Graph Analytics

Modeling Chat Data using a Graph Data Model

In the graph model for the game, a user in a team can create a chat session for the team he belongs to. Each team member can also create a new chat post (chatItem) for the specific session, and as the chat session extends, user can respond or also can be mentioned for other chat post from other users in same team. Users at any point in time can join or leave teams as well.

Creation of the Graph Database for Chats

1. The database for the graph analytics exercise consist of 6 .csv files as we detailed in the table below.

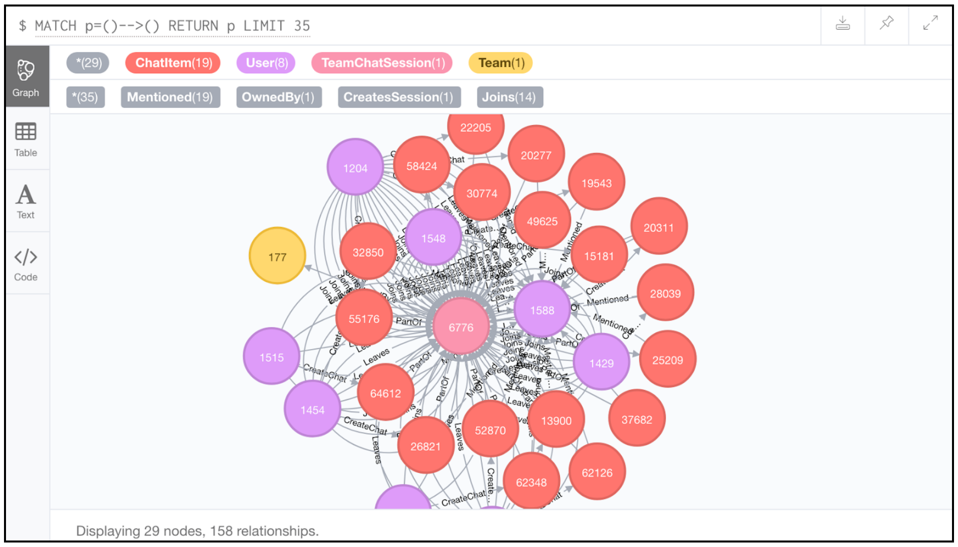
Files	**Description**	**Attributes**
chat_create_team_chat .csv	A line is added to this table when a user creates a new chat session for the team.	**userId:** user id who created the session. **teamid:** the id of the team. **teamChatDataSessionId:** unique id for the chat session. **timestamp**: time session was created.
chat_item_team_chat .csv	A line is added to this table when a user creates a new chat post for a specific team chat session.	**userId:** user id who created the post. **teamChatDataSessionId:** id for the chat session. **chatItemId**: unique id for chat post. **timestamp**: time post was created.
chat_join_team_chat .csv	A line is added to this table when a user joins to a team chat session	**userId:** user id joins the session. **teamChatDataSessionId:** id for the chat session. **timestamp**: time user joins session.
chat_leave_team_chat .csv	A line is added to this table when a user leaves from a team chat session.	**userId:** user id leaves the session. **teamChatDataSessionId:** id for the chat session. **timestamp**: time user leaves session.
chat_mention_team_chat .csv	A line is added to this table when a user is mentioned in a chat post.	**userId:** user id is mentioned. **chatItemId**: id for chat post. **timestamp**: time user was mentioned.
chat_respond_team_chat .csv	A line is added to this table when a chat post is in response to another chat post	**chatItemId**: id of chat item responses. **chatItemId:** id of chat item **timestamp**: time when chat post responded to the another.

