# Graph Analytics of Catch the Pink Flamingo Chat Data Using Neo4j

In the **schema video from Lesson 1 of the Capstone project**, Slide 2 gives a screenshot of the chat graph data that would be created as part of this exercise. These instructions, along with review material from the Graph Analytics course, should help you complete the accompanying quiz. In addition to answering the quiz questions, the results of the queries you submitted for the quiz questions will also be needed for your technical appendix Peer Review submission this week.

## Loading the Chat Data

1. Locate the chat-data folder you downloaded and extracted in Week 1. You should find 6 CSV files in the folder.

2. Start the Neo4J graph database in your Browser. You may want to consult the notes from the Hands-On exercises from the Graph Analytics course.

neo4j_supplementary_resource_pdfs_v2.zip×

3. In the Neo4J system first execute the following instructions:

```
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;
```

4. Load the CSV data files into Neo4J. Please consult the hands on notes from the Graph Analytics course for loading them based on the following directives.

**Example: First file**

The following script loads the first file:

```
LOAD CSV FROM "file:/path/to/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)
```

The first line gives the path of the file. Use the directory path on your machine to replace the "path/to/" string in the LOAD CSV line. This command reads the chat_create_team_chat.csv file one row at a time and creates User nodes. The 0th column value is converted to an integer and is used to populate the id attribute. Similarly the other nodes are created.

Line 4, MERGE (u)-[:CreatesSession{timeStamp: row[3]}]->(c) creates an edge labeled "CreatesSession" between the user node u and the TeamChatSession node c. This edge has a property called timeStamp. This property is filled by the content of column 3 of the same row.

Similarly, the last line creates an edge labeled "OwnedBy" between the TeamChatSession node and the Team node. Copy this script, run it in Neo4j, and verify that these nodes and edges are created.

**Follow the same logic to load the other files based on the following information:**

(i) chat_join_team_chat.csv file creates an edge labeled "Joins" from User to TeamChatSession. The columns are the User id, TeamChatSession id and the timestamp of the Joins edge. For this graph, you would use a similar MERGE statement for User id and TeamChatSession as previously done (take care that column values are slightly different), and you would define your edges labelled "Join" with a similar MERGE statement using the timeStamp as the "OwnedBy" edges were defined previously.

(ii) chat_leave_team_chat.csv file creates an edge labeled "Leaves" from User to TeamChatSession. The columns are the User id, TeamChatSession id and the timestamp of the Leaves edge. For this graph, you would use a similar MERGE statement for User id and TeamChatSession as previously done, and you would define your edges labelled "Leaves" with a similar MERGE statement using the timeStamp as the "Join" edges were defined previously.

(iii) chat_item_team_chat.csv file creates nodes labeled ChatItems. Column 0 is User id, column 1 is the TeamChatSession id, column 2 is the ChatItem id (i.e., the id property of the ChatItem node), column 3 is the timestamp for an edge labeled "CreateChat". Also create an edge labeled "PartOf" from the ChatItem node to the TeamChatSession node. This edge should also have a timeStamp property using the value from Column 3. For this graph, you would use a similar MERGE statement for User id and TeamChatSession as previously done. You would define a third node type, "ChatItem" in a similar way as the Team node in (i) above, and you would define two sets of edges, one labelled "CreateChat" and the other labelled "PartOf" with similar MERGE statements using the timeStamp as the those which were defined previously in (i) above.

(iv) chat_mention_team_chat.csv file creates an edge labeled "Mentioned". Column 0 is the id of the ChatItem, column 1 is the id of the User, and column 2 is the timeStamp of the edge going from the chatItem to the User. For this graph, you would use a similar MERGE statement for User and ChatItem as previously done (take care that column values are slightly different), and you would define your edge labelled "Mentioned" with a similar MERGE statement using the timeStamp as was defined previously.

(v) chat_respond_team_chat.csv file creates an edge labeled "ResponseTo" from a ChatItem node to another ChatItem node. Column 0 has the ID of the first ChatItem node, column 1 has the ID of the second ChatItem node and column 2 has the timeStamp of the edge. For this graph, you would use two MERGE statements for each of the two ChatItem columns as previously done, and you would define your edge labelled "ResponseTo" with a similar MERGE statement using the timeStamp as was defined previously.

5. Execute the load commands one by one and verify that you have 45463 nodes and 118502 edges.

## Assignment Instructions

With this graph now in place, answer the quiz questions in the Quiz for this lesson (after this Reading) based on the following instructions, and document the steps that you took because you will need them for the appendix of your Peer Review submission this week and for your Week 6 final report.

