

Neo4j Graph Analysis of Catch the Pink Flamingo Chat Data

Modeling Chat Data using a Graph Data Model

The graph model illustrates the dynamics of chat interaction between ‘Catch the Pink Flamingo’ users. A user of a team can create a chat-session, where other users on the same team are free to join and subsequently leave. Interaction is initiated when a user creates a post (also known as chat-item), to which other users can reply to with their own chat-item, or simply mention it as a quick reply. Such posts are linearly linked to one another as a response, and all of these posts are part of the same team chat-session. All relationships between entities of this graph model are universally marked with a timestamp, while the entities are identified by a unique ID value.

Creation of the Graph Database for Chats

i) Schema of the 6 CSV files as follows:

CSV file	Field	Description
chat_create_team_chat	user.id	User ID value
	team.id	Team ID value
	teamchatsession.id	ID value assigned to the chat-session
	timestamp	Date/time of creation of chat-session and ownership assignment of chat-session to team.
chat_join_team_chat	user.id	User ID value
	teamchatsession.id	Team chat-session ID value
	timestamp	Date/time of user joining team chat-session
chat_leave_team_chat	user.id	User ID value
	teamchatsession.id	Team chat-session ID value
	timestamp	Date/time of user leaving team chat-session

chat_item_team_chat	user.id	User ID value
	teamchatsession.id	Team chat-session ID value
	chatitem.id	Post ID value
	timestamp	Date/time of post creation by user and linking of post to team chat-session.
chat_mention_team_chat	chatitem.id	Post ID value
	user.id	User ID value
	timestamp	Date/time of user mentioning a particular post as a quick reply.
chat_respond_team_chat	chatitem.id	Post -1 ID value
	chatitem.id	Post-2 ID value
	timestamp	Date/time of post-1 created as a response to post-2

- ii) When loading a CSV file, each row is parsed for extracting ID values to create nodes, and timestamp values to create edges. Consider the following load command:

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

1st line loads the csv file from the specified location and parses each line as a 'row' variable. 2nd, 3rd and 4th lines create User, Team and TeamChatSession nodes, with respective ID values extracted from the 'row' tuple using index values in sequential notation. 4th and 5th lines create the CreatesSession and OwnedBy edges with respective timestamp values extracted in similar manner, to link up the nodes. The MERGE command used to create the node/edge if it does not exist in the graph, or merge the attributes into already existing nodes/edges.

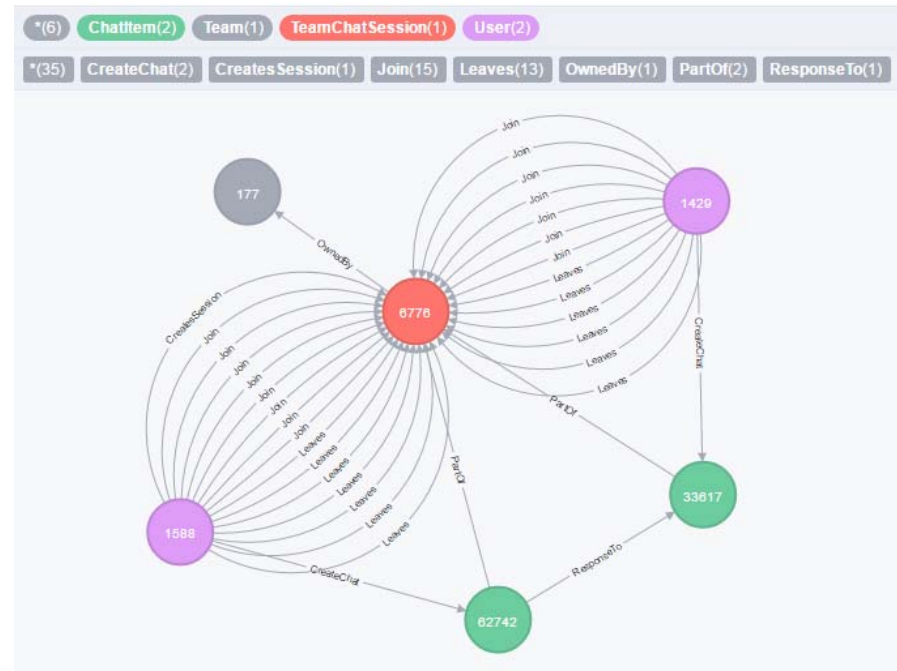
iii) Sample screenshot of a part of the graph:

Query:

match

```
(u2:User)-[cc2:CreateChat]->(ci2:ChatItem)<-[rt:ResponseTo]-(ci1:ChatItem)<-[cc1:CreateChat]-(u1:User)-[cs:CreatesSession]->(tcs:TeamChatSession)-[ob:OwnedBy]->(t:Team)
return u2,cc2,ci2,rt,ci1,cc1,u1,cs,tcs,ob,t limit 1
```

Graph:



Finding the longest conversation chain and its participants

Longest conversation chain is traced via ChatItem nodes that are linked by ResponseTo edges. A logical chain would be to trace a set of nodes with edges going in the same direction, for the longest possible path.

