‘CATCH THE PINK FLAMINGO’ GRAPH ANALYTICS REPORT USING NEO4J

Prepared by

Ken Wood

December 14, 2020

Modeling Chat Data using a Graph Data Model

All chat-related data is contained in six .csv files that were downloaded in Week 1 of the Capstone Project. 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

Describe the steps taken to create the graph database.

i) Write the schema of the 6 CSV files

chat_create_team_chat.csv
0. UserId
1. TeamId
2. TeamChatSessionId
3. timestamp

chat_join_team_chat.csv
0. UserId
1. TeamChatSessionId
2. timestamp

chat_leave_team_chat.csv
0. UserId
1. TeamChatSessionId
2. timestamp

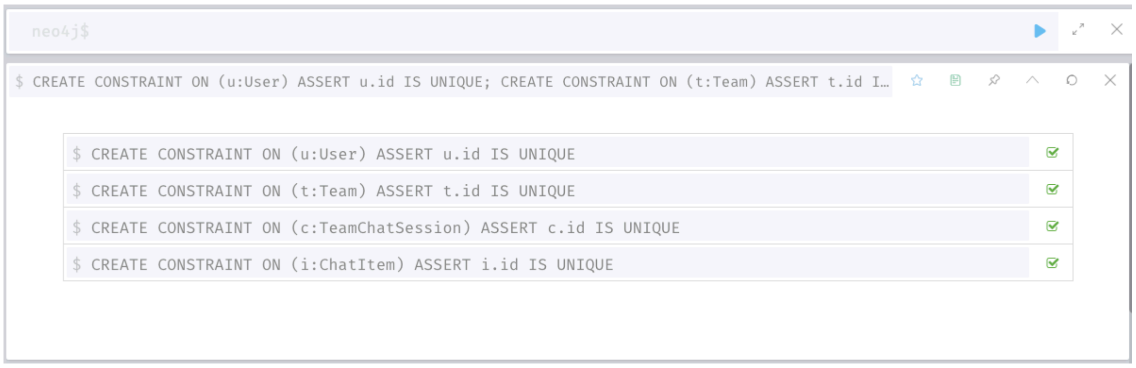
chat_item_team_chat.csv
0. UserId
1. TeamChatSessionId
2. ChatItemId
3. timestamp

chat_mention_team_chat.csv
0. ChatItemId
1. UserId
2. timestamp

chat_respond_team_chat.csv
0. ChatItemId
1. ChatItemId
2. timestamp

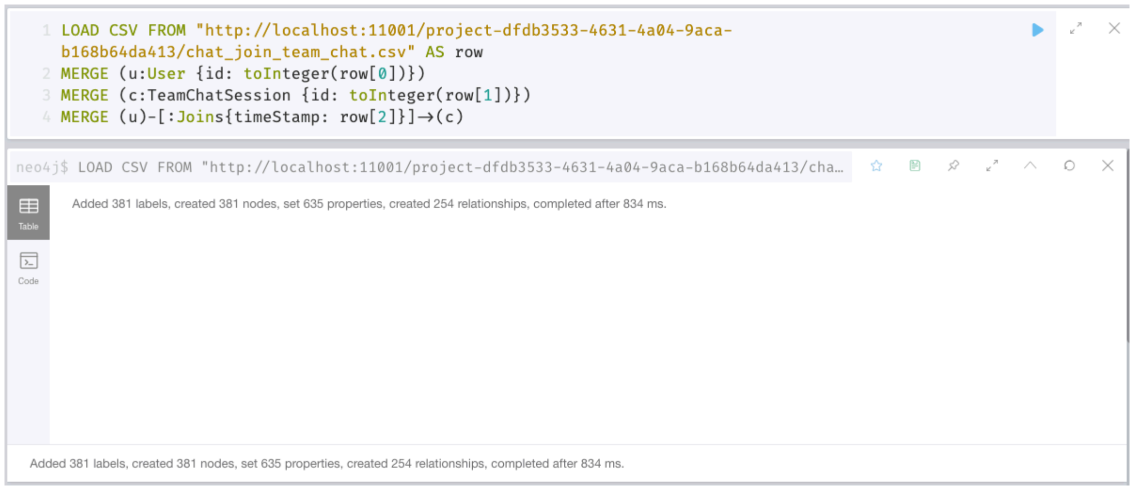
ii) Explain the loading process and include a sample LOAD command

