SDM Project

Margarita Hernández
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
maritahc99@gmail.com

Yanjian Zhang
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
yanjian.zhang@outlook.com

In this document, we will be discussing about the Semantic Data Management part for the Joint Project in BDMA. Fisrt of all, we came up with a business idea that we would expand on in our VBP course. For BDM, we extracted and managed data that we would need to develop this project. And finally, in SDM course we want to integrate and improve with the use of semantics. Code and Data are provided in our Github repository[1]

M1. Purpose Statement

Our business idea consists of generating personalised recommendations to users based on tags extracted from people's experiences in restaurants, bars and cafes in Barcelona. These recommendations will have a high affinity with the user once they start interacting with the app, so that they match their personal preferences.

In our BDM course, we focused on extracting reviews from different sources to obtain the mentioned tags to get our app started. This means that once the app would be launched, our users would start creating their own tags. So, unlike our proof of concept in BDM, where we used data from TripAdvisor, Google Maps and Yelp, in the future, we would only have one source: data generated from our ForMe App.

Based on the above information, we believe graph-based analytics is the most appropriate approach for our data. The ultimate goal would be to create a recommendation system with which to suggest relevant establishments for each user of our app.

M2. Graph Family

After extracting the tags from establishments and users, this data would be stored in a property graph, where some analytics (e.g., recommendation system) could be performed to improve the functionalities of the application. Moreover, as mentioned before, we would have only one source of data input(The integration for different data source was implemented in BDM Project with Spark). This means that data integration performed with knowledge graph wouldn't be necessary.

Regarding the language that will be used to query the property graph, this will be Cypher on top of Neo4j graph database. Cypher could support diverse multi-hop query and various graph algorithm. This way, we will have the opportunity to expand our knowledge using this technology and language in our own entrepreneurial idea.

M3. Graph Design

Making use of the extracted data from TripAdvisor, Yelp and Google Maps, we have designed the following graph, which we think would give us the change to use its topology and exploit it in a convenient way for our business idea.

[1] https://github.com/yanjianzhang/FIB-Course-Project/tree/master/Semantic__Data__Management/Final__PJ

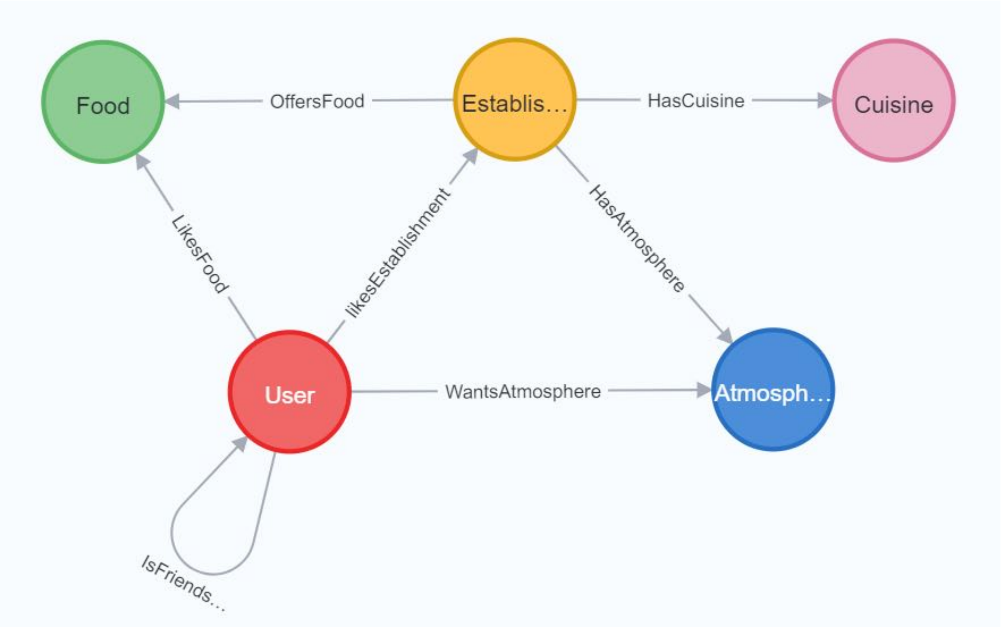


Figure 1: Graph Design

M4. Populate the graph

The overall pipeline can be seen from Figure 2.

