Graph Analytics

Modeling Chat Data Using a Graph Data Model

The graph model is a network based on chat interactions between users. A chat session can be initiated by a user, other users on the same team are able to join and leave the session. Interactions between users begins when a user create a post. It’s possible for a user, mention another user. All relationship between entities are logged with a timestamp.

Creation of the Graph Database for Chats

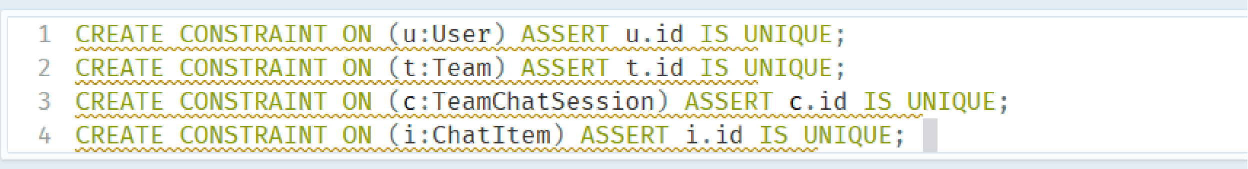
Describe the steps you took for creating the graph database

Write the schema of the 6 CSV files

File	Entity
Chat_create_team_chat.csv	userID teamID teamChatSessionID timestamp
Chat_join_team_chat.csv	userID teamChatSessionID timestamp
Chat_leave_team_chat.csv	userID teamChatSessionID timestamp
Chat_item_team_chat.csv	userID teamChatSessionID chatItemID timestamp
Chat_mention_team_chat.csv	ChatItemID UseID timestamp
Chat_respons_team_chat.csv	ChatItemID_1 ChatItemID_2 timestamp

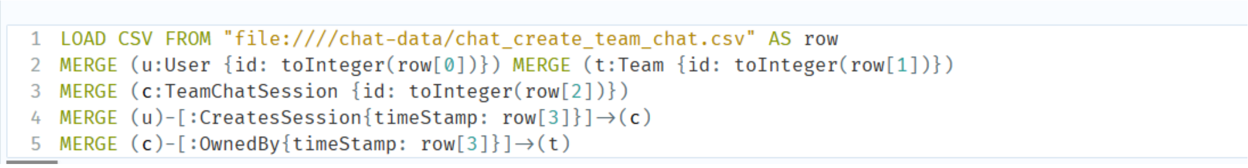
Explain the loading process and include a sample LOAD command

Creating constraints



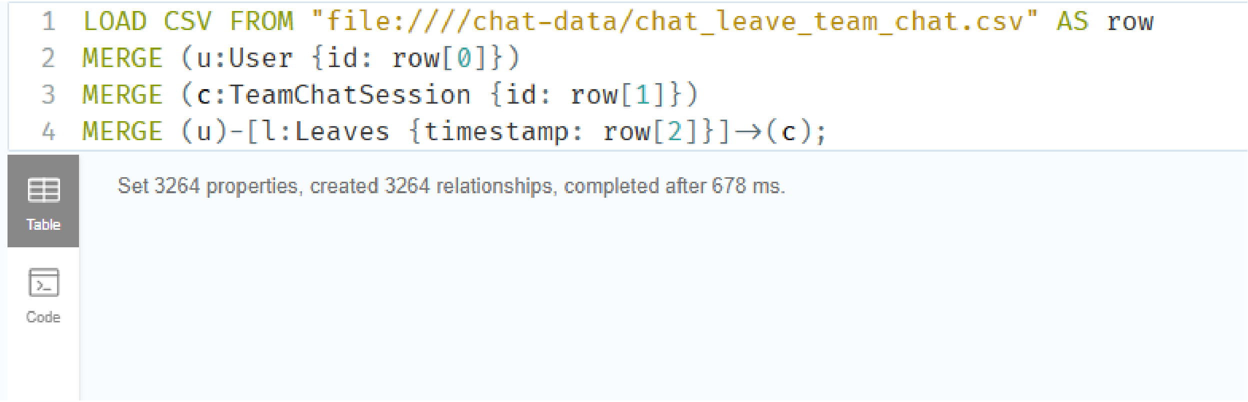
```
CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE;
CREATE CONSTRAINT ON (t:Team) ASSERT t.id IS UNIQUE;
CREATE CONSTRAINT ON (c:TeamChatSession) ASSERT c.id IS UNIQUE;
CREATE CONSTRAINT ON (i:ChatItem) ASSERT i.id IS UNIQUE;
```