### 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? , and 2) How many users participated in this chain?**

For Part 1, you may recall from the Graph Analytics you learned how to calculate the length of a path between specific nodes with:

```
match p=(a)-[:TO*]-(c)

where a.Name='H' and c.Name='P'

return length(p) limit 1
```

You should be able to adapt this script by modifying the edge value in the square brackets, ordering the results in descending order and limiting the results to a single value, which should be the length of the longest path.

For Part 2, you will use a similar MATCH statement but you will refine the search for only those paths whose length equals the value from Part 1. You will then need to use the "WITH p" command and a second MATCH statement to find all users with a CreateChat edge for the longest path p, and finally you will also need to use the count(distinct u) command to return a value of 1 for each of the unique users in the longest chat.

You will report two values for your results, the number of items in the longest chat, and the number of unique users in the longest chat, and you will include a screen capture of your longest path. You will also report how the results of this kind of search may be relevant to Eglence Inc. business plan.

## Question 2:

**Do the top 10 the chattiest users belong to the top 10 chattiest teams?**

For this question you will need to perform two separate queries, 1) identify the top 10 chattiest users, and 2) identify the top 10 chattiest teams.

To find the chattiest users, you need to MATCH all Users with a CreateChat edge, return the Users (their ID) and the count of the Users and sort them in descending order and limit the results to 10.

The query to find the top 10 chattiest teams is a bit more complex. You need to MATCH all ChatItems with a PartOf edge connecting them with a TeamChatSession node AND the TeamChatSession nodes must have an OwnedBy edge connecting them with any other node. Thus your query would look something like:

ChatItem-PartOf->TeamChatSession-OwnedBy->n

You will report the top 3 results for each query and identify which, if any, chattiest users belong to chattiest teams. You will also report how this kind of search may be relevant to Eglence Inc. business plan.

## Question 3:

**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.

(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. For the first condition, this query would have the following structure:

Match (u1:User)-SOMETHING IN THE MIDDLE-[:Mentioned]->(u2:User) create (u1)-[:InteractsWith]->(u2)

You will complete the query by filling in the SOMETHING IN THE MIDDLE part with a Cypher query fragment. You may want to write down how two users are connected through mention by reviewing the load script you created.

Use the same logic to create the query statement for the second condition. This query will also have the form

MATCH PORTION followed create (u1)-[:InteractsWith]->(u2)

(b) The above scheme will create an undesirable side effect if a user has responded to her own chatItem, because it will create a self loop between two users. So after the first two steps we need to eliminate all self loops involving the edge “Interacts with”. This will take the form:

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

(c) Given this new edge type, we will have to create a scoring mechanism to find users having dense neighborhoods. The score should range from 0 (a disconnected user) to 1 (a user in a clique – where every node is connected to every other node). One such scoring scheme is called a “clustering coefficient” defined as follows. If the number of neighbors of node is 5, then the clustering coefficient of the node is the ratio between the number of edges amongst these 5 neighbors(not counting the given node) and 5*4 (all the pairwise edges that could possibly exist). Thus the denominator is k * (k-1) if the number of neighbors of the node is k.

Your task in this question is to find the clustering coefficients of the chattiest users (you know their ids) from Q2.

(d) To do this computation, we need to

· get the list of neighbors and

· the number of neighbors of a node based on the "InteractsWith" edge.

For each of these neighbors, we need to find

· The number of edges it has with the other members on the same list.

· If one member has multiple edges with another member we need to count it as 1 because we care only if the edge exists or not

We then need to add the edges we get for each member and divide this by k*(K-1)

(e) To program this computation in Cypher you will need to review a few constructs you have seen before.

// This shows how the "with" clause is used to pass variable from one match query fragment to another match query fragment and then return the results with a "RETURN" statement

match (n:MyNode)-[r]-()

with n, count(distinct r) as degree

set n.deg = degree

return n.Name, n.deg

// This shows how the "Case" statement is used to perform an if-then-else kind of value reporting

match (n:MyNode), (m:MyNode)

return n.Name, m.Name,

```
case

when (n)-->(m) then 1

else 0

end as value


// This shows how to collect neighbors of a node and test if the neighbors belong to a list.

match (d {Name:'D'})-[:TO]-(b)

with collect(distinct b.Name) as neighbors

match (n)-[r:TO]->(m)

where

not (n.Name in (neighbors+'D'))

and

not (m.Name in (neighbors+'D'))

return n, r, m
```