In the Neo4J system we first execute the following instructions:



```
neo4j$

$ CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE; CREATE CONSTRAINT ON (t:Team) ASSERT t.id I…

$ 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
```

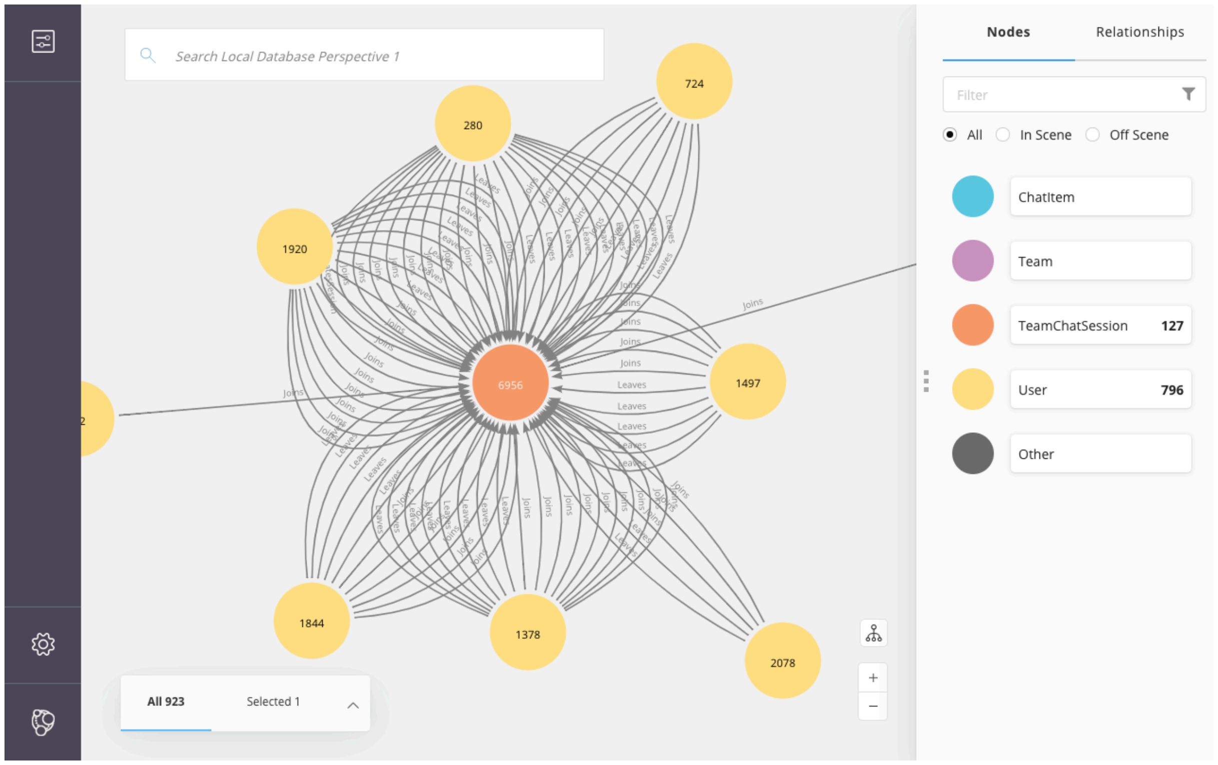
We then load the chat .csv files into Neo4J. Here is an example LOAD sequence:



```
LOAD CSV FROM "http://localhost:11001/project-dfdb3533-4631-4a04-9aca-
b168b64da413/chat_join_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])})
MERGE (c:TeamChatSession {id: toInteger(row[1])})
MERGE (u)-[:Joins{timeStamp: row[2]}]→(c)
```

```
neo4j$ LOAD CSV FROM "http://localhost:11001/project-dfdb3533-4631-4a04-9aca-b168b64da413/cha…

Added 381 labels, created 381 nodes, set 635 properties, created 254 relationships, completed after 834 ms.

Added 381 labels, created 381 nodes, set 635 properties, created 254 relationships, completed after 834 ms.
```