Loading chat_create_team_chat



```
LOAD CSV FROM "file:///chat-data/chat_create_team_chat.csv" AS row
MERGE (u:User {id: toInteger(row[0])}) MERGE (t:Team {id: toInteger(row[1])})
MERGE (c:TeamChatSession {id: toInteger(row[2])})
MERGE (u)-[:CreatesSession{timeStamp: row[3]}]→(c)
MERGE (c)-[:OwnedBy{timeStamp: row[3]}]→(t)
```

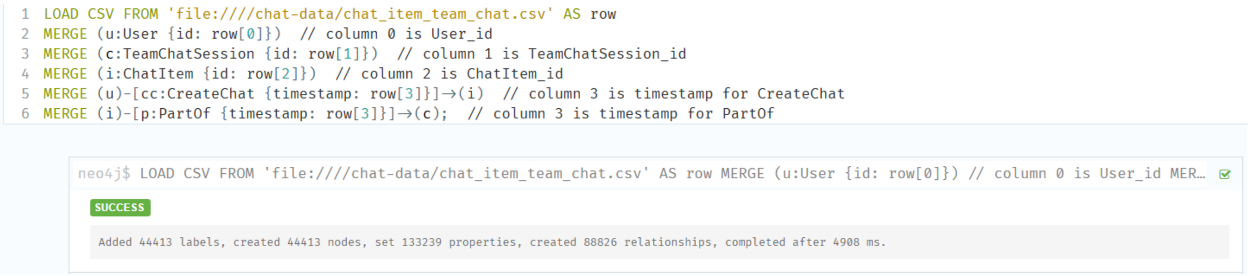
Loading chat_leave_team_chat



```
LOAD CSV FROM "file:///chat-data/chat_leave_team_chat.csv" AS row
MERGE (u:User {id: row[0]})
MERGE (c:TeamChatSession {id: row[1]})
MERGE (u)-[l:Leaves {timestamp: row[2]}]→(c);
```

```
Set 3264 properties, created 3264 relationships, completed after 678 ms.
```

Loading chat_item_team_chat

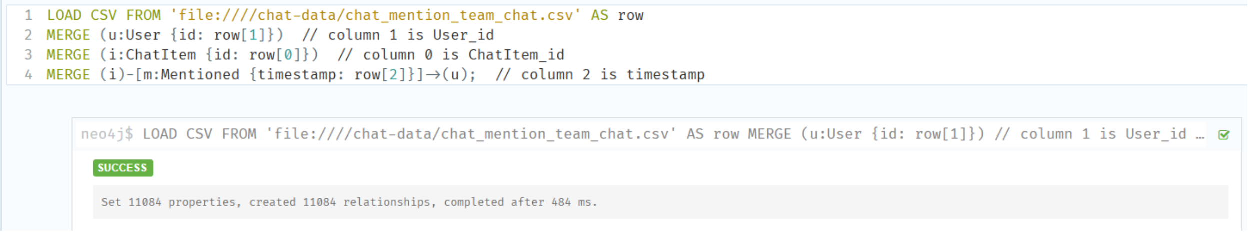


```
LOAD CSV FROM 'file:///chat-data/chat_item_team_chat.csv' AS row
MERGE (u:User {id: row[0]})  // column 0 is User_id
MERGE (c:TeamChatSession {id: row[1]})  // column 1 is TeamChatSession_id
MERGE (i:ChatItem {id: row[2]})  // column 2 is ChatItem_id
MERGE (u)-[cc:CreateChat {timestamp: row[3]}]→(i)  // column 3 is timestamp for CreateChat
MERGE (i)-[p:PartOf {timestamp: row[3]}]→(c);  // column 3 is timestamp for PartOf
```

```
neo4j$ LOAD CSV FROM 'file:///chat-data/chat_item_team_chat.csv' AS row MERGE (u:User {id: row[0]}) // column 0 is User_id MER…
SUCCESS
Added 44413 labels, created 44413 nodes, set 133239 properties, created 88826 relationships, completed after 4908 ms.
```

