

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: Schema

6 CSV files which were used for creating the graph model, this is the schema:

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 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
	TeamChatSessionId	Contains the id of the chat session the user is 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 entries are all int
	timestamp	Contains the timestamp one chat item responded to a chat item

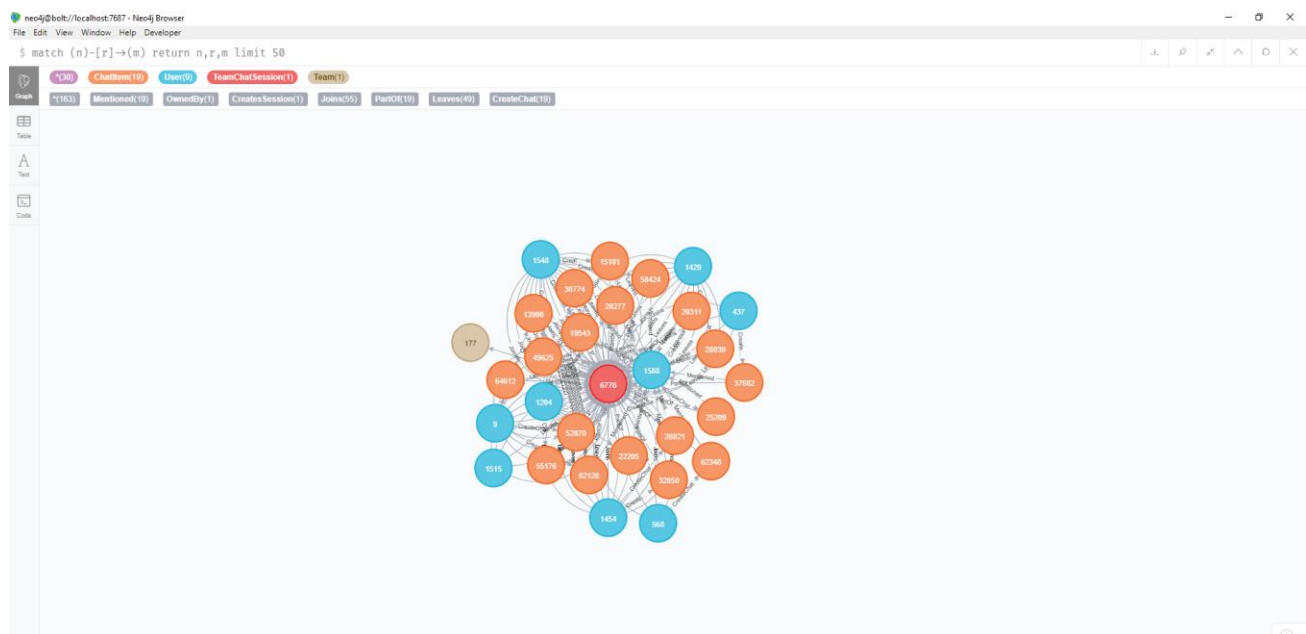
Part 4.2: Loading Process

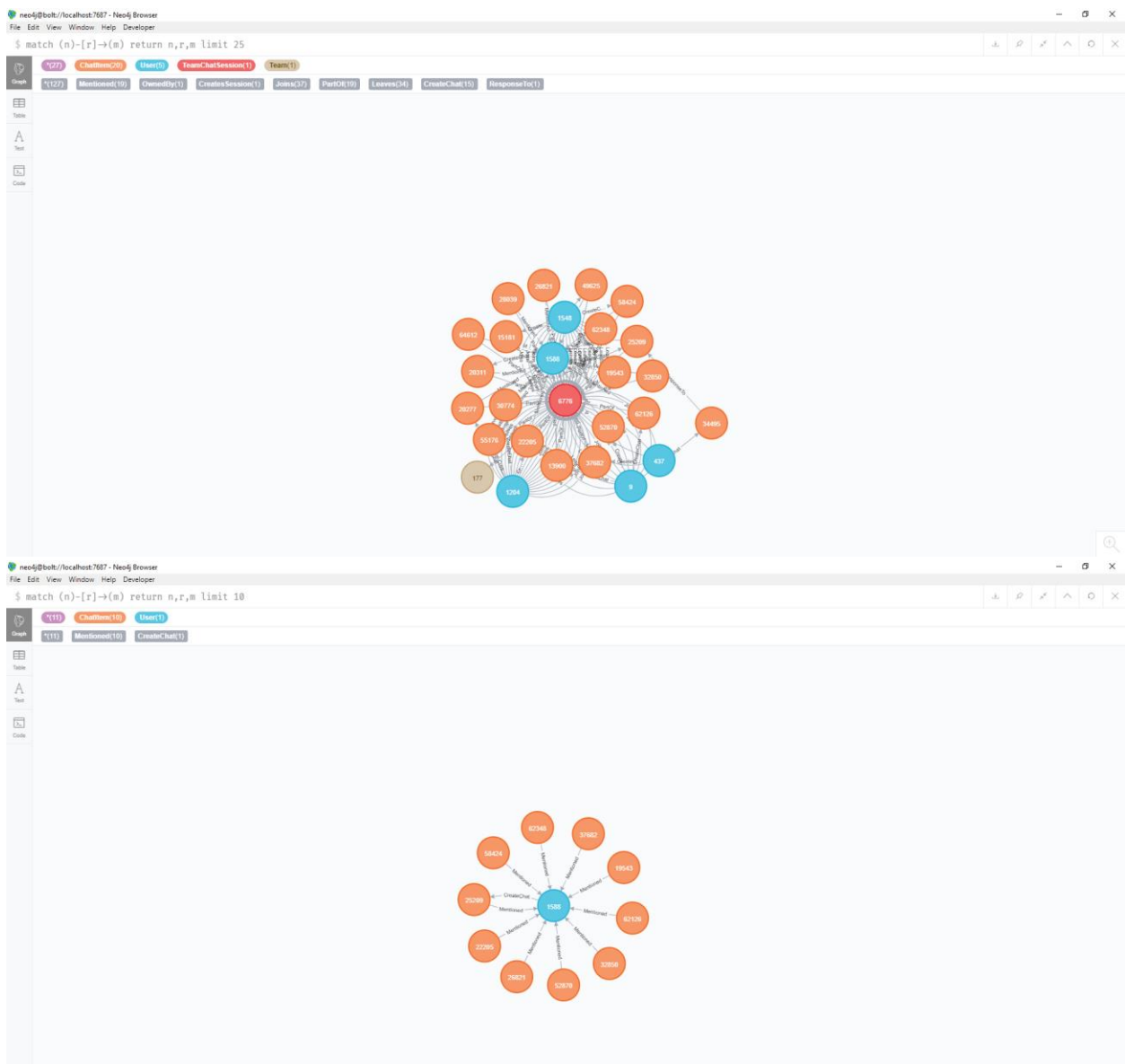
To load the datasets, we use the cypher language. In the image below we can see an example of how to load the file “chat_create_team_chat.csv”



The first line of command 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 allows build 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. Additionally, we create the edges by identifying which nodes they connect and including any additional attributes for each edge. In the example above, the edges created were "CreatesSession" with a "timeStamp" attribute which is stored in the fourth index (index three) of the "row". These edges connect nodes "User" and "TeamChatSession". Finally, we create the edges "OwnedBy" with the same "timeStamp" attribute as the edge "CreatesSession", these edges connecting the nodes "TeamChatSession" and "Team".

