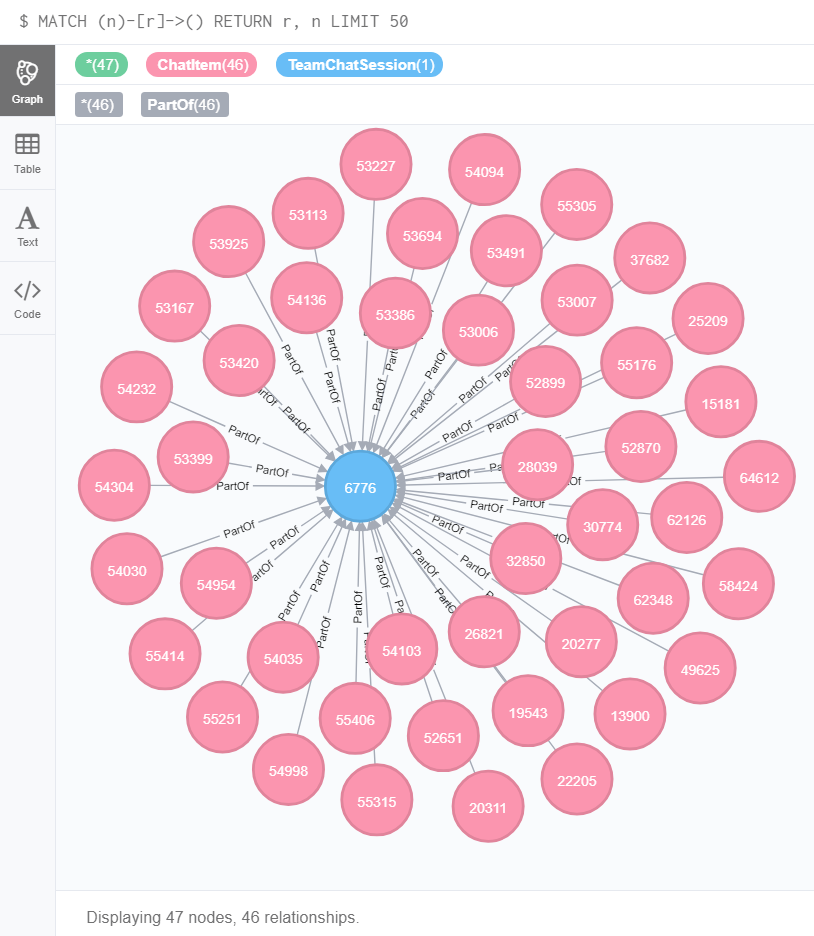
*MERGE (c)-[:OwnedBy{timeStamp: row[3]}]->(t)*

The first line gives the path of the file, this command reads the chat\_item\_team\_chat.csv at a time and create user nodes. The 0th column value is converted to an integer and is used to populate the id attribute. Similarly the other nodes are created.

Line 5, MERGE (u)-[:CreateChat{*timeStamp*: row[**3**]}]->(i) creates an edge labeled “CreateChat” between the User node u and the ChatItem node i. This edge has a property called timeStamp. This property is filled by the content of column 3 of the same row.

Line 6, MERGE (i)-[:PartOf{*timeStamp*: row[**3**]}]->(c) creates an edge labeled “PartOf” between the ChatItem node i and the TeamChatSession node c. This edge has a property called timeStamp. This property is filled by the content of column 3 of the same row.

**3. A screenshot of some part of the graph**



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

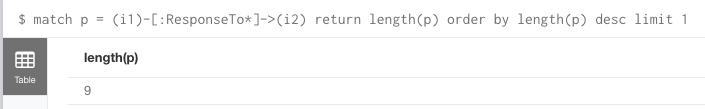
Find the longest conversation chain in the chat data using the "ResponseTo" edge label. This question has two parts

a. How many chats are involved in it?

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

return length(p)

order by length(p) desc limit 1

**Result: 9**

b. How many users participated in 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)

**Result: 5**

****

**Analysing 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.

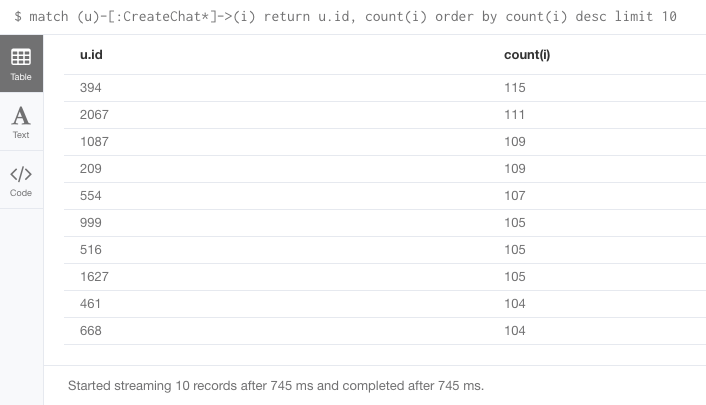
**Chattiest Users**

* Match the **CreateChat** edge from User node to **ChatItem** node, then return the **ChatItem** amount per user, and order by the amount in descending order.