2. For the loading process we are gonna use Cypher language in order to read the .csv files with the LOAD command row by row. Then, we will use the MERGE command in order to create the specific nodes and edges are represented in the file. Let explain the process with the following example.

```
1 LOAD CSV FROM "file:////chat-data/chat_item_team_chat.csv" AS row
2 MERGE (u:User {id: toInteger(row[0])})
3 MERGE (c:TeamChatSession {id: toInteger(row[1])})
4 MERGE (t:ChatItem{id: toInteger(row[2])})
5 MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(t)
6 MERGE (t)-[:PartOf{timeStamp: row[3]}]->(c)
```

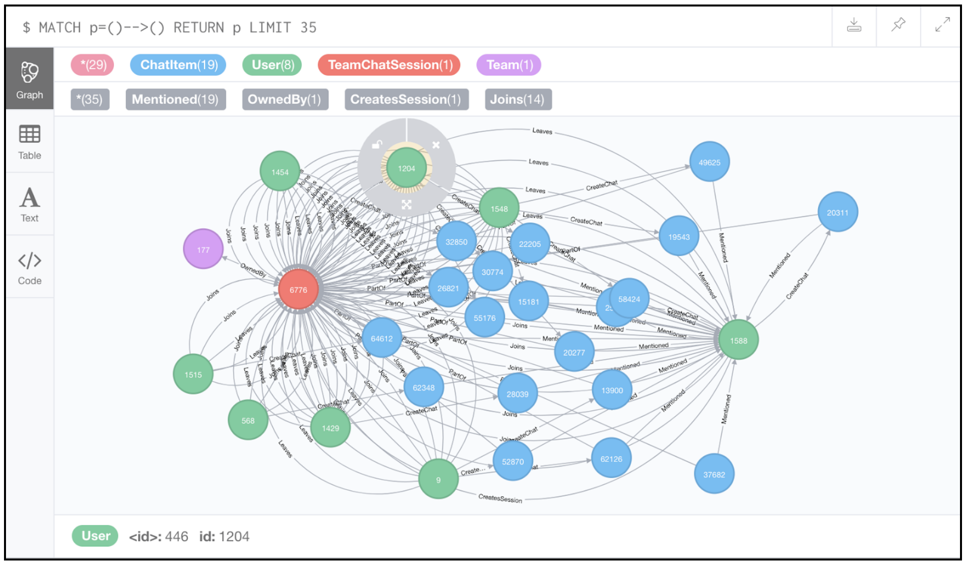
   The LOAD command reads the file row by row. For each row it creates a ***User*** node using as id attribute the column 0; a ***TeamChatSession*** node using as id attribute the column 1; and a ***ChatItem*** node using as id attribute the column 2. The MERGE command on line 5 crates a new edge (called "***CreateChat***") between ***User*** and ***TeamChatSession*** nodes and sets the timesStamp value with the value of column 3. The MERGE command on line 6 crates a new edge (called "***PartOf***") between ***ChatItem*** and ***TeamChatSession*** nodes and sets the timesStamp value with the value of column 4. Similar queries would be used to load the other files and merge commands will be modified to create nodes and edges as expected by the exercize.

3. Following screen shots show the resulting graph we obtained after loading all 6 files in our Neo4j environment. The first example is a rendered in the default Neo4j distribution, the second has had some nodes moved to expose the edges more clearly.



```
$ MATCH p=()-->() RETURN p LIMIT 35
```

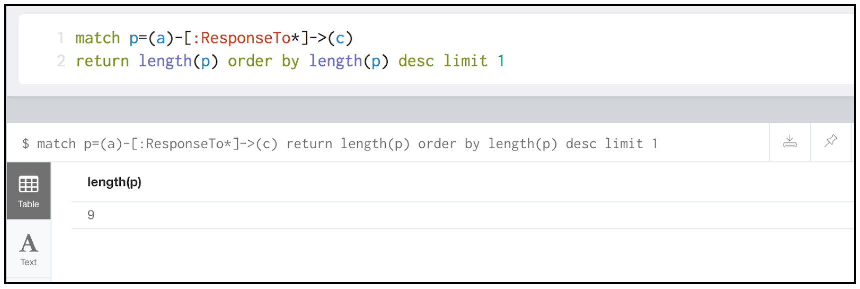
Displaying 29 nodes, 158 relationships.



```
$ MATCH p=()-->() RETURN p LIMIT 35
```

Finding the longest conversation chain and its participants

1. The longest conversation chain can be found analyzing the connectivity of ***ChatItem*** nodes through the ***ResponseTo*** edges. The following query will find the longest conversation chain in the graph, which is **9**.