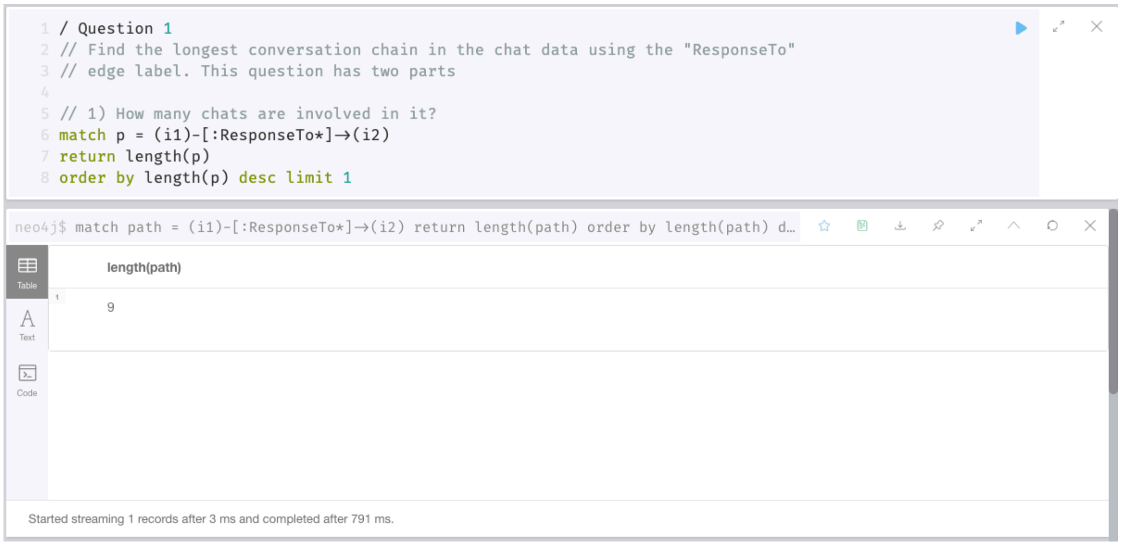
iii) Present a screenshot of some part of the graph you have generated. The graphs must include clearly visible examples of most node and edge types.



Finding the longest conversation chain and its participants

Report the results including the length of the conversation (path length) and how many unique users were part of the conversation chain.

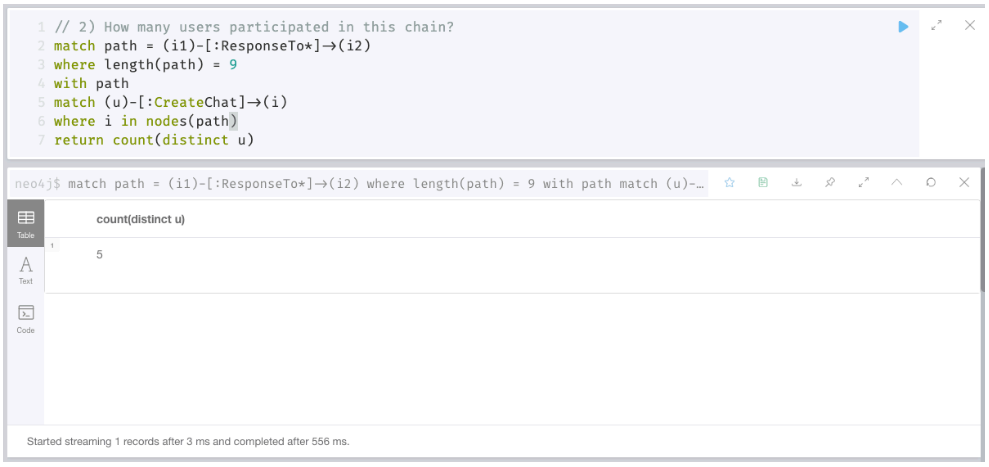
i) Longest conversation path length…



```
/ Question 1
// Find the longest conversation chain in the chat data using the "ResponseTo"
// edge label. This question has two parts

// 1) How many chats are involved in it?
match p = (i1)-[:ResponseTo*]→(i2)
return length(p)
order by length(p) desc limit 1
```

```
neo4j$ match path = (i1)-[:ResponseTo*]→(i2) return length(path) order by length(path) d…

length(path)
9

Started streaming 1 records after 3 ms and completed after 791 ms.
```

ii) Number of unique users in conversation…



```
// 2) How many users participated in this chain?
match path = (i1)-[:ResponseTo*]→(i2)
where length(path) = 9
with path
match (u)-[:CreateChat]→(i)
where i in nodes(path)
return count(distinct u)
```