Coursera Big Data Specialization Capstone Project, Week 4

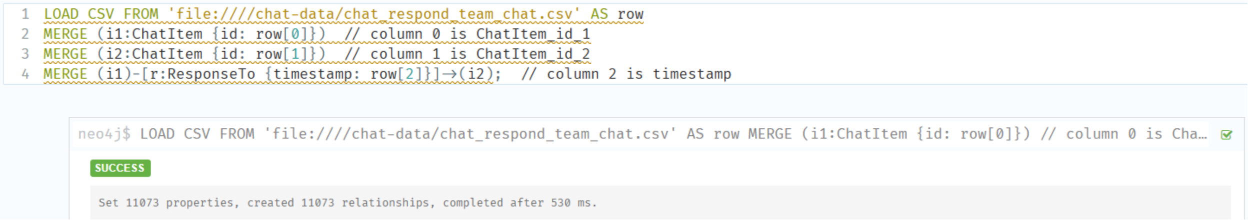
Loading chat_mention_team_chat



```
LOAD CSV FROM 'file:///chat-data/chat_mention_team_chat.csv' AS row
MERGE (u:User {id: row[1]})  // column 1 is User_id
MERGE (i:ChatItem {id: row[0]})  // column 0 is ChatItem_id
MERGE (i)-[m:Mentioned {timestamp: row[2]}]→(u);  // column 2 is timestamp
```

```
neo4j$ LOAD CSV FROM 'file:///chat-data/chat_mention_team_chat.csv' AS row MERGE (u:User {id: row[1]}) // column 1 is User_id …
SUCCESS
Set 11084 properties, created 11084 relationships, completed after 484 ms.
```

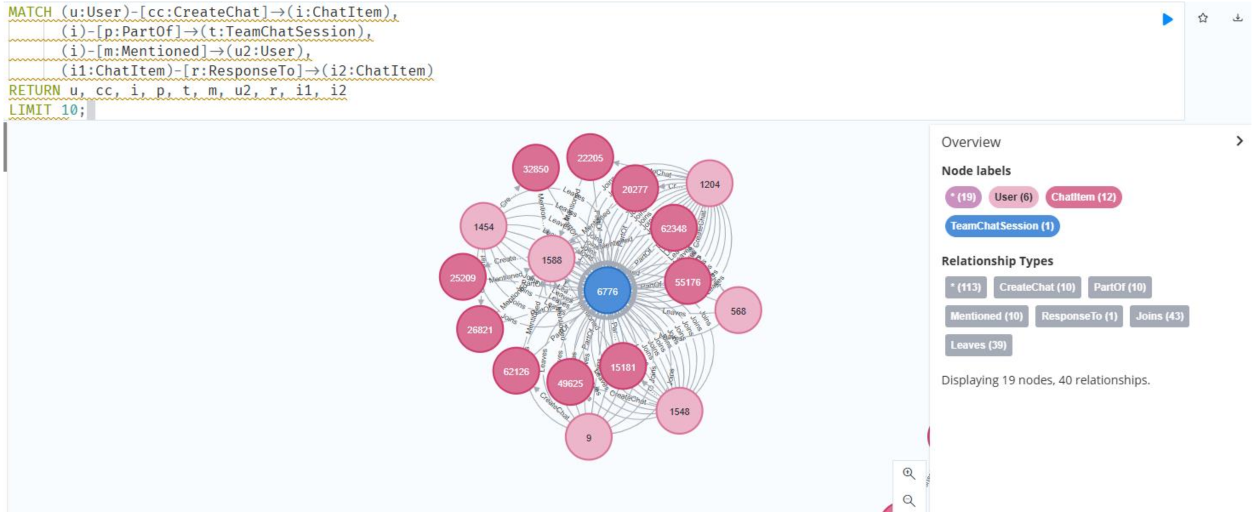
Loading Chat_resoibd_team_chat



```
LOAD CSV FROM 'file:///chat-data/chat_respond_team_chat.csv' AS row
MERGE (i1:ChatItem {id: row[0]})  // column 0 is ChatItem_id_1
MERGE (i2:ChatItem {id: row[1]})  // column 1 is ChatItem_id_2
MERGE (i1)-[r:ResponseTo {timestamp: row[2]}]→(i2);  // column 2 is timestamp
```

```
neo4j$ LOAD CSV FROM 'file:///chat-data/chat_respond_team_chat.csv' AS row MERGE (i1:ChatItem {id: row[0]}) // column 0 is Cha…
SUCCESS
Set 11073 properties, created 11073 relationships, completed after 530 ms.
```