Part 4.2.1: Screenshots of the Graphs

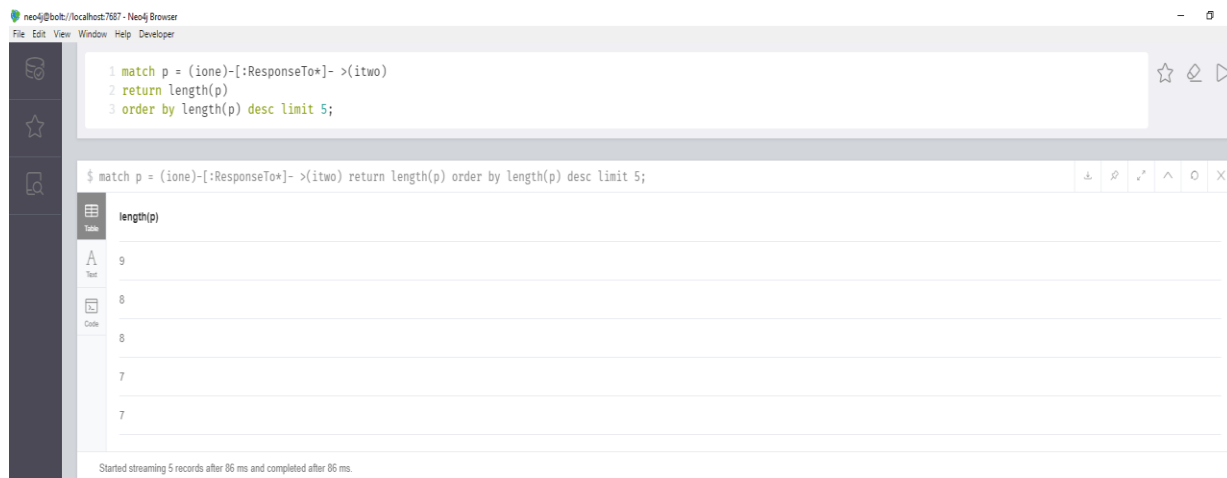




Finding the longest conversation chain and its participants

Part 4.3.1: Longest Conversation Chain

In order 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 the script, I’ve limited the results just to the top 5 to make sure that there are not two conversation chains with the same length.



Part 4.3.2: Longest Conversation Chain Participants

The query's answer, as seen above, is 9. It helps to find out the participants of the longest query. In order to find the participants, we can use the beginning of the query as in the example above and filter where the length of the edge chains is 9.

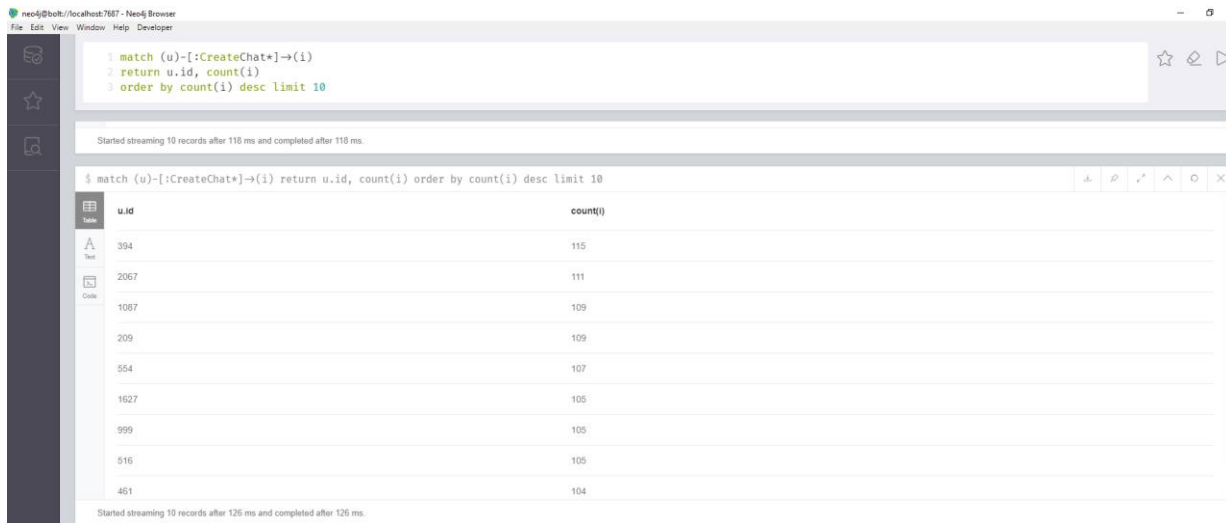
The information related to the nodes and edges is saved in the variable "p". We can find all the users that have created a chat item where the chat item is part of the nodes saved in "p" (the item chats that are part of the longest conversation chain). To get the number of participants, we return the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