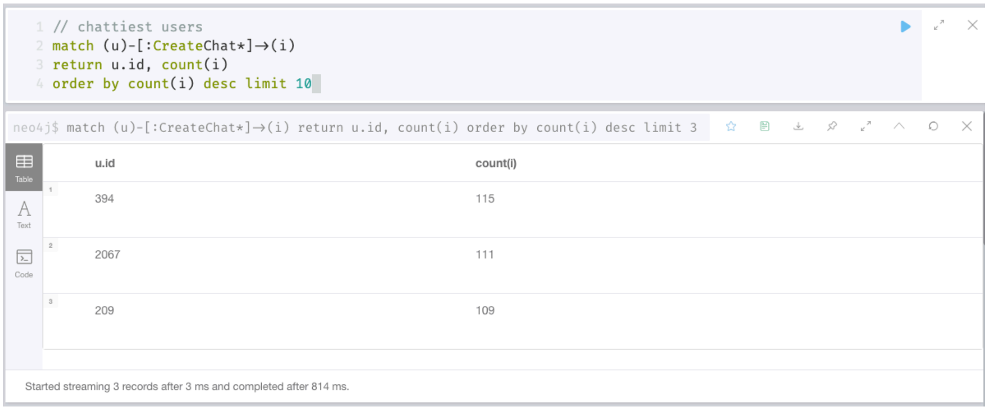
```
neo4j$ match path = (i1)-[:ResponseTo*]→(i2) where length(path) = 9 with path match (u)-…
```

count(distinct u)
5

Started streaming 1 records after 3 ms and completed after 556 ms.

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

Chattiest Users:



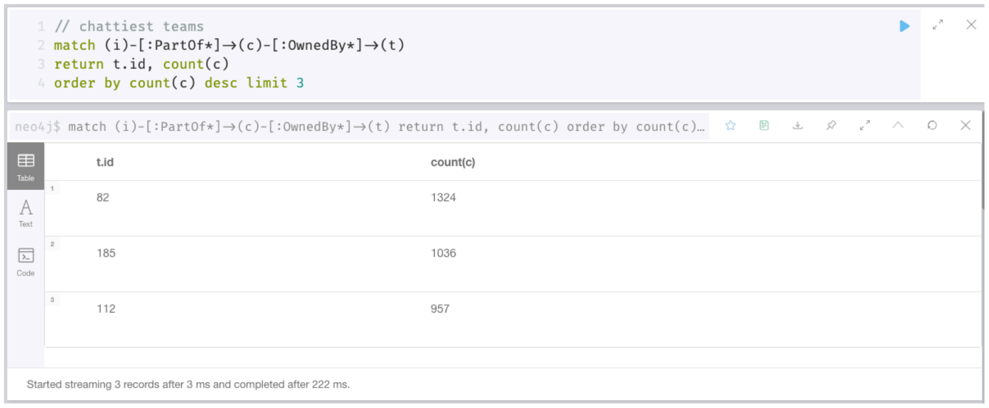
```
// chattiest users
match (u)-[:CreateChat*]→(i)
return u.id, count(i)
order by count(i) desc limit 10
```

```
neo4j$ match (u)-[:CreateChat*]→(i) return u.id, count(i) order by count(i) desc limit 3
```

u.id	count(i)
394	115
2067	111
209	109

Started streaming 3 records after 3 ms and completed after 814 ms.

Chattiest Teams:



```
// chattiest teams
match (i)-[:PartOf*]→(c)-[:OwnedBy*]→(t)
return t.id, count(c)
order by count(c) desc limit 3
```

```
neo4j$ match (i)-[:PartOf*]→(c)-[:OwnedBy*]→(t) return t.id, count(c) order by count(c)…
```

t.id	count(c)
82	1324
185	1036
112	957

Started streaming 3 records after 3 ms and completed after 222 ms.

