Graph Analytics

Modeling Chat Data using a Graph Data Model

The behaviour of users and their chats is modelled via graphs:
Users create TeamChatSessions, TeamChatSessions are owned by Teams, other Users Join or Leave TeamChatSessions, Users create ChatItems which are part of TeamChatSessions, ChatItems can be responses to other ChatItems, ChatItems may mention Users.

Creation of the Graph Database for Chats

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

i) The schema of the 6 CSV files

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

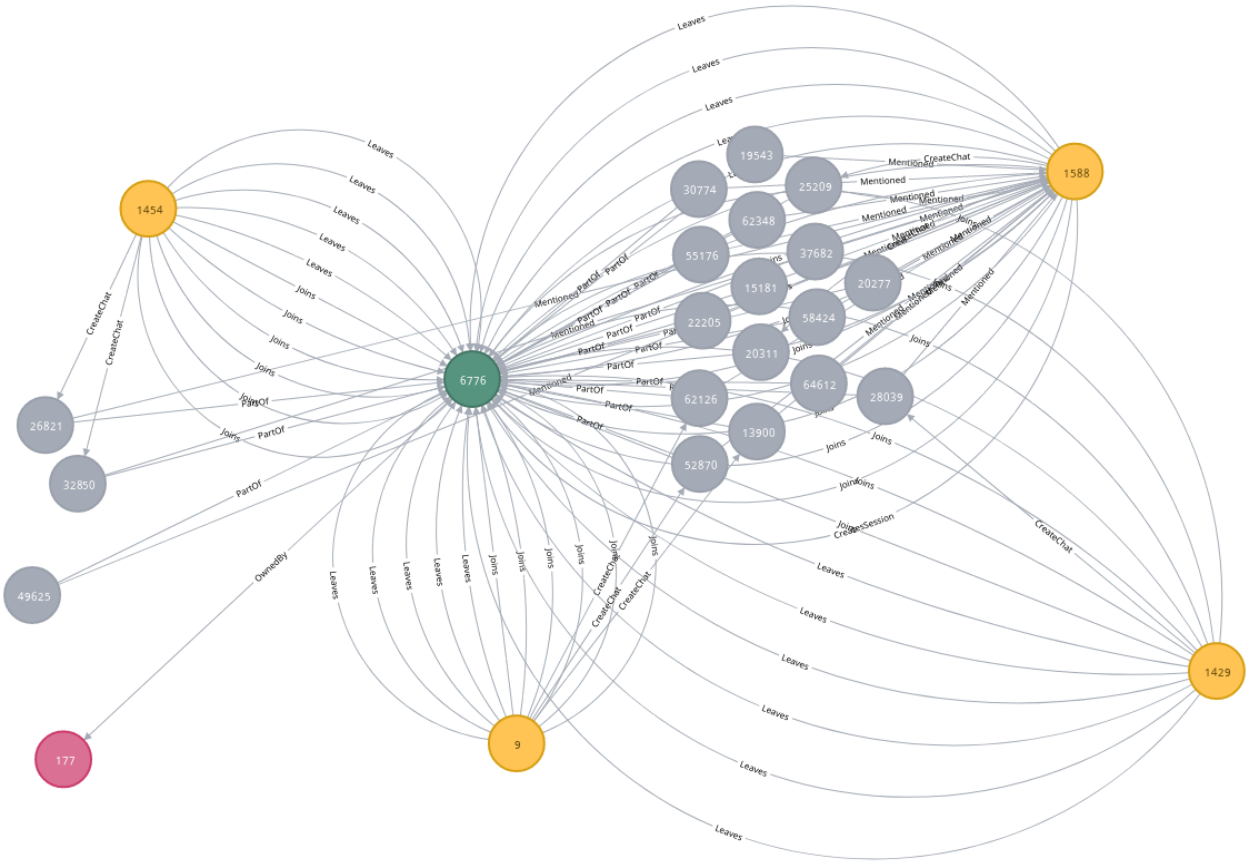
Chat_respond_team_chat:
chatid1, chatid2,timestamp

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

Load every row of the input CSVs as a 'row'. Create nodes or edges with required property values by indexing the values inside the row. For example

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

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.