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

Part 4.3.3: Top 10 Chattiest Users

First, we found the top ten chattiest users with the cypher query in the image below. We found all the user nodes that have created a chat item, consequently, are connected to a chat item and return the user id attribute and count the item chats each user has created. After that, we can order the results by showing the users that have a higher numbers of chat items created and limiting the results to top 10

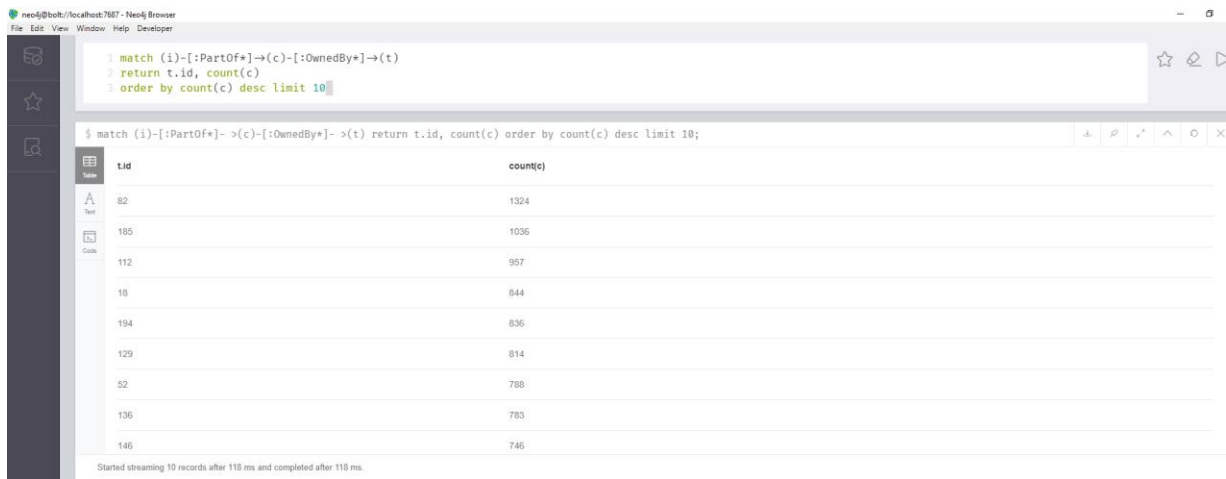


Chattiest Users

Users	Number of Chats
394	115
2067	111
209	109

Part 4.3.4: Top 10 Chattiest Teams

We found the top ten chattiest teams with cypher query in the upper part of the image. We found all the item chats that sessions so that the highest team chat sessions will be show up at the top and limit the display to top 10.

