

Graph Analytics Solution Report

Modelling Chat Data using a Graph Data Model

The graph model of the Chat Data for the players of the Flamingo game is a visual representation of the csv data files generated using scripts from the game software. The data obtained is purely technical but when modelled as a graph it can present a lot of valuable information on the players, their chat patterns, group behaviour and connectedness.

Using Graph Analytics on the neo4J Tool data can be conceptualized as entities (nodes) which are related by relations (edges). These nodes have properties and along with the edges they form graphs. Graphs provide the analytical base to derive meaningful insight about the various elements modelled within it.

Creation of the Graph Database for Chats

Input Data Set for the Graph Database:

The simulated game data consists of 6 data files in csv format whose contents and usage are as follows:

File1: chat_create_team_chat.csv

Relevance to graph schema: A line is added to this file when a player creates a new chat with their team.

Example data:

userid, teamid, timestamp

559,48,6288

876,15,6289

1166,68,6290

File2: chat_item_team_chat.csv

Relevance to graph schema: 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.

Example Data:

userid, teamid, timestamp

1956,6299,6305

2081,6296,6311

1166,6290,6316

File3: chat_join_team_chat.csv

Relevance to graph schema: Creates an edge labeled "Joins" from User to TeamChatSession. The columns are the User id, TeamChatSession id and the timestamp of the Joins edge.

Example Data:

userid, TeamChatSessionID

559,6288

876,6289

1166,6290

File4: chat_leave_team_chat.csv

Relevance to graph schema: Creates an edge labeled "Leaves" from User to TeamChatSession. The columns are the User id, TeamChatSession id and the timestamp of the Leaves edge.

Example Data:

userid, chatid, timestamp

1244,6821,1464241204.0

1074,6838,1464243024.0

350,6777,1464246654.0

File5: chat_mention_team_chat.csv

Relevance to graph schema: 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.

Example Data:

ChatItem, userid, timeStamp

6349,2508

6366,2491

6371,104

File6: chat_respond_team_chat.csv

Relevance to graph schema: A line is added to this file when a player responds to a chat post.

Example Data:

userid1, userid2

6326,6305

6364,6326

6371,6366

Loading CSV Data into Neo4J:

Neo4J has the capability to directly upload data from the csv files above via **CYPHER** (declarative Query language for neo4J) commands and can pick files from a specified location. Using the uploaded data, graph elements like nodes, relations, properties etc. can be easily defined. The neo4j load command – **LOAD CSV**- is used as shown below:

-----Load File 1: chat_create_team_chat.csv-----

```
LOAD CSV FROM "file:///chat-data/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 File 3: chat_join_team_chat.csv-----

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

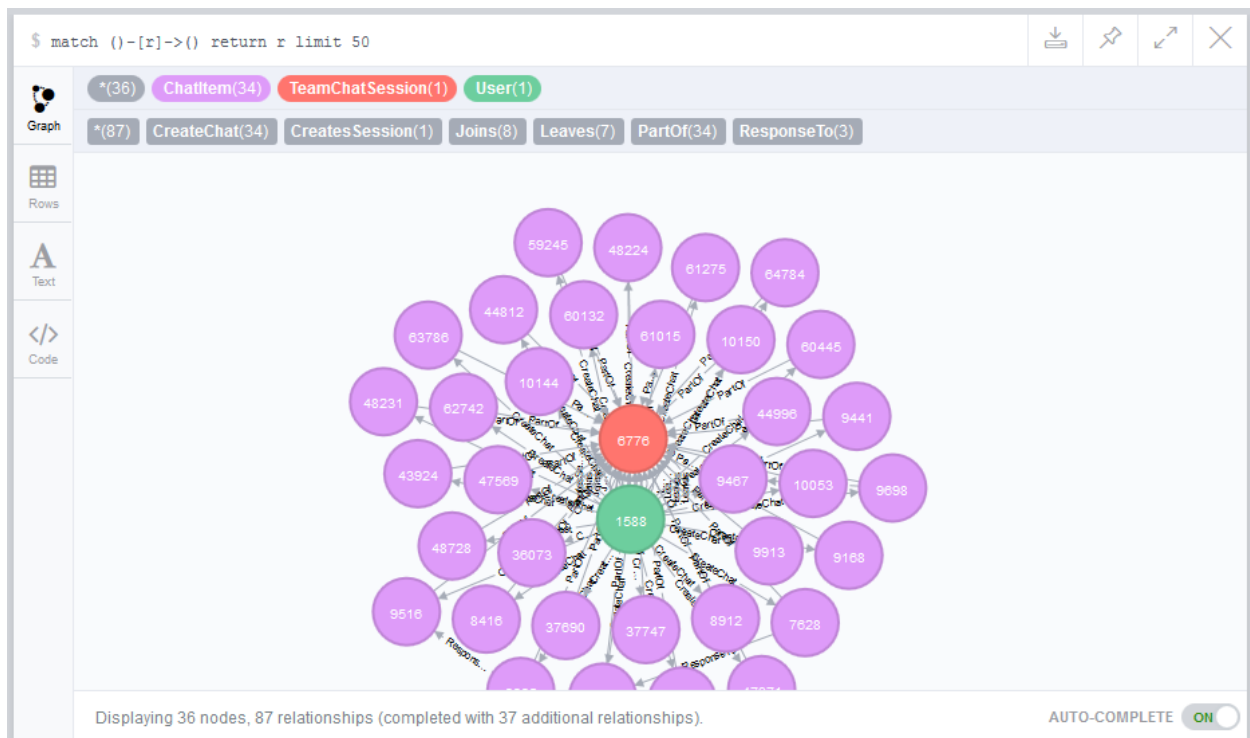
Note that the file path as specified in the first line "[file:///](#)" is relative to the installation folder of the neo4j graph database. The **MERGE** command appends individual data elements from the file into the graph nodes/edges as applicable. A timestamp value is used as a **property key** to connect relations among nodes.

A special mention must be made about the Constraint commands below. These *ensure the integrity/uniqueness of individual node elements* like **User, Team, ChatItem** etc. which are being appended into the database. If we don't use them, we will end up with duplicate node items.

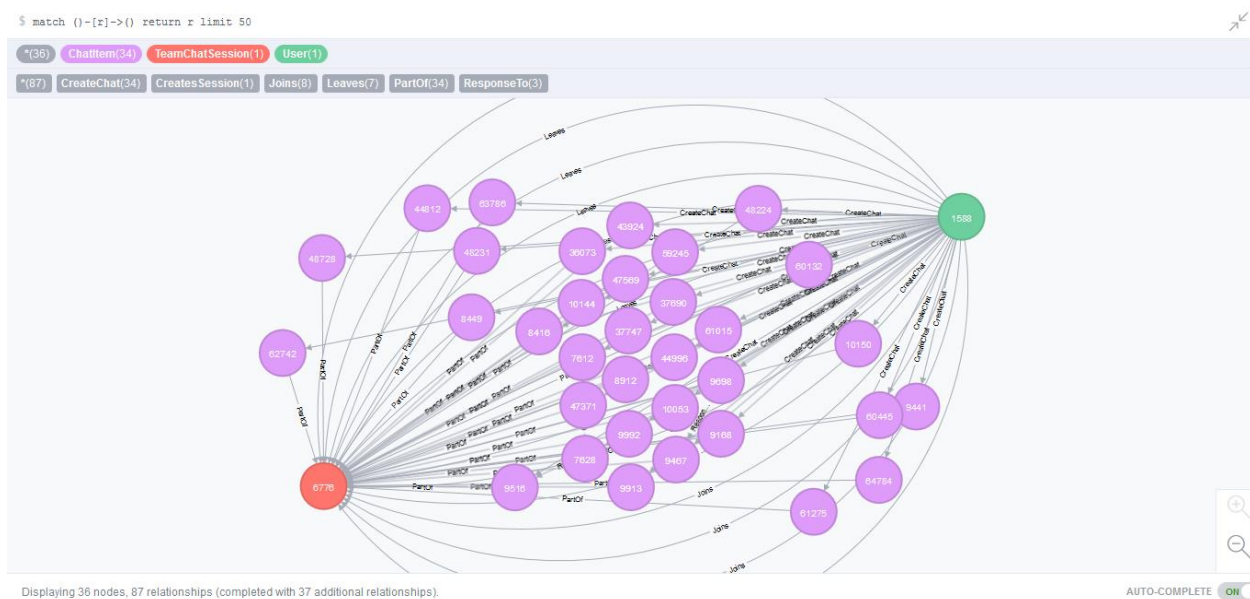
```
CREATE CONSTRAINT ON (u:User) ASSERT u.id IS UNIQUE;
CREATE CONSTRAINT ON (t:Team) ASSERT t.id IS UNIQUE;
CREATE CONSTRAINT ON (c:TeamChatSession) ASSERT c.id IS UNIQUE;
CREATE CONSTRAINT ON (i:ChatItem) ASSERT i.id IS UNIQUE;