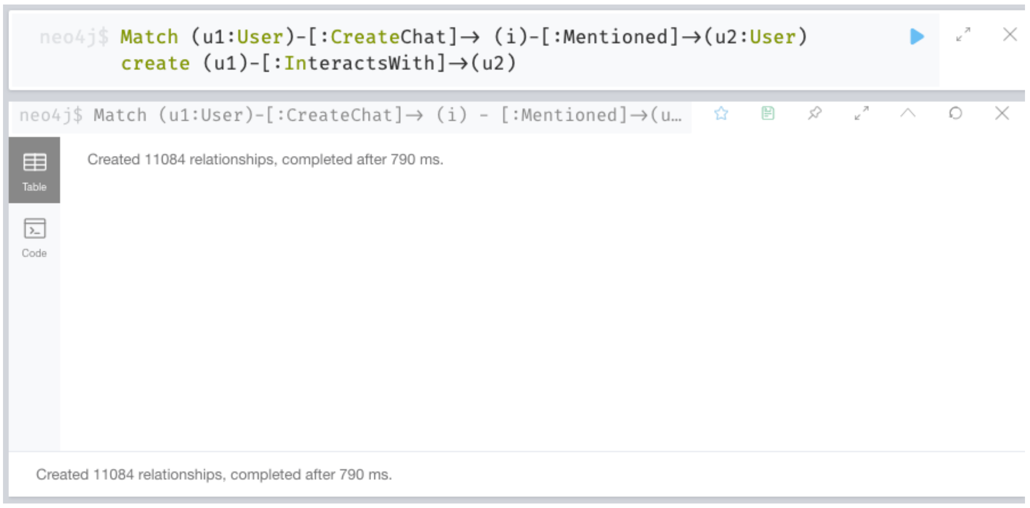
How Active Are Groups of Users?

First, 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. We will do this in a series of steps.

(a) We will construct the neighborhood of users. In this neighborhood, we will connect two users if:

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

The way to make this connection will be to create a new edge called, for example, “InteractsWith” between users to satisfy either of the two conditions. So, we will write a query to create these edges for each condition.

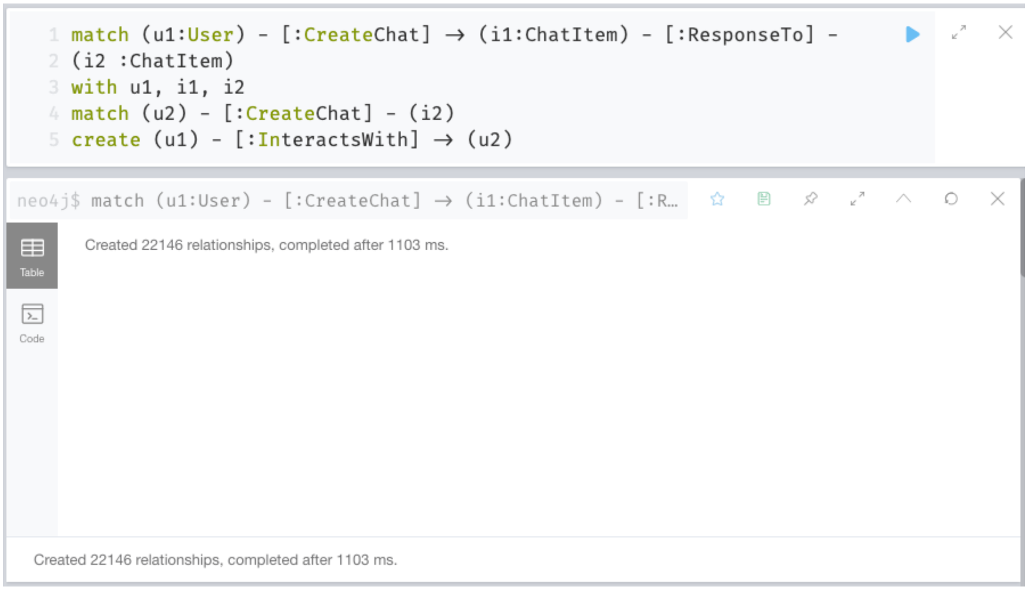


```
neo4j$ Match (u1:User)-[:CreateChat]→ (i)-[:Mentioned]→(u2:User)
       create (u1)-[:InteractsWith]→(u2)
```

```
neo4j$ Match (u1:User)-[:CreateChat]→ (i) - [:Mentioned]→(u…
Created 11084 relationships, completed after 790 ms.
```

```
Created 11084 relationships, completed after 790 ms.
```

We will use the same logic to create the query statement for the second condition.