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.



```
MATCH (u:User)-[cc:CreateChat]→(i:ChatItem),
      (i)-[p:PartOf]→(t:TeamChatSession),
      (i)-[m:Mentioned]→(u2:User),
      (i1:ChatItem)-[r:ResponseTo]→(i2:ChatItem)
RETURN u, cc, i, p, t, m, u2, r, i1, i2
LIMIT 10;
```

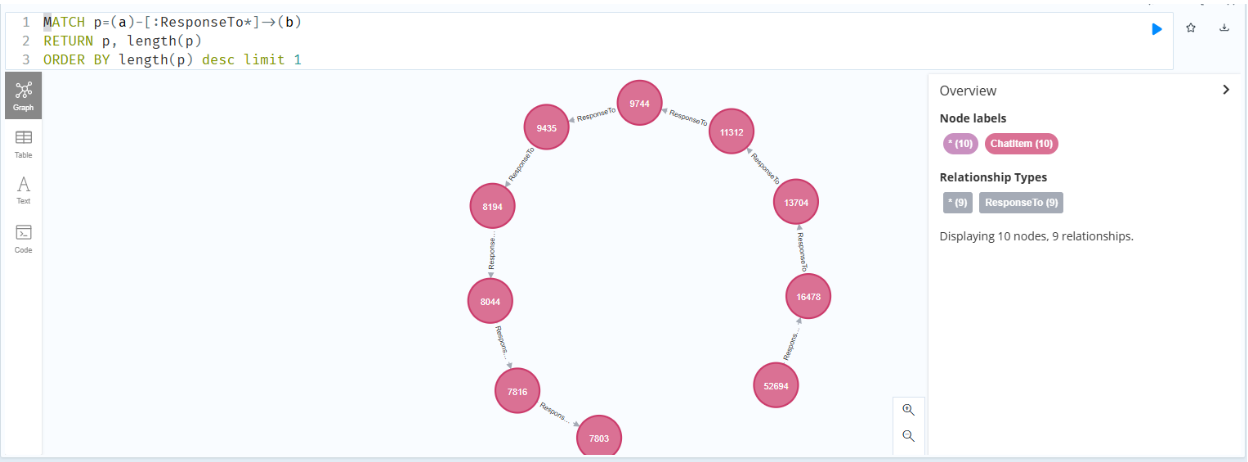
Displaying 19 nodes, 40 relationships.

Michael Ventura

Finding the longest conversation chain and its participants

Report the results including the length of the conversation and how many unique users were part of the conversation chain. Describe the steps. Write the query produces the correct answer

How many chats are involved in it?



```
MATCH p=(a)-[:ResponseTo*]→(b)
RETURN p, length(p)
ORDER BY length(p) desc limit 1
```

