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.)

This graph model based on chat data consists of nodes

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

And different edges for different relationship connecting two nodes like:

1. CreatChat,
2. CreatSession
3. Joins
4. Leaves
5. OwnedBy
6. PartOf
7. Mentioned
8. RespondTo

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:

Entity	Type	Column
userId	int	0
TeamId	int	1
TeamChatSessionId	int	2
timestamp	time	3

2. chat_item_team_chat.csv:

Entity	Type	Column
userId	int	0
TeamChatSessionId	int	1
chatId	int	2
timestamp	time	3

3. chat_join_team_chat.csv:

Entity	Type	Column
userId	int	0
TeamChatSessionId	int	1
timestamp	time	2

4. chat_leave_team_chat.csv

Entity	Type	Column
userId	int	0
TeamChatSessionId	int	1
timestamp	time	2

5. chat_mention_team_chat.csv

Entity	Type	Column
chatId	int	0
userId	int	1
timestamp	time	2

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

LOAD CSV is used to load the neo4j local server database.
Loading process is as follows:

1. These first 5 lines create the nodes and the rest load the files and create the edges.

```
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;
CREATE CONSTRAINT ON (j:ChatItem) ASSERT j.id IS UNIQUE;
```

2. Merge command fill up nodes and creates edges between nodes previously created. We are going to upload 6 files, which are kept in 'Neo4j/default.graphdb/import'

```
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)
```