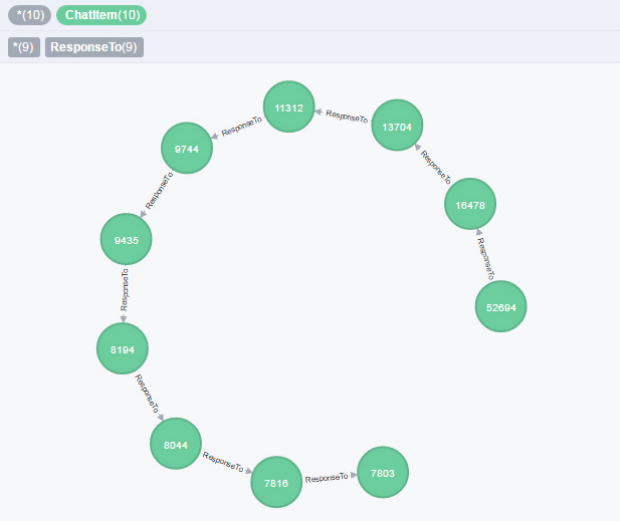
Longest conversation chain has path length of 9, i.e. **10 Chatitem nodes**.

Query:

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

Graph:

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



Now that the longest path has been determined to be 9, a subsequent query is used to find all the distinct users related to the 10 ChatItem nodes that are part this path. **5 unique users** were extracted.

Query:

```
match p=(a)-[:ResponseTo*]->(b)
where length(p) = 9
with p
match (u:User)-[]-(d)
where d in nodes(p)
return distinct(u)
```

Graph:

```
$ match p=(a)-[:ResponseTo*]->(b) where length(p) = 9 with p match (u:User)-[]-(d) where d in nodes(p) return distinct(u)
```



```
$ match p=(a)-[:ResponseTo*]->(b) where length(p) = 9 with p match (u:User)-[]-(d) where d in nodes(p) return distinct(u)
```

Graph

Rows

Text

Code

u
{id: 1978}
{id: 1514}
{id: 1192}
{id: 1153}
{id: 853}

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

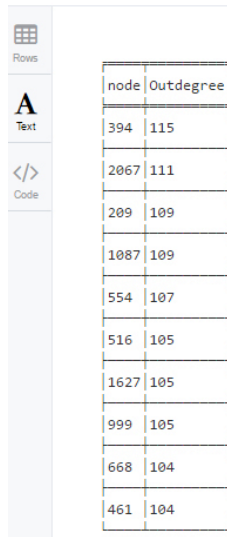
We first match for directional CreateChat relationship from User node to ChatItem node, then count the relationship links between the 2 mentioned nodes and subsequently return the sort list of user IDs, together with number of CreateChat edges (a.k.a out-degrees) that stem from them.

Query:

```
MATCH (u:User)-[r:CreateChat]->(ci:ChatItem)
RETURN u.id as node, count(r) as Outdegree
ORDER BY Outdegree desc LIMIT 10
```

Graph:

```
$ MATCH (u:User)-[r:CreateChat]->(ci:ChatItem) RETURN u.id as node, count(r) as Outdegree ORDER BY Outdegree desc LIMIT 10
```



node	Outdegree
394	115
2067	111
209	109
1087	109
554	107
516	105
1627	105
999	105
668	104
461	104

Chattiest Users

Users	Number of Chats
394	115
2047	111
209*	109
1087*	109

*A tie for 3rd placing between user 209 and 1087, both with 109 chatItems

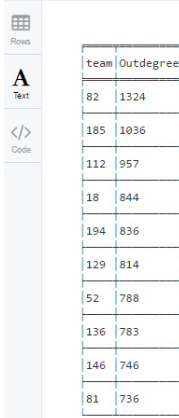
To find the chattiest teams, we match for all PartOf relationships linking ChatItem to TeamChatSession, where the latter is subsequently linked to a Team via OwnedBy relationship. Again we count the PartOf relationships for every TeamChatSession, isolate the top 10 with the highest numbers, then discover the Teams through the OwnedBy relationship.