Overview

Node labels

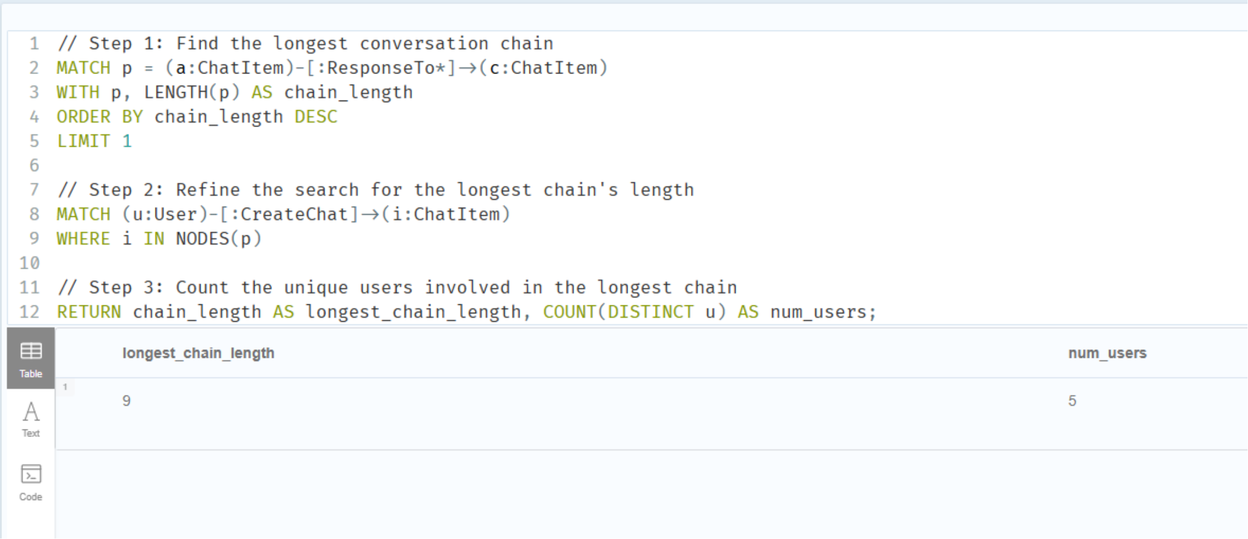
* (10) ChatItem (10)

Relationship Types

* (9) ResponseTo (9)

Displaying 10 nodes, 9 relationships.

How many users participated in this chain?



```
// Step 1: Find the longest conversation chain
MATCH p = (a:ChatItem)-[:ResponseTo*]→(c:ChatItem)
WITH p, LENGTH(p) AS chain_length
ORDER BY chain_length DESC
LIMIT 1

// Step 2: Refine the search for the longest chain's length
MATCH (u:User)-[:CreateChat]→(i:ChatItem)
WHERE i IN NODES(p)

// Step 3: Count the unique users involved in the longest chain
RETURN chain_length AS longest_chain_length, COUNT(DISTINCT u) AS num_users;
```