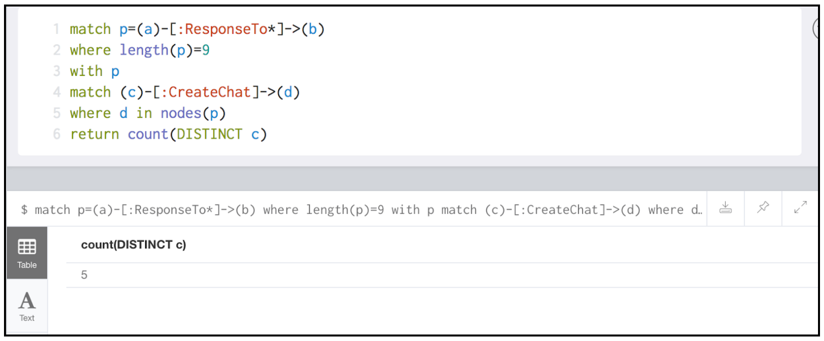


```
match p=(a)-[:ResponseTo*]->(c)
return length(p) order by length(p) desc limit 1
```

```
$ match p=(a)-[:ResponseTo*]->(c) return length(p) order by length(p) desc limit 1
```

length(p)
9

2. In order to get the number of unique users were part of the conversation chain. We can find all users are linked to ***ChatItem*** nodes in the longest chain. Via ***CreateChat*** edges, we can count the distinct users have created a post is part of longest conversation chain. The following query will give the value of **5** unique users.



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

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

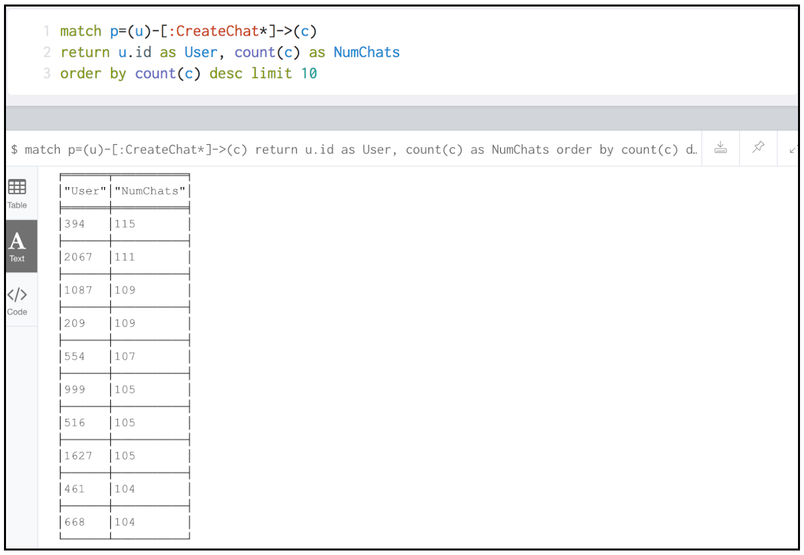
count(DISTINCT c)
5

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

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

We can find the top 10 chattiest ***User*** nodes using the ***CreateChat*** edge. Thus, we can count the number of ***ChatItem*** nodes each user has created. The following query will find the top 10 chattiest users.



```
match p=(u)-[:CreateChat*]->(c)
return u.id as User, count(c) as NumChats
order by count(c) desc limit 10
```