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

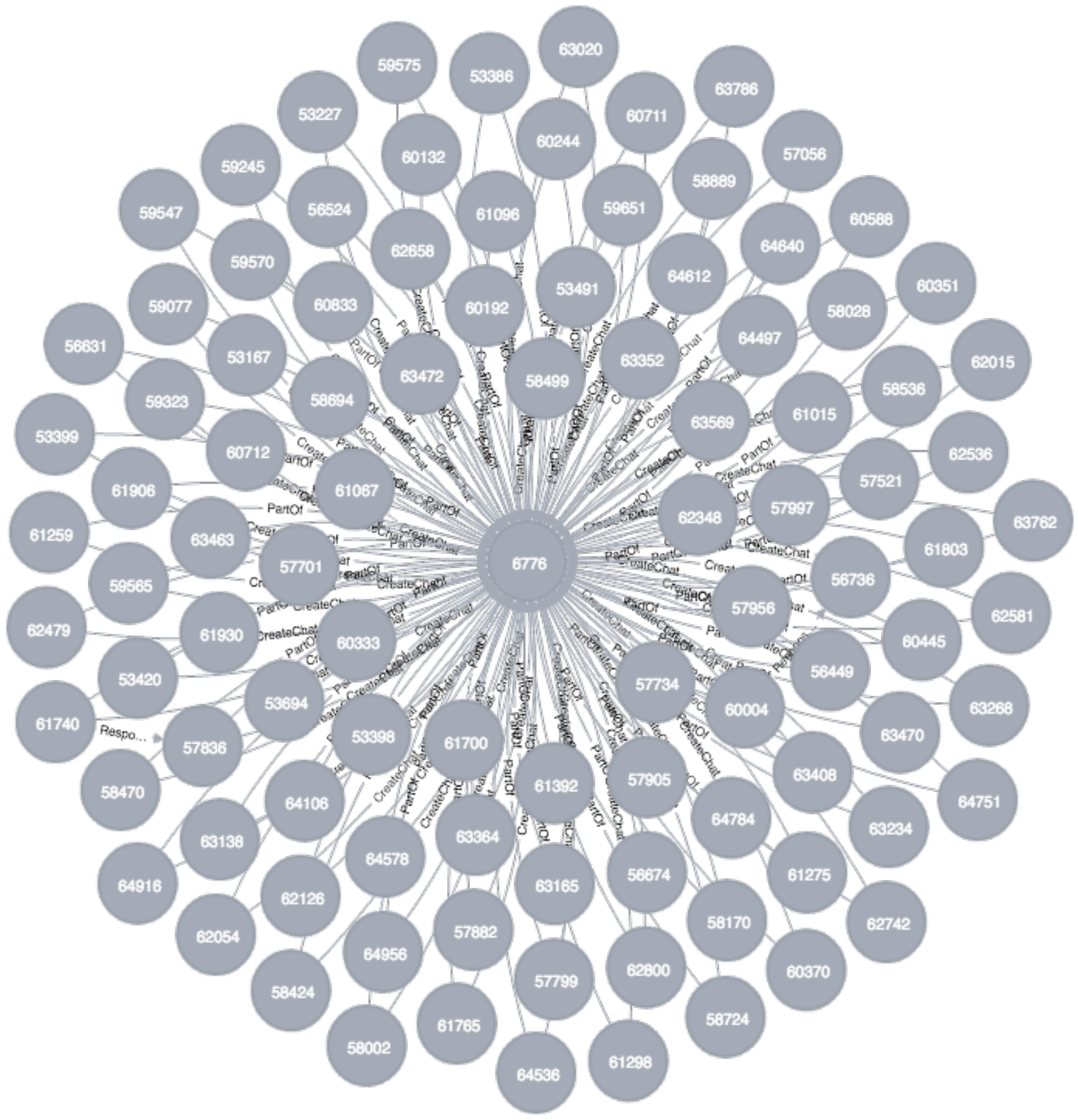
```
LOAD CSV FROM "file:/chat_leave_team_chat.csv" AS row
MERGE (u:User {id: toInt(row[0])})
MERGE (c:TeamChatSession {id: toInt(row[1])})
MERGE (u)-[:Leaves{timeStamp: row[2]}]->(c)
```

```
LOAD CSV FROM "file:/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 (i)-[:PartOf{timeStamp: row[3]}]->(c)
MERGE (i)-[:CreateChat{timeStamp: row[3]}]->(c)
```

```
LOAD CSV FROM "file:/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 CSV FROM "file:/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.



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.

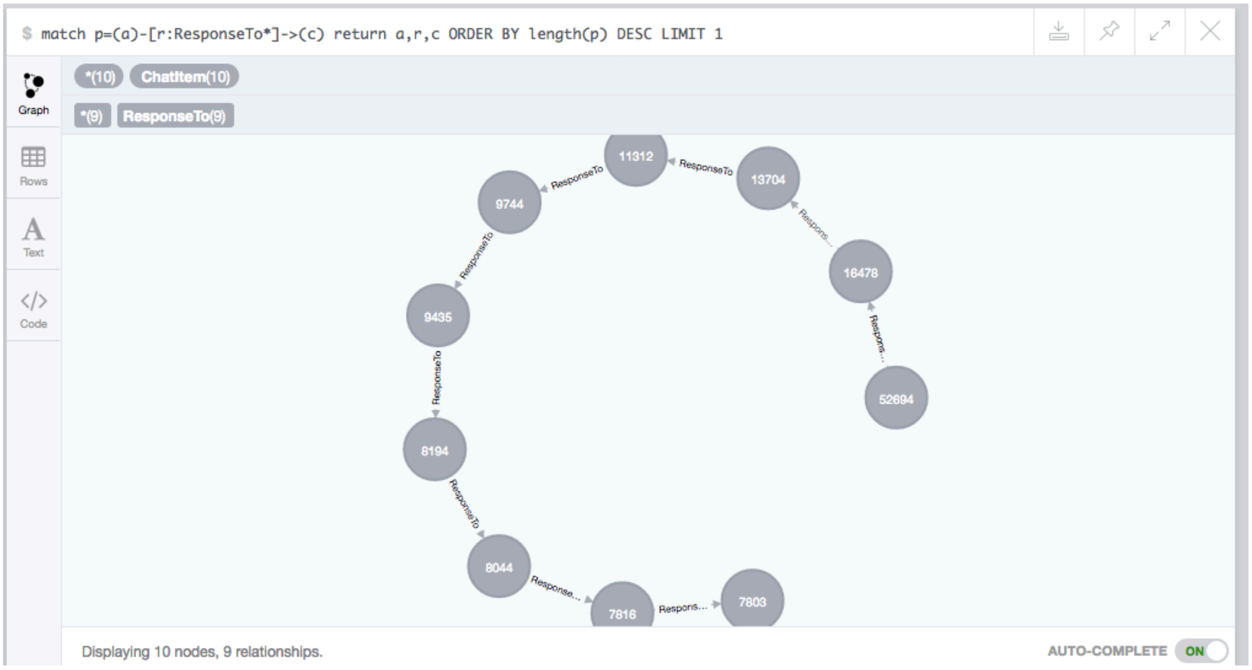
Query to path length

```
MATCH p=(a)-[r:ResponseTo*]->()
RETURN a,r ORDER BY length(p) DESC LIMIT 1
```

Path length = 9

Query to path plot

```
MATCH p=(a)-[r:ResponseTo*]->()
RETURN a,r ORDER BY length(p) DESC LIMIT 1
```



```
$ match p=(a)-[r:ResponseTo*]->(c) return a,r,c ORDER BY length(p) DESC LIMIT 1
```

*(10) ChatItem(10)

*(9) ResponseTo(9)

Displaying 10 nodes, 9 relationships.

AUTO-COMPLETE ON

Unique users involved in the conversation: 5

Query to unique users:

```
MATCH p=(a)-[:ResponseTo*]->()
WHERE length(p) = 9
WITH p  MATCH (u)-[:CreateChat]->(i)
WHERE i in nodes(p)
RETURN count(distinct u)
```

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

Process: We query users that connect to other user nodes with a CreateChat edge. The chattiest user is found by counting the outdegree of the node. We order the results in descending order and limiting to see top results.

```
MATCH (u)-[r:CreateChat*]-()
RETURN u, count(r) as Outdegree
ORDER BY Outdegree desc
LIMIT 10
```

Users	Number of Chats
394	115
2067	111
209,1087	109

Chattiest Teams

Process: To find the chattiest teams, **we** query teams that connect to TeamChatSession(c) nodes that in order connect to ChatItem(i) node. Then we compute the indegree of the team nodes, order the results in descending order and limit the top results

```
MATCH (i)-[r:PartOf*]->(c)-[o:OwnedBy]->(t)
RETURN t.id, COUNT(o) as Indegree
ORDER BY Indegree desc
LIMIT 10
```

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.

Process: To analyze if top chattiest users belong to chattiest teams, we first identified those chattiest user and team and put the restriction in each such graph by WHERE statement to find chattiest user in chattiest team

```
MATCH (u)-[: CreateChat]->(i)-[:PartOf]->(c)-[:OwnedBy]->(t)
WHERE u.id in [394, 2067, 209,1087,554,516,1627,999,668,461] and t.id in [82, 1085,
112,18,194,129,52,136,146,81]
RETURN DISTINCT u.id, t.id
```

Result: u.id = 999, t.id = 52 (i.e. Only one of the chattiest player 999 belongs to one of the Chattiest team 52)

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.

Process and Commands:

1. First we construct the neighborhood relationship:

```
MATCH (u1:User)-[:CreateChat]->(i)-[:Mentioned]->(u2:User)
CREATE(u1)-[:InteractsWith]->(u2)
MATCH (u1)-[:CreateChat]->(i1)-[:RespondTo]->(i2)
WITH u1, i2 MATCH (u2)-[:CreateChat]->(i2)
CREATE (u1)-[:InteractsWith]->(u2)
```

2. Then we delete all the loops

```
MATCH (u)-[r:InteractsWith]->(u) DELETE r
```

3. To compute the Clustering Coefficient we follow the steps:

- Obtain the neighbors of each of the top chattiest users
- Compute all distinct edges linking each neighbor with other neighbors
- Divide the number of distinct edges by K*(K-1) where K is the number of neighbors

```
MATCH (u)-[r:InteractsWith]-(v)
WHERE u.id in [394, 2067,1087,209,554,516,1627,999,668,461]
WITH u.id as user,collect(DISTINCT v.id) as  neighbors,
COUNT(DISTINCT v) as noOfNeighbors
MATCH (n)-[r:InteractsWith]-(m)
WHERE
n.id in neighbors
and
m.id in neighbors
WITH user as user, collect(DISTINCT [n.id,m.id]) as listOfEdges,
(noOfNeighbors *(noOfNeighbors-1)) as den
RETURN user,length(listOfEdges)*1.0 / den as  clusteringCoefficient
ORDER BY clusteringCoefficient DESC
```

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.95
554	0.90
1087	0.8