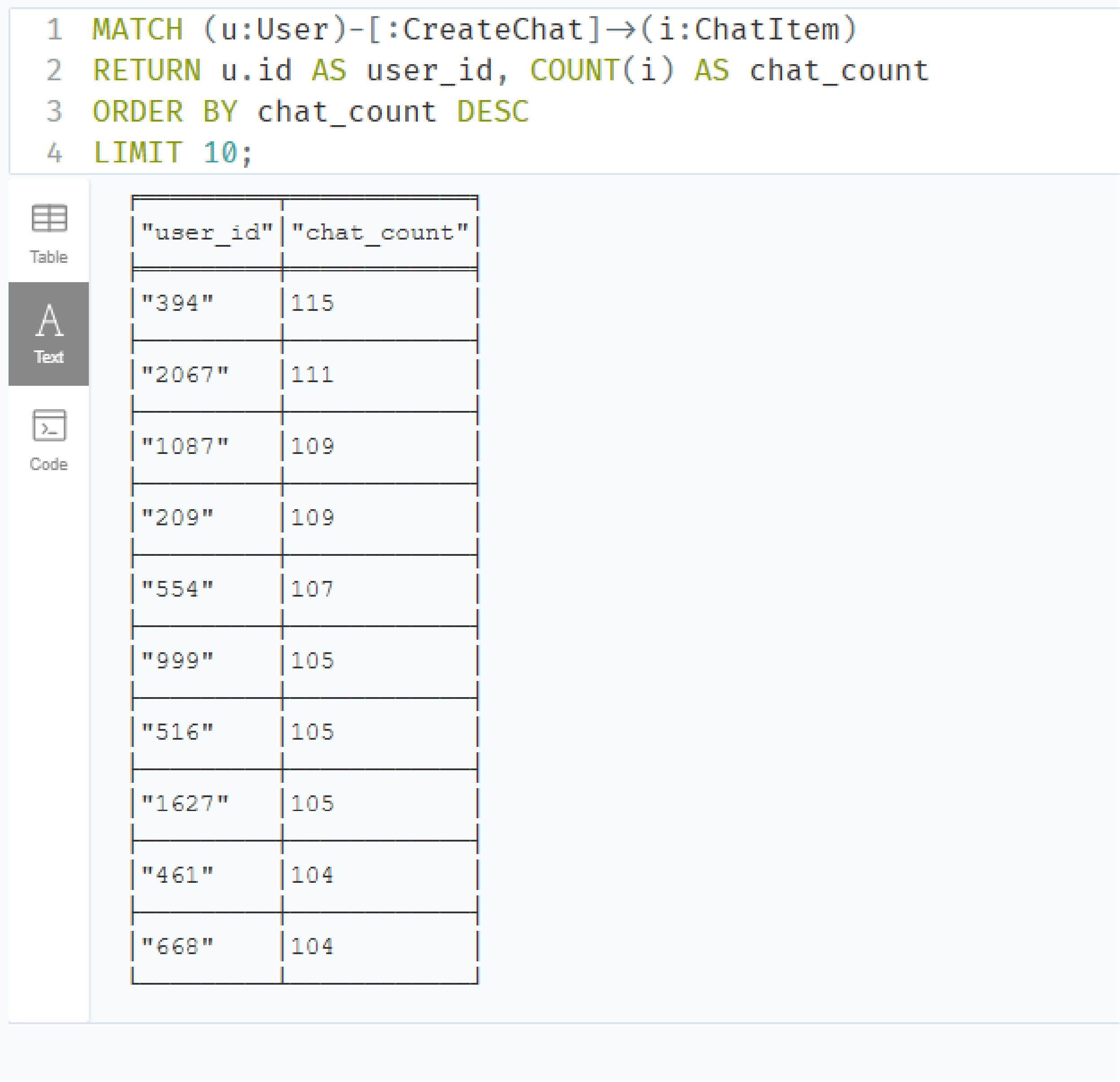
longest_chain_length	num_users
9	5

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

Determine the number of chats created by a user from the CreateChat edge

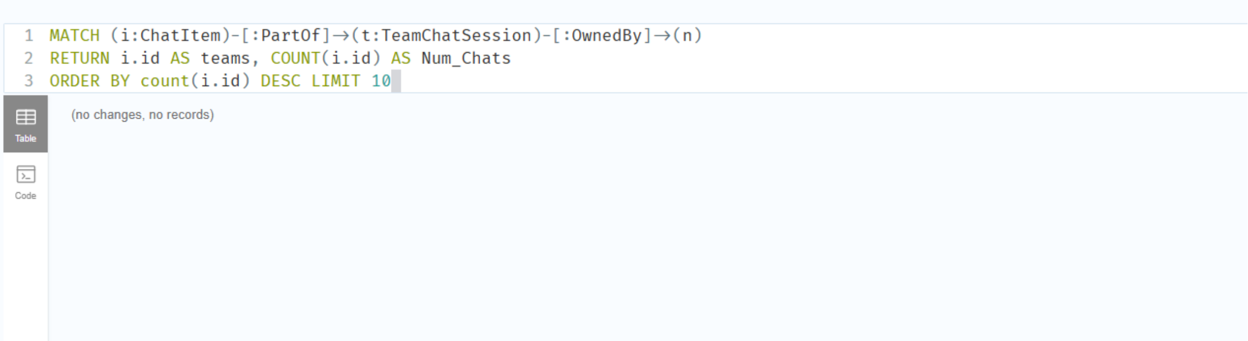


```
MATCH (u:User)-[:CreateChat]->(i:ChatItem)
RETURN u.id AS user_id, COUNT(i) AS chat_count
ORDER BY chat_count DESC
LIMIT 10;
```

```
╒═════════╤════════════╕
│"user_id"│"chat_count"│
╞═════════╪════════════╡
│"394"    │115         │
├─────────┼────────────┤
│"2067"   │111         │
├─────────┼────────────┤
│"1087"   │109         │
├─────────┼────────────┤
│"209"    │109         │
├─────────┼────────────┤
│"554"    │107         │
├─────────┼────────────┤
│"999"    │105         │
├─────────┼────────────┤
│"516"    │105         │
├─────────┼────────────┤
│"1627"   │105         │
├─────────┼────────────┤
│"461"    │104         │
├─────────┼────────────┤
│"668"    │104         │
└─────────┴────────────┘
```

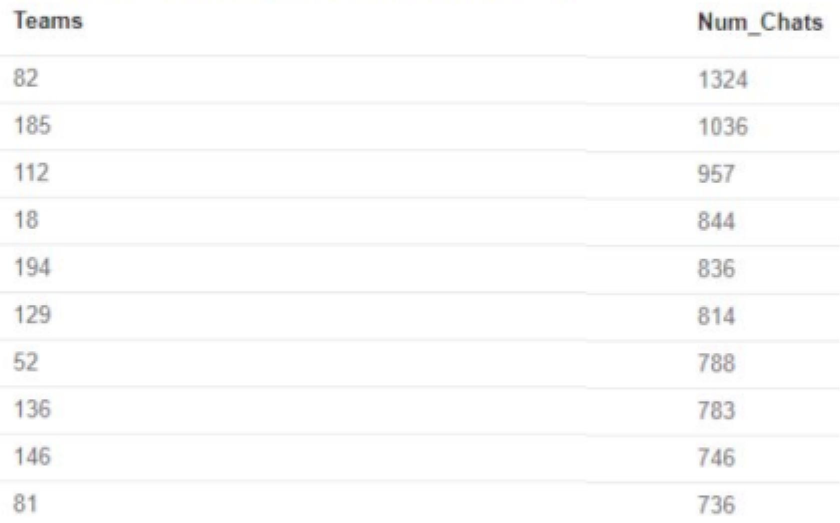
Chattiest Teams

Match all ChatItem with a PartOd edge and connect them with a TeamChatSession node that have an OwnedBy edge connection them with any other node.



```
MATCH (i:ChatItem)-[:PartOf]->(t:TeamChatSession)-[:OwnedBy]->(n)
RETURN i.id AS teams, COUNT(i.id) AS Num_Chats
ORDER BY count(i.id) DESC LIMIT 10
```

(no changes, no records)

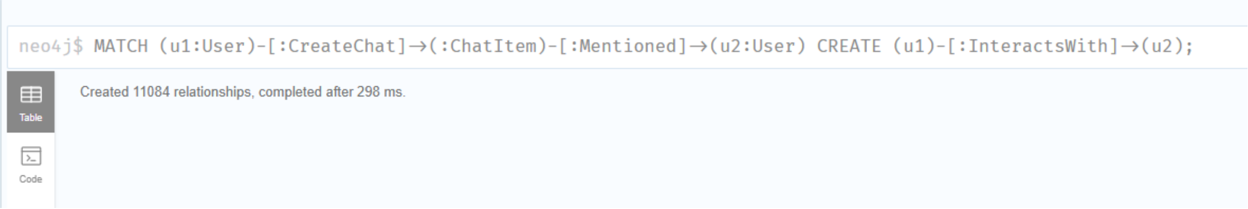


Teams	Num_Chats
82	1324
185	1036
112	957
18	844
194	836
129	814
52	788
136	783
146	746
81	736

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. We will do this in a series of steps.