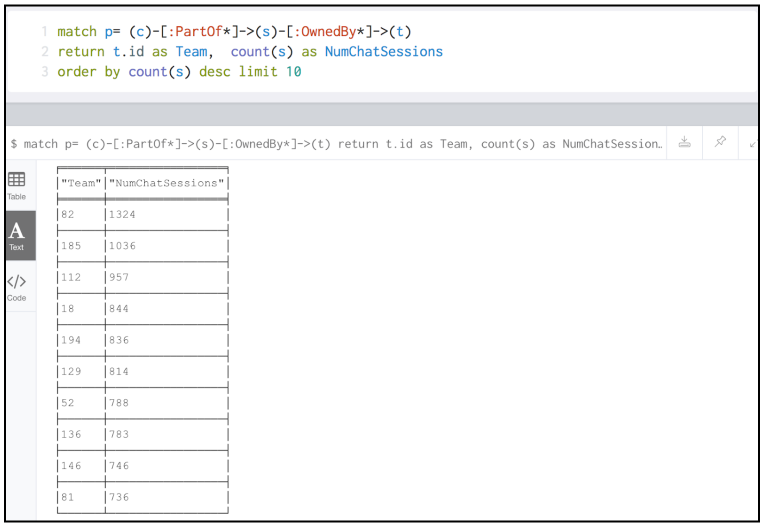
```
$ match p=(u)-[:CreateChat*]->(c) return u.id as User, count(c) as NumChats order by count(c) d…
```

```
"User"|"NumChats"
394   |115
2067  |111
1087  |109
209   |109
554   |107
999   |105
516   |105
1627  |105
461   |104
668   |104
```

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

Chattiest Teams

We can find the top 10 chattiest ***Team*** nodes using, as suggested in the exercise, all ***ChatItem*** nodes that has a ***PartOf*** edge connecting to a ***TeamChatSession*** node. Then, we can count the number of ***TeamChatSession*** nodes we find for each team. The following query will find top 10 chattiest teams.



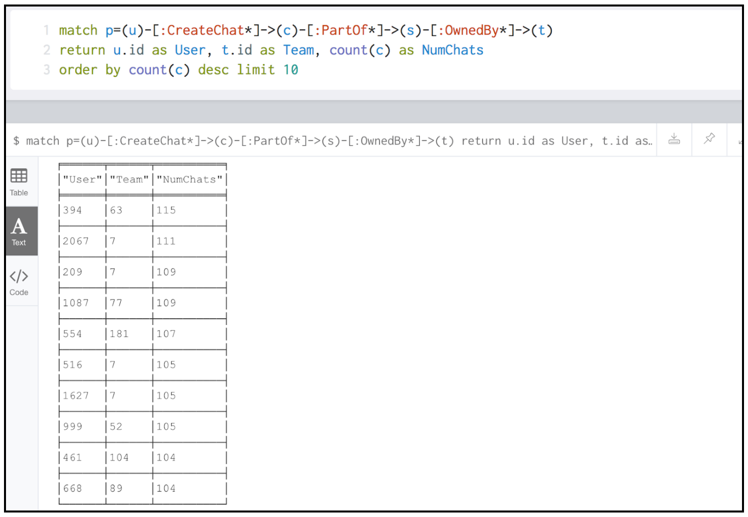
```
match p= (c)-[:PartOf*]->(s)-[:OwnedBy*]->(t)
return t.id as Team,  count(s) as NumChatSessions
order by count(s) desc limit 10
```

```
$ match p= (c)-[:PartOf*]->(s)-[:OwnedBy*]->(t) return t.id as Team, count(s) as NumChatSession…
```

```
╒══════╤═════════════════╕
│"Team"│"NumChatSessions"│
╞══════╪═════════════════╡
│82    │1324             │
├──────┼─────────────────┤
│185   │1036             │
├──────┼─────────────────┤
│112   │957              │
├──────┼─────────────────┤
│18    │844              │
├──────┼─────────────────┤
│194   │836              │
├──────┼─────────────────┤
│129   │814              │
├──────┼─────────────────┤
│52    │788              │
├──────┼─────────────────┤
│136   │783              │
├──────┼─────────────────┤
│146   │746              │
├──────┼─────────────────┤
│81    │736              │
└──────┴─────────────────┘
```