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

return u.id, count(i)

order by count(i) desc limit 10

****

|  |  |
| --- | --- |
| **Users** | **Number of Chats** |
| 394 | 115 |
| 2067 | 111 |
| 209 | 109 |
| 1087 | 109 |

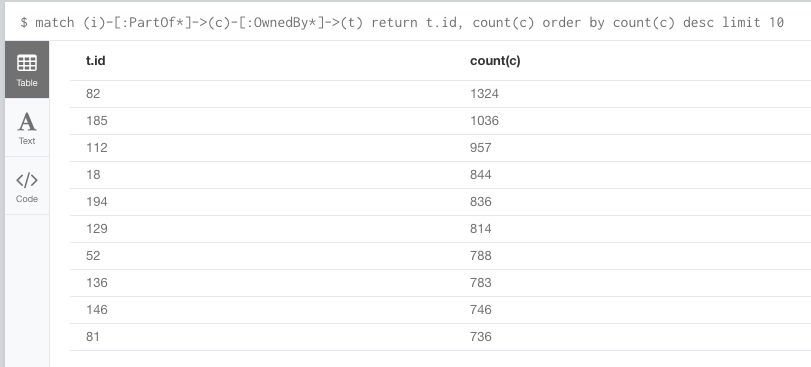
**Chattiest Teams**

* Match the PartOf edge from ChatItem node to TeamChatSession node, match the OwnedBy edge from TeamChatSession node to Team node, then return the TeamChatSession amount per team, and order by the amount in descending order.

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

return t.id, count(c)

order by count(c) desc limit 10

**

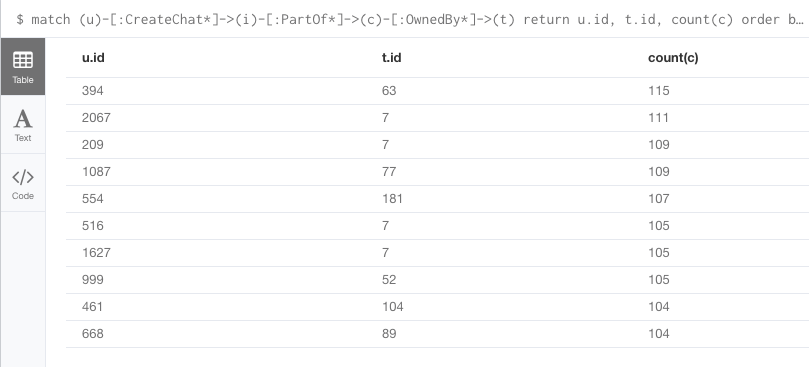
|  |  |
| --- | --- |
| **Teams** | **Number of Chats** |
| 82 | 1324 |
| 185 | 1036 |
| 112 | 957 |

* **Final result**

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

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

order by count(c) desc limit 10

****

User 999, which 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 states 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.

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

First you need to create InteractsWith relationship as described in instructions:

match (u1:User)-[:CreateChat]-(i:ChatItem)-[:Mentions]->(u2:User) create (u1)-[:InteractsWith]->(u2) match (u1:User)-[:CreateChat]->(i1:ChatItem)-[:ResponseTo]->(i2:ChatItem)<-[:CreateChat]-(u2:User) create (u1)-[:InteractsWith]->(u2) match (u1)-[r:InteractsWith]->(u1) delete r

After creating InteractsWith relationship, you can get the neighbors of each of the chattiest users with the following code:

match (u1:User{id:394})-[r:InteractsWith]-(u2:User) with u1,collect(distinct u2.id) as neighbors return u1.id,neighbors

394 [2011,1997,1012,1782]

The result is a set of nodes, in this case 4 users.

So, we have a neighbourhood of 5 users that interacts with each other. We can make the same query for each user in the neighborhood, changing the id:

match (u1:User{id:2011})-[r:InteractsWith]-(u2:User) with u1,collect(distinct u2.id) as neighbors return u1.id,neighbors

2011 [394,1997,1012,1782]

By repeating this query for each user in the neighborhood, you sum up the results and calculate the coefficient by dividing the sum with the number of possible interactions K\*(K-1). In this example, K=5 so we have 20 possible interactions in the neighborhood.

|  |  |
| --- | --- |
| **User ID** | **Coefficient** |
| 209 | 0.95 |
| 394 | 1,00 |
| 2067 | 0,93 |