Query:

```
MATCH (ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:OwnedBy]->(t:Team)
RETURN t.id as team, count(po) as Outdegree
ORDER BY Outdegree DESC LIMIT 10
```

Graph:

```
$ MATCH (ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:OwnedBy]->(t:Team) RETURN t.id as team, count(po) as Outdegree ORDER BY Outdegree DESC LIMIT 10
```



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

Chattiest Teams

Teams	Number of Chats
82	1324
185	1036
112	957

We now modify our query for top 10 chattiest users to trace the users back to their respective team. This facilitates the investigation of whether the top 10 chattiest users reside within the top 10 chattiest teams.

Query:

```
MATCH
(u:User)-[r>CreateChat]->(ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:OwnedBy]->(t:Team)
RETURN u.id as node, count(r) as Outdegree, t.id as team
ORDER BY Outdegree desc LIMIT 10
```

Graph:

```
$ MATCH (u:User)-[r>CreateChat]->(ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:Own
```

Rows

Text

Code

node	Outdegree	team
394	115	63
2067	111	7
209	109	7
1087	109	77
554	107	181
999	105	52
516	105	7
1627	105	7
461	104	104
668	104	89

As shown above, with the exception of user ID:999 (6th chattiest user) belonging to Team ID:52 (7th chattiest team), the other 9 users hail from teams that are not part of the top 10 chattiest teams list. Hence, it can be concluded the chattiest users do not usually hail from the chattiest teams.

How Active Are Groups of Users?

Describe your steps for performing this analysis. Be as clear, concise, and as brief as possible. Finally, report the top 3 most active users in the table below.

Activity is defined as user interacting with each other, i.e. 2 user nodes connected to each other via an edge for this definition to hold true. Given the present graph, we can connect 2 users in this manner based on either of the 2 following conditions:

- 1) User A is interacting with User B if A creates a ChatItem that B mentions, and both A and B are not referring to the same person. The following query creates the edge fulfilling this condition:

```
MATCH (u1:User)-[cc>CreateChat]->(ci:ChatItem)-[m:Mentioned]->(u2:User)
WHERE u1.id <> u2.id
MERGE (u1)-[:InteractsWith]->(u2)
```

- 2) User A is interacting with User B if A creates a ChatItem that is in response to a ChatItem that B has created on, and both A and B are not referring to the same person. The following query creates the edge fulfilling this condition:


```

MATCH
(u1:User)-[cc1:CreateChat]->(ci1:ChatItem)-[rt:ResponseTo]->(ci2:ChatItem)-[cc2:CreateChat]-(u2:User)
WHERE u1.id <> u2.id
MERGE (u1)-[:InteractsWith]->(u2)

```

Interactivity between 2 users is true if at any one of the above condition has been fulfilled at least once, hence the MERGE command is to ensure that at most 1 *interactsWith* edge can be formulated between 2 separate User nodes.

Density of activity can be measured by the clustering coefficient of User nodes. For a User node x with outgoing edges connected to k other User nodes, these k nodes are known as x 's neighbours, and the clustering coefficient of x is defined as the ratio of the total count of directed *InteractsWith* edges amongst neighbours themselves and the maximum possible number directed edges that could exist amongst themselves (which is $(k * (k-1))$ as the denominator).

Note:

A ratio value denotes that it can be between 0 and 1. A value of 0 means that either a node x has no neighbours, or only 1 neighbour, or have at least 2 neighbours but every neighbour is not connected to each other at all. A value of 1 means that a node x has at least 2 neighbours and that every neighbour is bi-directionally connected to each other.

Based on the formula described above, the following query would calculate the Cluster Coefficient of a targeted node with id=123:

```

match (u1:User { id: 123 })-[iw1:InteractsWith]->(u2:User)
with collect(u2.id) as neighbours, count(u2) as k
match (u3:User)-[iw2:InteractsWith]->(u4:User)
where (u3.id in (neighbours)) and (u4.id in (neighbours)) and (u3.id <> u4.id)
with count(iw2) as numerator, (k * (k - 1) * 1.0) as denominator
return numerator/denominator as clusterCoefficient

```

The above query is subsequently used to calculate the Cluster Coefficient for the top 3 chattiest users.

Most Active Users (based on Cluster Coefficients)

User ID	Coefficient
394	0.75
2047	0.92
209*	1
1087*	0.73

**Users 209 and 1087 shared 3rd placing for top chattiest user, as shown in the earlier table, thus both are considered for investigation here*

Cluster Coefficient could potentially serve as a yardstick for identifying high-roller chatty users with strong cliques, and subsequently target them for sharing their in-game purchases as a form of advertisement within Team Chat-sessions.