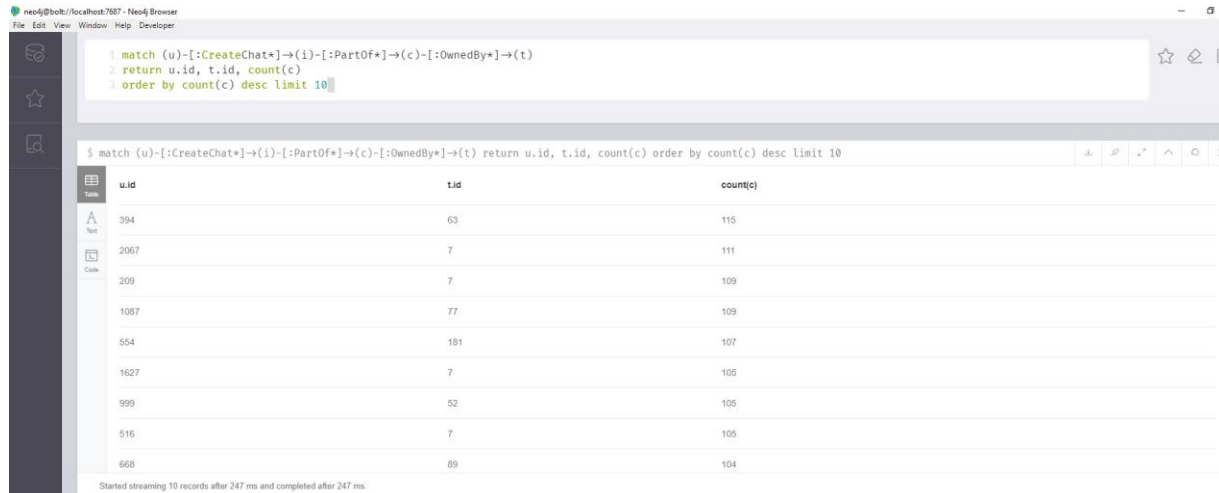


Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

Part 4.4: Results

We combined two queries to determine if the top ten chattiest users belong to the top ten chattiest teams. Just the user with the id 999 belonging to the team 52 is inside the top ten chattiest users and top ten chattiest teams.



The image shows the Neo4j Browser interface. The top panel contains the following Cypher query:

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

The bottom panel displays the results of the query in a table format:

u.id	t.id	count(c)
394	63	115
2067	7	111
209	7	109
1087	77	109
554	181	107
1627	7	105
999	52	105
516	7	105
668	89	104

Below the table, it states: "Started streaming 10 records after 247 ms and completed after 247 ms."

How Active Are Groups of Users?

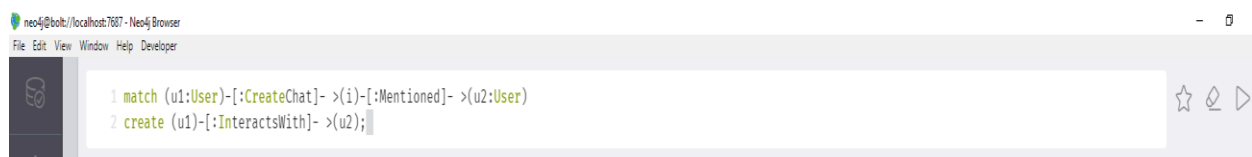
We need to look at how dense each neighborhood of group users is and use that information to capture how active the groups of users are.

Part 4.5: Create Edges “InteractsWith”

In order identify the neighborhoods of group users we need to create a connection which will be named “InteractsWith” between users if:

- a) One user mentioned another in a chat item

Creating the edge in the query with the attribute “InteractsWith” if one user mentioned another used in a chat item:



The image shows the Neo4j Browser interface with the following Cypher query:

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

- b) One user created a chat item in response to another user’s chat item

Creating the edge in the query with the attribute “InteractsWith” if one user created a chat item in response to another user’s chat item:



The image shows the Neo4j Browser interface with the following Cypher query:

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

This query above 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 applying this query, we will need to eliminate all the self-loops involving the edge “InteractsWith”. This could be done with this query:

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

Part 4.6: Creating the Clustering Coefficient

In order to create the clustering coefficient:

- 1.-We need to find all the users that are connected through the edge “InteractiveWith”(line 1)
- 2.- Those that aren’t the same user (line 2)
- 3.- Those that are part of the top chattiest users (line 3)
- 4.- Collect the user node information in a list call “neighbors” and the number of distinct nodes as “neighborCount” (line 4)
- 5.- For the list “neighbors” we must collect number of edges “InteractsWith” (lines 5 & 6).
- 6.- We are going to count as 1 any pair of neighbor node, while any neighbor node with no edge, we going to count it as 0 (lines 7, 8, 9)
- 7.- The command sum(hasedge) will provide the total number of edges between neighbor nodes
- 8.- Line 11 calculate the coefficient as was indicated in the assignment and return the coefficients from highest to lowest

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

u1.id	coeff
668	1.0
461	1.0
209	0.9523809523809523
516	0.9523809523809523
394	0.9166666666666666
999	0.8194444444444444
554	0.8095238095238095
2067	0.7678571428571429
1627	0.7678571428571429

Started streaming 10 records after 151 ms and completed after 151 ms.

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.9523
554	0.9047
1087	0.7666