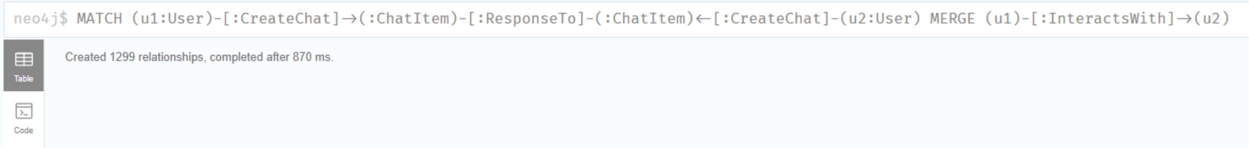
Connect mentioned users



```
neo4j$ MATCH (u1:User)-[:CreateChat]→(:ChatItem)-[:Mentioned]→(u2:User) CREATE (u1)-[:InteractsWith]→(u2);
```

Created 11084 relationships, completed after 298 ms.

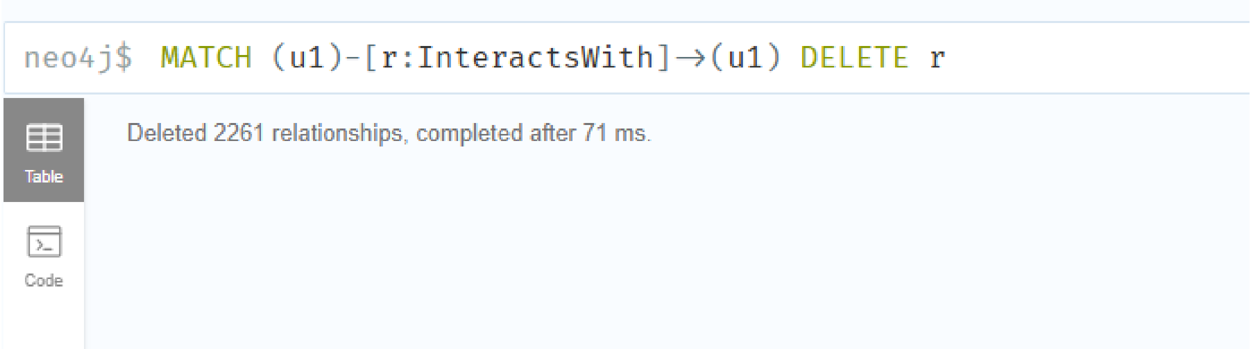
Connect user’s responses with the chat creator



```
neo4j$ MATCH (u1:User)-[:CreateChat]→(:ChatItem)-[:ResponseTo]-(:ChatItem)←[:CreateChat]-(u2:User) MERGE (u1)-[:InteractsWith]→(u2)
```

Created 1299 relationships, completed after 870 ms.

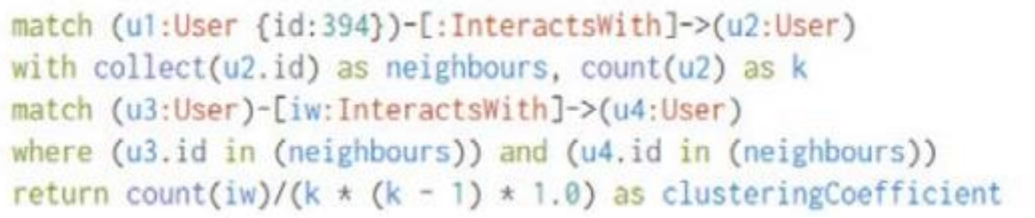
Eliminate all self-interaction



```
neo4j$ MATCH (u1)-[r:InteractsWith]→(u1) DELETE r
```

Deleted 2261 relationships, completed after 71 ms.

Calculate the cluster coefficient.



```
match (u1:User {id:394})-[:InteractsWith]->(u2:User)
with collect(u2.id) as neighbours, count(u2) as k
match (u3:User)-[iw:InteractsWith]->(u4:User)
where (u3.id in (neighbours)) and (u4.id in (neighbours))
return count(iw)/(k * (k - 1) * 1.0) as clusteringCoefficient
```

Most Active Users

User ID	Coefficient
394	0.9167
2067	0.7679
209	0.9524