```

Visual Overview of resultant graph model in Neo4J:

Screen1: Part of graph model showing all types of nodes/edges



Screen2: Part of graph model showing close-up view of certain major nodes and edges among them



Finding the longest conversation chain and its participants

Scenario Description and Analysis Background:

The graph consists of **ChatItem** nodes which represent a unique chat content created by a **User** which is represented by another type of node. In this graph only a **User can create a ChatItem**. Also a **ChatItem** can be **a response to** another **ChatItem** resulting in a series or chain of chats. With this background, the 2 basic objectives of this analysis are i) find the length of the path which shows the longest chat, i.e. a continuous series of related **ChatItem nodes** linked to each other and ii) Which are the unique set of users (**User nodes**) involved in this chat chain-

Solution commands and Graph screenshots

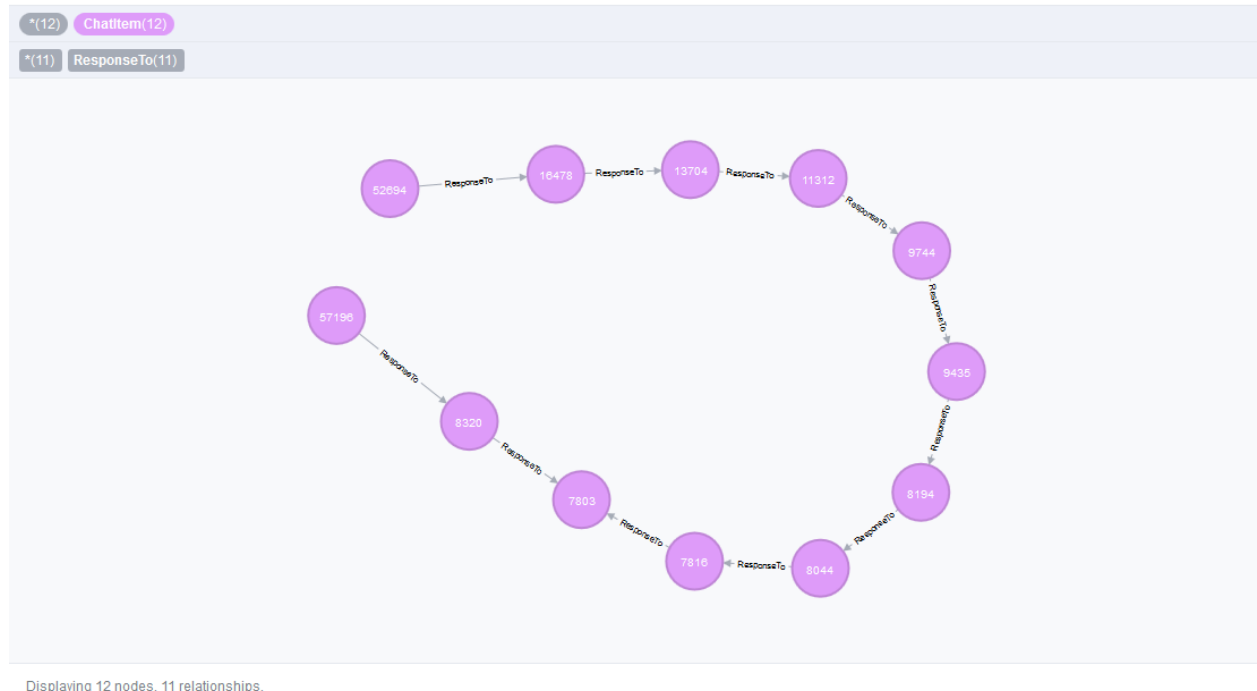
- i) Length of the path of largest chat series is **11** and consists of 12 ChatItem nodes as can be seen from screenshots below:

CYPHER Command:

```
match p=(c1:ChatItem)-[:ResponseTo*]-(c2:ChatItem)
where c1<>c2
return p,length(p) order by length(p) desc limit 1
```

Graph output showing the longest chat path with **ChatItem Nodes(12)** and the directed relation **ResponseTo** between them

```
$ match p=(c1:ChatItem)-[:ResponseTo*]-(c2:ChatItem) where c1<>c2 return p order by length(p) desc limit 1
```



- ii) Number of unique users on the longest chat path above is **6** as shown below

CYPHER Command:

```
match p=(c1:ChatItem)-[:ResponseTo*]-(c2:ChatItem)
*where length(p)=11 and c1<>c2
with extract(n in nodes(p)|n.id) as nodeid
match (u:User)-[:CreateChat*]-(c3:ChatItem)
where c3.id IN nodeid
return count(distinct u
```

***Note:** The length(p)=11 has been determined from the cypher query results in part i)

Count of Unique **User nodes**

```
1 match p=(c1:ChatItem)-[:ResponseTo*]-(c2:ChatItem)
2 where length(p)=11 and c1<>c2
3 with extract(n in nodes(p)|n.id) as nodeid
4 match (u:User)-[:CreateChat*]-(c3:ChatItem)
5 where c3.id IN nodeid
6 return count(distinct u)
```




```
$ match p=(c1:ChatItem)-[:ResponseTo*]-(c2:ChatItem) where length(p)=11 and c1<>c2 with extract(n in nodes(p)|n.id) as nodeid match (u:User)-[...
```

	count(distinct u)
Rows	6

List of unique User Ids is below

```
1 match p=(c1:ChatItem)-[:ResponseTo*]-(c2:ChatItem)
2 where length(p)=11 and c1<>c2
3 with extract(n in nodes(p)|n.id) as nodeid
4 match (u:User)-[:CreateChat*]-(c3:ChatItem)
5 where c3.id IN nodeid
6 return distinct u.id
```

```
$ match p=(c1:ChatItem)-[:ResponseTo*]-(c2:ChatItem) where length(p)=11 and c1<>c2 with extract(n in nodes(p)|n.id) as nodeid match (u:User)-[...
```

	u.id
Rows	1192
	853
Text	1514
	754
Code	1153
	1978

Solution significance to Eglenec Inc: Long chats can signify that topics of interest are being discussed. The users involved are perhaps the most active contributors in the forum. By looking at the chat topics (not violating privacy laws!) and the list of

unique users one might conclude that these users might be interested in products/services linked to these topics in general. They can then be targeted for promotions linked to such items.

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

Scenario Description and Analysis Background:

The chattiest users are the most active and can be influential, if they create convincing/interesting chat items. Here the tasks are to: 1) identify the top 10 chattiest users, and 2) identify the top 10 chattiest teams.

Solution commands and Graph screenshots

i) Top 10 chattiest users:

CYPHER Command:

```
match (u:User)-[r:CreateChat*]-(a:ChatItem)
return u.id, count(r) order by count(r) desc limit 10
```

Result of query shown above



The screenshot shows a graph database interface with a query editor and a results table. The query is: `1 match (u:User)-[r:CreateChat*]-(a:ChatItem)` and `2 return u.id, count(r) order by count(r) desc limit 10`. The results table has two columns: `u.id` and `count(r)`. It displays 10 rows of data, sorted by `count(r)` in descending order. The interface also shows a status bar at the bottom indicating 'Returned 10 rows in 1522 ms.'

	u.id	count(r)
Rows	394	115
Text	2067	111
	209	109
Code	1087	109
	554	107
	516	105
	1627	105
	999	105
	668	104
	461	104
	Returned 10 rows in 1522 ms.	

In the query above we find all chat items created by all users and then count them per user. Finally, we order the results from highest to lowest and find the Top 10. From the list first 3 are chosen for further analysis and displayed in the table below

Chattiest Users

Users	Number of Chats
394	115
2067	111
*209 & 1087	109

*There is a tie for 3rd place in table above

ii) Top 10 chattiest Teams:

CYPHER Command:

```
match (c:ChatItem)-[r:PartOf*]-(t:TeamChatSession)-[s:OwnedBy*]-(n)
return n.id,count(r) order by count(r) desc limit 10
```

In the query above, we find the paths where a chat item is part of a Team's chat session and then track the Team node based on the **OwnedBy** relationship. Then we count (for each team), how many chat items are indirectly owned by it. Finally, we order the list from high to low values and then pick the top 3 for further analysis.

Query results:

```
1 match (c:ChatItem)-[r:PartOf*]-(t:TeamChatSession)-[s:OwnedBy*]-(n)
2 return n.id,count(r) order by count(r) desc limit 10
```

```
$ match (c:ChatItem)-[r:PartOf*]-(t:TeamChatSession)-[s:OwnedBy]-(n) return n.id,count(r) order by count(r) desc limit 10
```

	n.id	count(r)
Rows	82	1324
A	185	1036
Text	112	957
</>	18	844
Code	194	836
	129	814
	52	788
	136	783
	146	746
	81	736

Returned 10 rows in 276 ms.

Chattiest Teams


Teams	Number of Chats
82	1324
185	1036
112	957