Finding the longest conversation chain and its participants

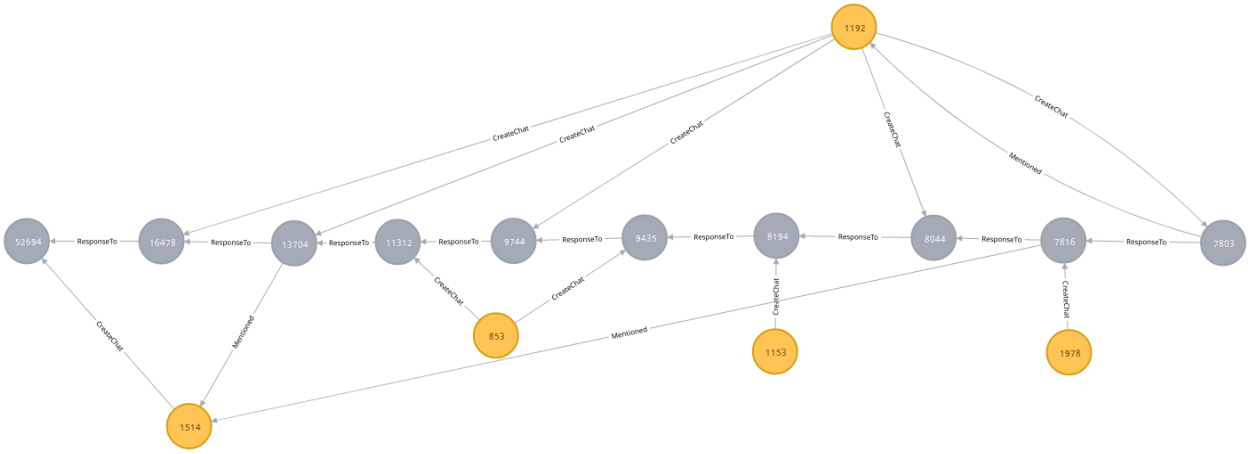
The longest Chain: 9 Responses after the 1 Original ChatItem

Find all paths between all ChatItems, let the maximum length be max_length. Find the path which is as long as max_length. Store its' ChatItem nodes in list named items. Return all distinct paths where User creates ChatItem that is in the items list.

Query:

```
MATCH p=(i1:ChatItem)-[:ResponseTo*]→(i2:ChatItem)
WITH max(length(p)) as max_length
MATCH p=(i1:ChatItem)-[:ResponseTo*]→(i2:ChatItem) WHERE length(p) = max_length
WITH [i in nodes(p)] as items
MATCH path=(u:User)-[:CreateChat]→(i:ChatItem)
WHERE i IN items
RETURN DISTINCT path
```

Longest Path:



The ChatItems of the Longest Path are marked in Grey, while the involved Users are Marked in Yellow. The no. of Users involved is 5.

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

Chattiest Users

Match User nodes with CreateChat edge, for every such node, find the count of CreateChat edges, Order them by Count but Descending, Limit our results to the top 3.

```
MATCH (u:User)-[c:CreateChat]→()
RETURN u.id as UserID, COUNT(c) as Chats
ORDER BY Chats DESC
LIMIT 3
```

Users	Number of Chats
394	115
2067	111
1087	109

Chattiest Teams

Match ChatItems that should be PartOf TeamChatSessions which should be OwnedBy Teams. For every such Team, find the count of ChatItems, Order them by Count but Descending, Limit our results to the top 3.

```
MATCH (i:ChatItem)-[:PartOf]→(:TeamChatSession)-[:OwnedBy]→(t:Team)
RETURN t.id as TeamID, COUNT(i) as Chats
ORDER BY Chats DESC
LIMIT 3
```

Teams	Number of Chats
82	1324
185	1036
112	957

We found that **One** of the top 10 Chattiest Users was in the top 10 Chattiest Teams.

```
MATCH (u:User)-[c:CreateChat]->()
WITH u, COUNT(c) as UserChats
ORDER BY UserChats DESC LIMIT 10
WITH [u.id] as ChattiestUsers

MATCH (u:User)-[:CreateChat]->(:ChatItem)-[:PartOf]->(:TeamChatSession)-[:OwnedBy]->(t:Team)
WHERE u.id IN ChattiestUsers
  AND t.id IN [82,185,112,18,194,129,52,136,146,81]
return DISTINCT u.id AS User, t.id as Team
```

"User"	"Team"
999	52

Only User 999 is a chatty User belonging to a Chatty Team(i.e Team 52).