Teams	Number of Chats
82	1324
185	1036
112	957

To respond whether or not any of the chattiest user are part of any of the chattiest teams. We can list in a new query (below) the team where each top 10 chattiest users belongs. Thus, correlating this new table with the top 10 chattiest teams table (found previously); we can verify that just the user **999** belongs to one of chattiest teams, for instance team **52**. The following query will show the team each top 10 chattiest user belongs to.



```
match p=(u)-[:CreateChat*]->(c)-[:PartOf*]->(s)-[:OwnedBy*]->(t)
return u.id as User, t.id as Team, count(c) as NumChats
order by count(c) desc limit 10
```

```
$ match p=(u)-[:CreateChat*]->(c)-[:PartOf*]->(s)-[:OwnedBy*]->(t) return u.id as User, t.id as…
```

```
╒══════╤══════╤══════════╕
│"User"│"Team"│"NumChats"│
╞══════╪══════╪══════════╡
│394   │63    │115       │
├──────┼──────┼──────────┤
│2067  │7     │111       │
├──────┼──────┼──────────┤
│209   │7     │109       │
├──────┼──────┼──────────┤
│1087  │77    │109       │
├──────┼──────┼──────────┤
│554   │181   │107       │
├──────┼──────┼──────────┤
│516   │7     │105       │
├──────┼──────┼──────────┤
│1627  │7     │105       │
├──────┼──────┼──────────┤
│999   │52    │105       │
├──────┼──────┼──────────┤
│461   │104   │104       │
├──────┼──────┼──────────┤
│668   │89    │104       │
└──────┴──────┴──────────┘
```

How Active Are Groups of Users?

In this question, we will compute an estimate of how "dense" the neighborhood of a node is. In the context of chat that translates to how mutually interactive a certain group of users are. If we can identify these highly interactive neighborhoods, we can potentially target some members of the neighborhood for direct advertising. As suggested by the exercise we can follow the basic steps described below.

a) We will construct a neighborhood of users creating new edges (called "***InteractsWith***") between then. Following the queries that creates new edges for: users are mentioned for another users chat posts.

```
Match (u1:User)-[:CreateChat]->(c)-[:Mentioned]->(u2:User)
create (u1)-[:InteractsWith]->(u2)
```

And when a user creates a new chat post in response to another user's chat post. Here, for each user we are collecting all chat post created in response to chat items he created. Then, we are creating a new edge with the users owners of these chat post.

```
match (u1:User)-[:CreateChat]->(c:ChatItem)-[:ResponseTo]->(c1:ChatItem)
with u1, c, c1
match (u2:User)-[:CreateChat]->(c1)
create (u1)-[:InteractsWith]->(u2)
```

b) As mentioned by the exercise, after these two queries it is necessary to delete some unnecessary lood nodes may be created executing the previous queries.

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

c) In order to calculate the clustering coefficient we can follow the steps as suggested by the exercise. Notice that same query can be used for all user nodes; but in our case we are just interested in the top 10 chattiest users. With help of the following query we will explain in more detail each step we do to obtain final coefficients.

First, we need to collect all neighbor nodes "***InteractsWith***" a specific chattiest user node (line 1). We collect nodes have different ids (line 4) in a list called "***neighbors***" and also count the number of different nodes as "***neighborCount***". Second, for a list of "***neighbors***", we collect the number of "***InteractsWith***" edges between them (lines 4,5,6). If any pair of neighbors nodes have more than one edge, we count it just as 1 (lines 7,8,9). Thus, ***sum(hasedge)*** (line 11) will have at the end the number of edges between neighbor nodes. Finally (in line 11), we calculate the clustering coefficient as indicates in the exercize.

```
 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 hasedge
11 return u1.id, sum(hasedge)*1.0/(neighborCount*(neighborCount-1)) as cCoeff order by
   cCoeff desc
```

Finally, report the top 3 most active users in the table below.

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.9523
554	0.9047
1087	0.8