```
match (u:User)-[r:CreateChat*]-(a:ChatItem)-[s:PartOf*]-(t:TeamChatSession)-[o:OwnedBy*]-(n)
where u.id =394 or u.id=2067 or u.id=209
return distinct n.id
```

In the query above we try and link the top 3 users to the team they belong to. This is a linked search and finally shows the team IDs as shown below. One can observe that none among the Top 3 users belong to Top 3 chattiest Teams

```
1 match (u:User)-[r:CreateChat*]-(a:ChatItem)-[s:PartOf*]-(t:TeamChatSession)-[o:OwnedBy*]-(n)
2 where u.id =394 or u.id=2067 or u.id=209 or u.id=1087
3 return distinct n.id
```

\$ match (u:User)-[r:CreateChat*]-(a:ChatItem)-[s:PartOf*]-(t:TeamChatSession)-[o:OwnedBy*]-(n) where u.id =394 or u.id=2067 or u.id=209 or u.i...

	n.id
Rows	63
	7
Text	77
	
Code	

Returned 3 rows in 924 ms.

Solution significance to Eglence Inc: It can be seen above that there might be active users even in teams which are not that active collectively. One might just use the Top 3 teams for marketing campaigns and might have missed more active users. The Top users as well as Top Teams can be targeted with discount coupons/bulk deals so as to encourage users from the relative dormant group (to which Top3 users belong) to get motivated as well (to accept the offers!)

Analyzing the most active users and their neighbourhood

Scenario Description and Analysis Background:

In this scenario, we are required to 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.

Solution commands and Graph screenshots

Step 1: Create a new relationship called ‘InteractsWith’ among users based on following rules:

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

These can be accomplished using the Cypher queries below:

```
MATCH p=(u1:User)-[r:Mentioned]->(c:ChatItem)<-[s:Mentioned]-(u2:User)
where u1<>u2
create (u1)-[:InteractsWith]->(u2)---for part a
```

```
MATCH p=(u1:User)-[:CreateChat]-(c1:ChatItem)-[r:ResponseTo]->(c2:ChatItem)<-
[s:CreateChat]-(u2:User)
where u1<>u2 and c1<>c2
create (u1)-[:InteractsWith]->(u2)----for part b
```

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

Step 2: Using the new edge type-‘InteractWith’ created in **Step 1**, 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 .

Note that a neighbour is defined as an adjoining node which is linked to a particular node by a single relationship edge.

In the CYPHER queries below, I have calculated the clique coefficient as explained above, for the Top 3 users, i.e. userid **394, 2067 and 209** as follows: The coefficient formula is: **Total**

Number of edges between neighbours(excluding the node)/(K*(K-1)) where K is the number of neighbours for that particular user.

```
match (d {id:394})-[:InteractsWith]-(b)
return collect(distinct b.id) as neighbors---3 neighbors;k=3
```

```
match (d {id:394})-[:InteractsWith]-(b)
with collect(distinct b.id) as neighbors
match (n)-[r:InteractsWith]->(m)
where n.id in (neighbors) and m.id in (neighbors) and n<>m
return count(distinct r)--5/{k*(K-1)}=5/3*2=5/6=0.833
```

```
match (d {id:2067})-[:InteractsWith]-(b)
return collect(distinct b.id) as neighbors---8 neighbors;k=8
```

```
match (d {id:2067})-[:InteractsWith]-(b)
with collect(distinct b.id) as neighbors
match (n)-[r:InteractsWith]->(m)
where n.id in (neighbors) and m.id in (neighbors) and n<>m
return count(distinct r)--26/{k*(K-1)}=26/8*7=26/56=0.464
```

```
match (d {id:209})-[:InteractsWith]-(b)
return collect(distinct b.id) as neighbors---7 neighbors;k=7
```

```
match (d {id:209})-[:InteractsWith]-(b)
with collect(distinct b.id) as neighbors
match (n)-[r:InteractsWith]->(m)
where n.id in (neighbors) and m.id in (neighbors) and n<>m
return count(distinct r)--27/{k*(K-1)}=27/7*6=27/42=0.643
```

As can be seen from the calculations above, the user with ID=394 is in a dense neighbourhood (clique coefficient=.833). This neighbourhood thus has a good marketing potential!

A screenshot of this **most active neighbourhood** is shown in the next section:

