Graph Analytics

Modeling Chat Data using a Graph Data Model

In the graph model for chats, a user can create, join or leave a chat session while being a member of a team. In that chat session, a user can create chat items and be a part of a chat item. A user can also be mentioned in a chat item. Lastly, a chat item can respond to another chat item, which represents the communication between users.

Creation of the Graph Database for Chats

Part 4.1.1: Schema

The schema of the 6 CSV files which were used for creating the graph model is as follows:

chat_create_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamId	Contains the id of the team that the user is part of; format of the entries are all int
	TeamChatSessionId	Contains the id of the chat session the user is a part of; format of the entries are all int
	timestamp	Contains the timestamp that the user created the chat
chat_join_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamChatSessionId	Contains the id of the chat session the user is a part of; format of the entries are all int
	timestamp	Contains the timestamp that the user joined the team
chat_leave_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamChatSessionId	Contains the id of the chat session the user is a part of; format of the entries are all int
	timestamp	Contains the timestamp that the user left the team
chat_item_team_chat.csv	UserId	Contains the id of the user; format of the entries are all int
	TeamChatSessionI	Contains the id of the chat session the user is

|	d	a part of; format of the entries are all int
	ChatItemId	Contains the id of the chat item; format of the entries are all int
	timestamp	Contains the timestamp the user created the chat item; can also be used as timestamp that denotes the time the user became a part of the chat item
chat_mention_team_chat.csv	ChatItemId	Contains the id of the chat item; format of the entries are all int
	UserId	Contains the id of the user; format of the entries are all int
	timestamp	Contains the timestamp the user was mentioned in the chat item
chat_respond_team_chat.csv	ChatItemId	Contains the id of the chat item; format of the entries are all int
	ChatItemId	Contains the id of the chat item; format of the entries are all int
	timestamp	Contains the timestamp one chat item responded to a chat item

Part 4.1.2: Loading Process

To load the datasets we use the cypher language. Below is an example of how to load the file chat_create_team_chat.csv.

```
LOAD CSV FROM "file:///chat-data/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 row loads the csv file from the path under which the data files are saved and iterates through every row of the csv file as "row". It then creates the graph data by creating the nodes "User", "Team", and "TeamChatSession", and converting their type from String to Int. The indexing of the rows indicates that the first item (index zero) is the "User" node information, the second index (index one) is the "Team" node information, and the third index (index two) is the "TeamChatSession" information. Then, the edges are created by identifying which nodes they connect and including any additional attributes

for each edge. In the example above, we create edges "CreatesSession" with a "timeStamp" attribute which is stored in the fourth index (index three) of the "row". These edges connect nodes "User" and "TeamChatSession". Lastly, we create the edges "OwnedBy" with the same "timeStamp" attribute as the edge "CreatesSession", but these edges connect nodes "TeamChatSession" and "Team".

Part 4.1.3: Screenshots of the Graph
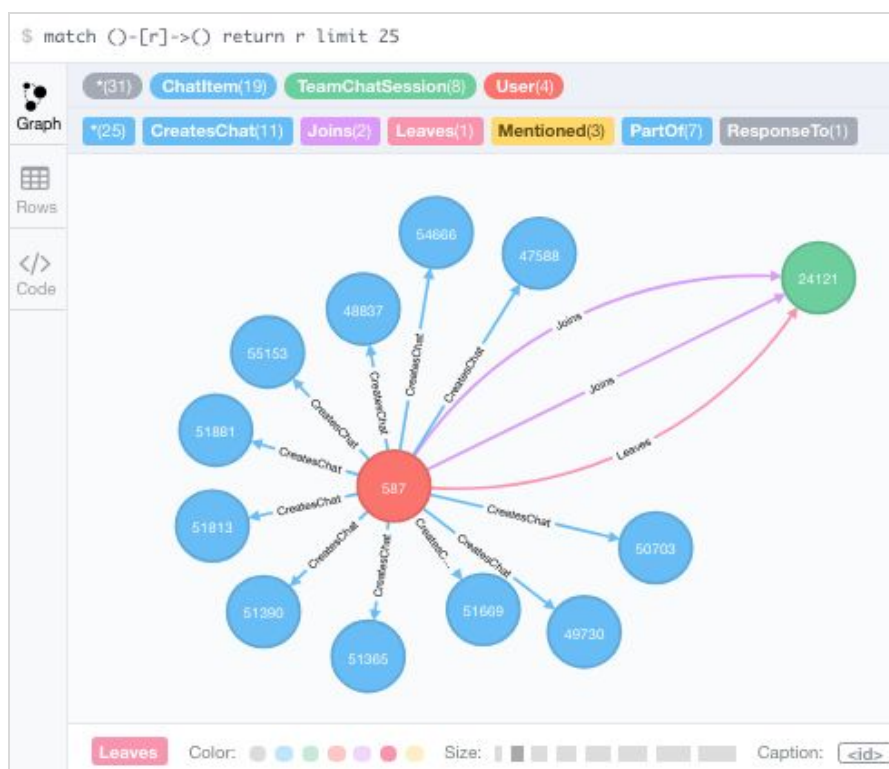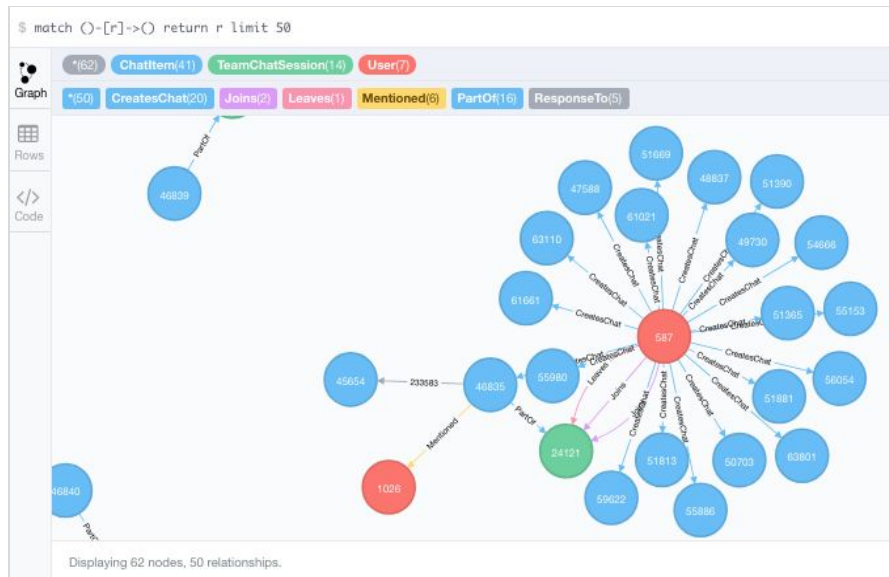
Finding the longest conversation chain and its participants

Part 4.2.1: Longest Conversation Chain

To find the longest conversation chain in the chat data using the "ResponseTo" edge label we can use the cypher script below to find the length of the edge chains with label "ResponseTo" and sort them by descending order so that the longest conversation chains are shown. In my script, I have limited the results to the top 5 to make sure that there are no two conversation chains with the same length.

```
match p = (ione)-[:ResponseTo*]->(itwo)
return length(p)
order by length(p) desc limit 5;
```


Part 4.2.2: Longest Conversation Chain Participants

The answer to the query, as seen above, is 9. This information helps to find out the participants of the longest query. To find the participants, we use the beginning of the query as in the example above and filter where the length of the edge chains is 9.

```
match p=(ione)-[:ResponseTo*]->(itwo)
where length(p)=9
with p
match (u)-[r:CreateChat]->(i)
where i in nodes(p)
return count(distinct u)
```

The nodes and edges information is saved in the variable "p". We then find all the users that have created a chat item where the chat item is part of the nodes that have been saved in variable p (the item chats that are part of the longest conversation chain). To get the number of participants, we return the count of the unique user nodes. Results can be seen in the image below:

```
1 match p=(a)-[:ResponseTo*]->(b)
2 where length(p)=9
3 with p
4 match (c)-[:CreateChat]->(d)
5 where d in nodes(p)
6 return count(DISTINCT c)
```

$ match p=(a)-[:ResponseTo*]->(b) where length(p)=9 with p match (c)-[:CreateChat]->(d) where d...

count(DISTINCT c)
5

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

Part 4.3.1: Top 10 Chattiest Users

First we find the top 10 chattiest users with the following cypher query:

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

We find all the user nodes that have created a chat item, therefore are connected to a chat item and return the user id attribute and count the item chats each user has created. We then order the results by showing the users that have higher numbers of chat items created and limiting the results to top 10. The table below shows the results.

$ 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

Returned 10 rows in 2206 ms.

Chattiest Users

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

Part 4.3.1: Top 10 Chattiest Teams

We then proceed to find the top 10 chattiest teams with the following cypher query:

```
match (i)-[:PartOf*]->(c)-[:OwnedBy*]->(t)
return t.id, count(c)
order by count(c) desc limit 10;
```

We find all the item chats that are part of team chat sessions that are owned by teams, order them by count of team chat sessions so that the highest team chat sessions will show up at the top and limit the display to top 10. The table below shows the results of the query:

$ 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

Returned 10 rows in 1004 ms.

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

Part 4.3.1: Results

To find whether the top 10 chattiest users belong to the top 10 chattiest teams, we can combine the two queries to find the teams the top 10 chattiest users belong to, and order the results by highest team chat sessions and only display top 10.

```
match (u)-[:CreateChat*]->(i)-[:PartOf*]->(c)-[:OwnedBy*]->(t)
return u.id, t.id, count(c)
order by count(c) desc limit 10;
```

The table below shows the results for this 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
999	52	105
516	7	105
1627	7	105
461	104	104
668	89	104

Returned 10 rows in 1712 ms.

It is evident that the only user that is one of the top 10 chattiest users who also belongs to the top 10 chattiest teams is user with id 999 who belongs to team with id 52. The rest of the users are not part of the top 10 chattiest teams.

How Active Are Groups of Users?

To answer this question, we look at how dense each neighborhood of group users is and use that information to capture how active the groups of users are.

Part 4.4.1: Create Edges "InteractsWith"
To identify the neighborhoods of group users we need to create a connection which will be named "InteractsWith" between users if:

1) One user mentioned another user in a chat item

The following query will create the edge with attribute "InteractsWith" if one user mentioned another used in a chat item:

```
match (u1:User)-[:CreateChat]->(i)-[:Mentioned]->(u2:User)
create (u1)-[:InteractsWith]->(u2);
```

2) One user created a chat item in response to another user's chat item

The following query will create the edge with attribute "InteractsWith" if one user created a chat item in response to another user's chat item:

```
match (u1:User)-[:CreateChat]->(i1:ChatItem)-[:ResponseTo]-(i2:ChatItem)
with u1, i1, i2
match (u2)-[:CreateChat]-(i2)
create (u1)-[:InteractsWith]->(u2);
```

The above query creates an undesirable side effect if a user has responded to their own chat item, because it will create a self loop between two users. So after, the query above, we will need to eliminate all self loops involving the edge "InteractsWith". This can be done through this query:

```
match (u1)-[r:InteractsWith]->(u1) delete r;
```

Part 4.4.2: Creating the Clustering Coefficient

To create the clustering coefficient -following the query below - we need to find all the users that are connected through the edge "InteractsWith" (line 1), that are not the same user (line 2) and that are part of the top chattiest users (line 3) and collect the user node information in a list called "neighbors" and the number of distinct nodes as "neighborCount" (line 4). Then, for the list of "neighbors", we collect number of edges "InteractsWith" (lines 5 and 6). For any pairs of neighbor nodes have more than one edge, we count them as just 1, while for any neighbor nodes that have no edges, we count it as 0 (lines 7, 8, and 9). Therefore, the command *sum(hasedge)* will have the total number of edges between neighbor nodes. We then use the formula on line 11 to calculate the coefficient as was indicated in the assignment and return the coefficients from highest to lowest.

```
match (u1:User)-[r1:InteractsWith]->(u2:User)
where u1.id <> u2.id
AND u1.id in [394,2067,1087,209,554,999,516,1627,461,668]
with u1, collect(u2.id) as neighbors, count(distinct(u2)) as neighborCount
match (u3:User)-[r2:InteractsWith]->(u4:User)
where (u3.id in neighbors) AND (u4.id in neighbors) AND (u3.id <> u4.id)
with u1, u3, u4, neighborCount,
case when count(r2) > 0 then 1
else 0
end as answer
return u1.id, sum(answer)*1.0/(neighborCount*(neighborCount-1)) as coeff
order by coeff desc limit 10;
```

Most Active Users (based on Cluster Coefficients rounded to 4 decimals)

User ID	Coefficient
209	0.9523
554	0.9047
1087	0.8