Scripts

Constraint Creation

```
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;
```

Data Loading & Graph Construction

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

```
LOAD CSV FROM "file:/chat_join_team_chat.csv" AS row  
MERGE (u:User {id: toInt(row[0])})  
MERGE (c:TeamChatSession {id: toInt(row[1])})  
MERGE (u)-[:Join{timeStamp: row[2]}]->(c)
```

```
LOAD CSV FROM "file:/chat_leave_team_chat.csv" AS row  
MERGE (u:User {id: toInt(row[0])})  
MERGE (c:TeamChatSession {id: toInt(row[1])})  
MERGE (u)-[:Leaves{timeStamp: row[2]}]->(c)
```

```
LOAD CSV FROM "file:/chat_item_team_chat.csv" AS row  
MERGE (u:User {id: toInt(row[0])})  
MERGE (c:TeamChatSession {id: toInt(row[1])})
```

```
MERGE (ci:ChatItem {id: toInt(row[2])})
```

```
MERGE (u)-[:CreateChat {timeStamp: row[3]]->(ci)
```

```
MERGE (ci)-[:PartOf{timeStamp: row[3]]->(c)
```

```
LOAD CSV FROM "file:/chat_mention_team_chat.csv" AS row
```

```
MERGE (ci:ChatItem {id: toInt(row[0])})
```

```
MERGE (u:User {id: toInt(row[1])})
```

```
MERGE (ci)-[:Mentioned{timeStamp: row[2]]->(u)
```

```
LOAD CSV FROM "file:/chat_respond_team_chat.csv" AS row
```

```
MERGE (ci1:ChatItem {id: toInt(row[0])})
```

```
MERGE (ci2:ChatItem {id: toInt(row[1])})
```

```
MERGE (ci1)-[:ResponseTo{timeStamp: row[2]]->(ci2)
```

Verification by Counting Nodes (count=45463)

```
start n=node(*)
```

```
match (n)
```

```
return count(n)
```

Verification by Counting Edges (count=118502)

```
MATCH ()-[r]->()
```

```
return count(*);
```

Find Longest Conversation Chain

```
match p=(a)-[:ResponseTo*]->(b)
```

```
RETURN p, length(p)
```

```
ORDER BY length(p) desc limit 1
```

Find All Users Involved In Longest Conversation Chain

match p=(a)-[:ResponseTo*]->(b)

where length(p) = 9

with p

match (u:User)-[]-(d)

where d in nodes(p)

return u

Find Top 10 Chattiest Users

MATCH (u:User)-[r>CreateChat]->(ci:ChatItem)

RETURN u.id as node, count(r) as Outdegree

ORDER BY Outdegree desc LIMIT 10

Find Top 10 Chattiest Teams

MATCH (ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:OwnedBy]->(t:Team)

RETURN t.id as team, count(po) as Outdegree

ORDER BY Outdegree DESC LIMIT 10

Trace Teams of Top 10 Chattiest Users

MATCH

(u:User)-[r>CreateChat]->(ci:ChatItem)-[po:PartOf]->(tcs:TeamChatSession)-[o:OwnedBy]->(t:Team)

RETURN u.id as node, count(r) as Outdegree, t.id as team

ORDER BY Outdegree desc LIMIT 10

Creation of Interaction Edges For Users Who Mention Others

MATCH (u1:User)-[cc>CreateChat]->(ci:ChatItem)-[m:Mentioned]->(u2:User)

WHERE u1.id <> u2.id

MERGE (u1)-[:InteractsWith]->(u2)

Creation of Interaction Edges For Users Who Respond To ChatItem Of Others

MATCH

(u1:User)-[cc1:CreateChat]->(ci1:ChatItem)-[rt:ResponseTo]->(ci2:ChatItem)-[cc2:CreateChat]-(u2:User)

WHERE u1.id <> u2.id

MERGE (u1)-[:InteractsWith]->(u2)

Sample Query For Calculating Cluster Coefficient For A Targeted User

match (u1:User { id: 394 })-[iw1:InteractsWith]->(u2:User)

with collect(u2.id) as neighbours, count(u2) as k

match (u3:User)-[iw2:InteractsWith]->(u4:User)

where (u3.id in (neighbours)) and (u4.id in (neighbours)) and (u3.id <> u4.id)

with count(iw2) as numerator, (k * (k - 1) * 1.0) as denominator

return numerator/denominator as clusterCoefficient