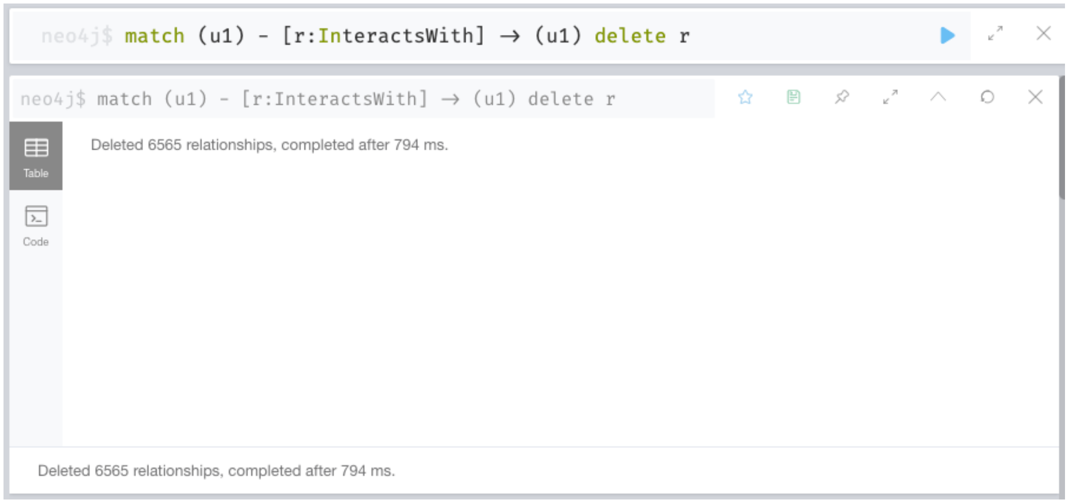


```
match (u1:User) - [:CreateChat] → (i1:ChatItem) - [:ResponseTo] -
(i2 :ChatItem)
with u1, i1, i2
match (u2) - [:CreateChat] - (i2)
create (u1) - [:InteractsWith] → (u2)
```

```
neo4j$ match (u1:User) - [:CreateChat] → (i1:ChatItem) - [:R…
Created 22146 relationships, completed after 1103 ms.
```

```
Created 22146 relationships, completed after 1103 ms.
```

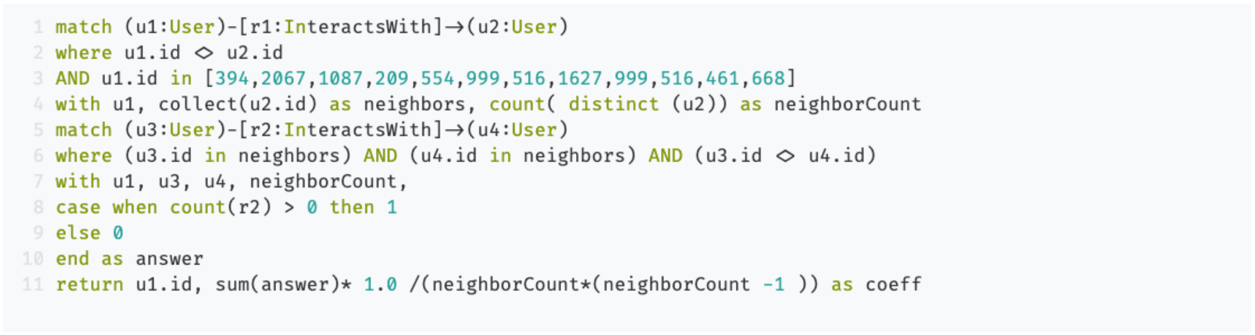
The above query creates an undesirable side effect if a user has responded to their own chat item, because it will create a self-loop between two users. So, after executing the query above, we will need to eliminate all self-loops involving the edge “InteractsWith”. This can be done through this query:



```
neo4j$ match (u1) - [r:InteractsWith] -> (u1) delete r
```

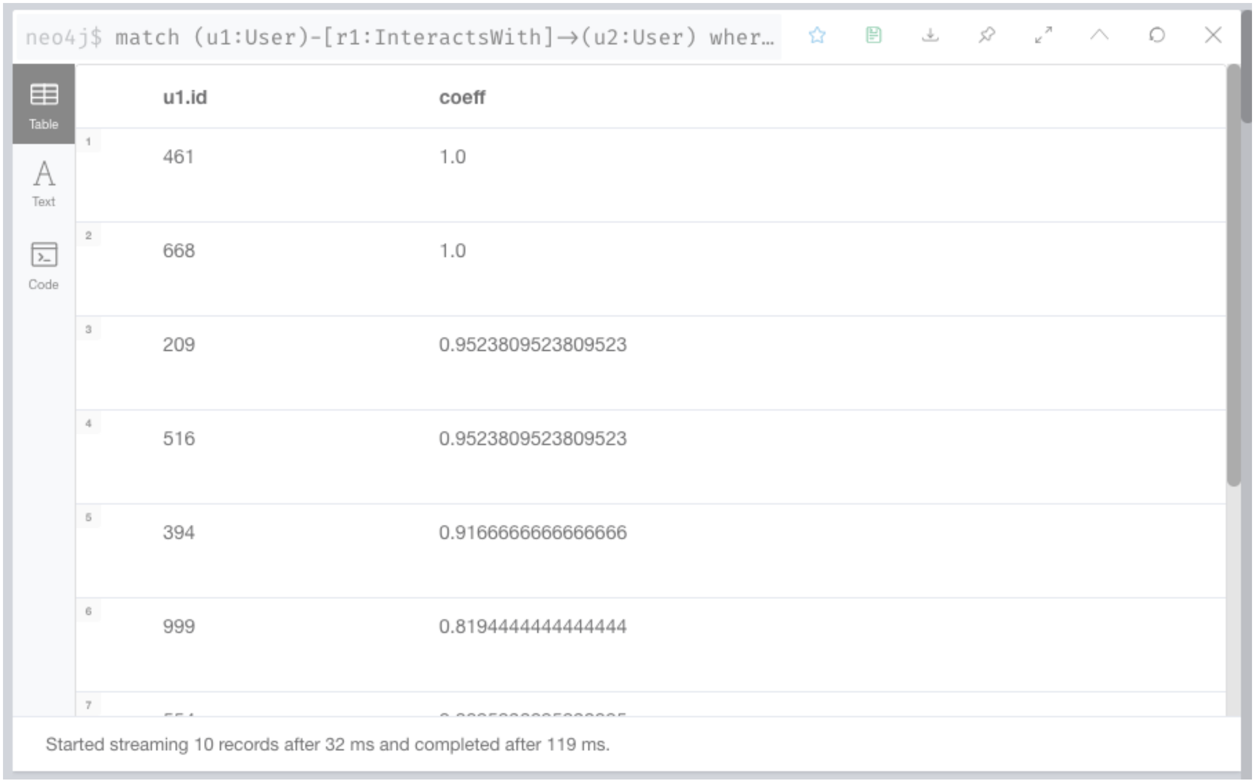
```
neo4j$ match (u1) - [r:InteractsWith] -> (u1) delete r

Deleted 6565 relationships, completed after 794 ms.

Deleted 6565 relationships, completed after 794 ms.
```

To create the clustering coefficient, we need to find all the users that are connected through the edge “InteractsWith” (line 1) and 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.



```
1  match (u1:User)-[r1:InteractsWith]->(u2:User)
2  where u1.id <> u2.id
3  AND u1.id in [394,2067,1087,209,554,999,516,1627,999,516,461,668]
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
```

Most Active Users (based on Cluster Coefficients)



```
neo4j$ match (u1:User)-[r1:InteractsWith]→(u2:User) wher…
```

	u1.id	coeff
1	461	1.0
2	668	1.0
3	209	0.9523809523809523
4	516	0.9523809523809523
5	394	0.9166666666666666
6	999	0.8194444444444444
7	[illegible]	[illegible]

Started streaming 10 records after 32 ms and completed after 119 ms.