How Active Are Groups of Users?

We will construct the neighborhood of users. In this neighborhood, we will connect two users if

- One user mentioned another user in a chat
- One user created a chatItem in response to another user's chatItem

Creating New Edges:

For the first condition, this query would have the following structure:

```
MATCH (u1:User)-[:CreateChat]→(:ChatItem)-[:Mentioned]→(u2:User)
CREATE (u1)-[:InteractsWith]→(u2)
```

Use the same logic to create the query statement for the second condition. This query will also have the form

```
MATCH (u1:User)-[:CreateChat]→(:ChatItem)-[:ResponseTo]→(:ChatItem)←[:CreateChat]-(u2:User)
CREATE (u1)-[:InteractsWith]→(u2)
```

So after the first two steps we need to eliminate all self loops involving the edge "Interacts with". This will take the form:

```
MATCH (u1)-[r:InteractsWith]→(u1) DELETE r
```

Part 1:

For Every TOP 10 Chatty User, we find the Neighbours and No. of Neighbours:

```
MATCH (u:User)-[c:CreateChat]→()
WITH u, COUNT(c) as Chats
ORDER BY Chats DESC LIMIT 10
WITH [u] as ChattiestUsers

//Getting the neighbours of all Users and the count
MATCH (u1:User)-[:InteractsWith]→(u2:User)
WHERE u1 in ChattiestUsers
WITH u1.id AS UserID, COLLECT(DISTINCT u2.id) AS Neighbours
RETURN UserID, Neighbours, SIZE(Neighbours) AS k
```

Output:

"UserID"	"Neighbours"	"k"
394	[1997,1012,2011,1782]	4
2067	[697,1672,63,209,1627,516,1265,2096]	8
1087	[426,772,929,1879,1311,1098]	6
209	[516,63,2067,1672,2096,1265,1627]	7
554	[2018,1687,1010,1959,1096,1412,610]	7
1627	[2096,1672,2067,63,516,209,697,1265]	8
516	[63,209,1672,2067,2096,1627,1265]	7
999	[1554,1587,778,1056,1606,1601,1398,1506,1839]	9
668	[2034,648,698,458]	4
461	[1675,1482,482]	3

Part 2:

Find Interecting Users such that both belong in Neighbours list, Finding Coefficient.

```
// Getting TOP 10 Chattiest Users
MATCH (u:User)-[c:CreateChat]→()
WITH u, COUNT(c) as Chats
ORDER BY Chats DESC LIMIT 10
WITH [u] as ChattiestUsers

// Getting the neighbours of TOP 10 Users and the count
MATCH (u1:User)-[:InteractsWith]→(u2:User)
WHERE u1 in ChattiestUsers
WITH u1.id AS UserID, COLLECT(DISTINCT u2.id) AS Neighbours
WITH UserID, Neighbours, SIZE(Neighbours) AS k

// Find Interecting Users
MATCH (u1:User)-[:InteractsWith]→(u2:User)
// Such that both belong in Neighbours list
WHERE u1.id IN Neighbours AND u2.id IN Neighbours
// For every valid combination of a User and its two neighbours,
// Value is 1 if neighbours have interacted atleast once, k is no. of Neighbours
WITH DISTINCT UserID, u1.id AS N1, u2.id as N2, CASE WHEN (u1)-[:InteractsWith]→(u2) THEN 1 ELSE 0 END AS VALUE, k

RETURN UserID,SUM(VALUE) as NUM,k*(k-1) AS DENUM, SUM(VALUE)/(k*(k-1)*1.0) AS ClusteringCoefficient
ORDER BY ClusteringCoefficient DESC
```

Output:

"UserID"	"NUM"	"DENUM"	"ClusteringCoefficient"
668	12	12	1.0
461	6	6	1.0
209	38	42	0.9047619047619048
516	38	42	0.9047619047619048
394	10	12	0.8333333333333334

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
668	1.0
461	1.0
209	0.9047619047619048