Graph Analytics

Modeling Chat Data using a Graph Data Model

A Graph Data Model is used here to portray the interactions among the users in the game. A user can create a chat session and include other team members and have discussion threads between them. A user can also join a chat session, leave a chat session if he wishes to. A user can mention other users in the chat session. This data model helps us to analyze which team interacts more and helps us to improvise the game.

Creation of the Graph Database for Chats

Steps:

1. **Schema of 6 CSV files**

Name of the File	Attributes	Description
chat_create_team_chat.csv	UserID	It represents the ID of the user.
	TeamID	It represents the ID of the team.
	TeamChatSessionID	It represents the ID of the team's chat session.
	Timestamp	It represents the creation time of chat session.
chat_item_team_chat.csv	UserID	It represents the ID of the user.
	TeamChatSessionID	It represents the ID of the team's chat session.
	ChatItemID	It represents the ID of a chat item.
	Timestamp	It represents the creation time of chat session.
chat_join_team_chat.csv	UserID	It represents the ID of the user.
	TeamChatSessionID	It represents the ID of the team's chat session.
	Timestamp	It represents the creation time of chat session.
chat_leave_team_chat.csv	UserID	It represents the ID of the user.
	TeamChatSessionID	It represents the ID of the team's chat session.

	Timestamp	It represents the creation time of chat session.
chat_mention_team_chat.csv	ChatItemID	It represents the ID of a chat item.
	UserID	It represents the ID of the user.
	Timestamp	It represents the creation time of chat session.
chat_respond_team_chat.csv	ChatID1	It represents the ID of chat session 1
	ChatID2	It represents the ID of chat session 2
	Timestamp	It represents the creation time of chat session.

2. Loading Process

LOAD Command

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

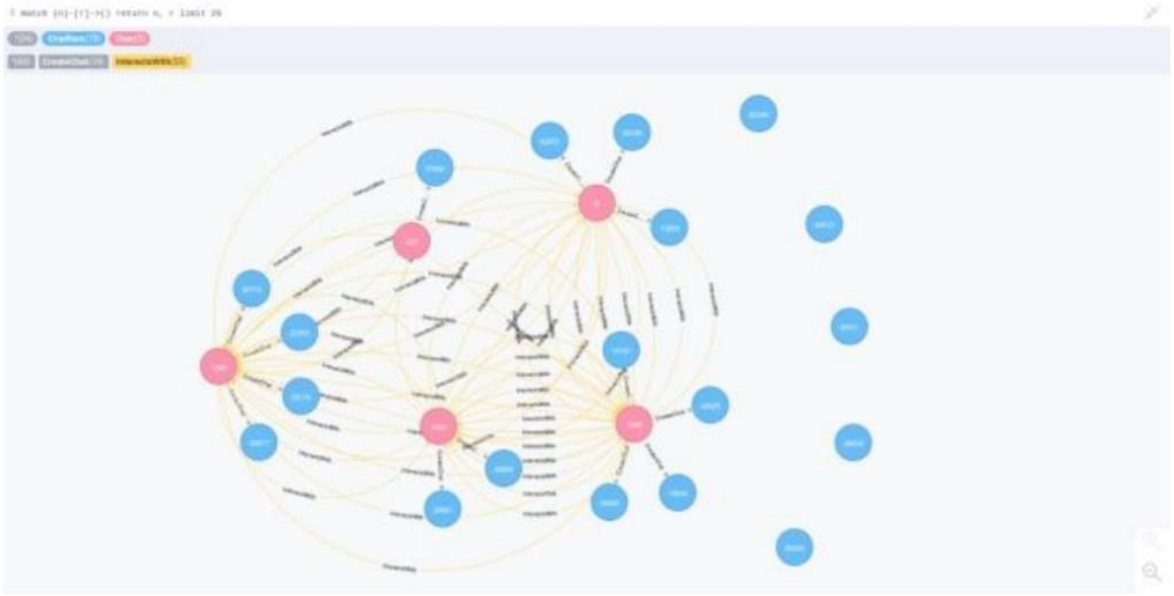
The LOAD CSV command loads the csv data present in that location as row at a time and creates user nodes.

The MERGE (u)-[:CreateChat{timeStamp: row[3]}]->(i) command is used to create an edge labeled “CreateChat” between the User ‘u’ and the ChatItem ‘I’. The edge contains a property called timeStamp. This property is filled by the content of column 3 of the same row.

The MERGE (i)-[:PartOf{timeStamp: row[3]}]->(c) Command is used to create an edge labeled “PartOf” between the ChatItem ‘I’ and the TeamChatSession ‘c’. The edge contains a property called timeStamp. This property is filled by the content of column 3 of the same row.

3. Screenshot

The graphs must include clearly visible examples of most node and edge types. Below are two acceptable examples. The first example is a rendered in the default Neo4j distribution, the second has had some nodes moved to expose the edges more clearly. Both include examples of most node and edge types.



Finding the longest conversation chain and its participants

1) **The longest conversation chain** Is traced via `ChatItem` nodes which are connected by `ResponseTo` edges, to find the longest one order it's length. The following query when executed gives the longest conversation chain that has path length of **9**.

   a) Query

```
match p = (i1)-[:ResponseTo*]->(i2)
return length(p)
order by length(p) desc limit 1
```

   b) Result

2) **Number of unique users** can be identified using the below query, we already know that the path length of the longest conversation chain is 9, hence to find all the distinct users which are part of this path, the query is executed, **5** unique users are extracted.

   a) Query

```
match p = (i1)-[:ResponseTo*]->(i2)
where length(p) = 9
with p
match (u)-[:CreateChat]->(i)
where i in nodes(p)
return count(distinct u)
```

   b) Result

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

1) Chattiest Users

The CreateChat edge from User node is matched with the ChatItem node and then return the ChatItem amount for every user , grouping is done in descending order.

a) Query

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

b) Result

Users	Number of Chats
394	115
2067	111
1087	109

2) Chattiest Teams

The part of edge from ChatItem node is matched with the TeamChatSession node , next we match the OwnedBy edge from TeamChatSession node to Team node, TeamChatSession amount per team is returned in adescending order.

a) Query

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

b) Result

Teams	Number of Chats
82	1324
185	1036
112	957

3) Check Whether Chattiest users belong to the Chattiest teams

To perform checking we combine the above two queries:

a) Query

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

b) Result

From the above result it's evident that only one user '999' belonging to team '52' is part of the chattiest teams, but other users are not part of the chattiest teams.

Therefore, Most of the chattiest users do not belong to the chattiest teams.

How Active Are Groups of Users?

To know how active are groups of users it is necessary to determine the highly interactive neighborhoods. Activity of a group can be determined based on their chatting. The following query is executed to determine this result.

```
match (u1:User)-[r1:InteractsWith]->(u2:User)
where u1.id <> u2.id
with u1, collect(u2.id) as neighbors, count(distinct(u2)) as neighborAmount
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, neighborAmount,
case
when (u3)-->(u4) then 1
else 0
end as value
return u1, sum(value)*1.0/(neighborAmount*(neighborAmount-1)) as coeff
order by coeff desc limit 10
```

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
209	0.9523809523809523
554	0.9047619047619048
1087	0.8