Before creating and populating the graph, we had to make sure that our data had the right format for it.

In BDM Part 2 we extracted relevant tags using Named-entity recognition (NER), AutoPhrase and TF-IDF algorithms. However, semantics need to go beyond that: tags must be grouped by meaning. Extracting only meaningful words is not so straightforward.

We have therefore decided to use datasets that are available in: data world[2] and kaggle[3]. The former one was used to create the node named food, the latter for cuisine (see Figure 1). These sources were used as dictionaries as "right" options. We then joined our original table (having merged the tags obtained from the three algorithms previously mentioned), with these dictionaries, keeping only those present in the latter.

Moreover, to create the edge named :*likesEstablishment* we assumed that only those reviews with a rating of exactly 5 were "liked" by a user. Following this logic, the edges named :*WantsAtmosphere* and :*LikesFood* were derived from that subset of reviews.

Finally, in order to create all 5 nodes and 7 edges, one csv was generated for each. If any of these did not have a unique identifier, one was randomly created. All edges could be successfully created with the information retrieved from BDM Project, except for *IsFriendsWith*, since Google Maps does not link users' profiles. In this case, relationships were synthetically generated.

[2] https://data.world/kgarrett/whats-on-the-menu/workspace/query?queryid=5d1d1edb-5e6c-4bc7-bf9e-b128f846b959

[3] https://www.kaggle.com/datasets/ngshiheng/michelin-guide-restaurants-2021

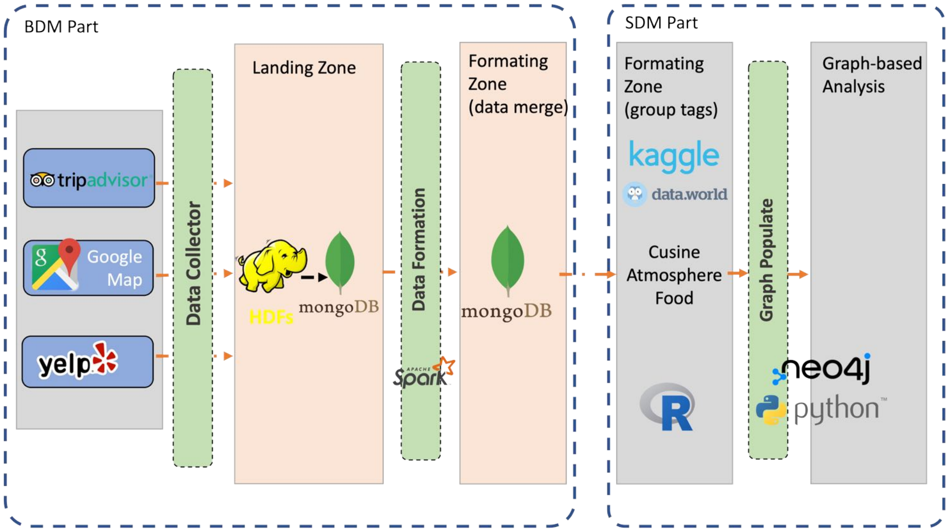


Figure 2: Overall Pipeline

M5. Exploit the graph

Restaurant similarity

We want to find out the pairs of restaurant that have the similar kind of **Cuisine** and **Atmosphere**. The similarity can be used to recommend restaurants based on user's historical records. We calculate the similarity score based on the following steps:

- Create the similarity graph 1 based on Establishment and Atmosphere over "HasAtmosphere" Relationship.
- Calculate the atmosphere similarity score based on graph 1.
- Create the similarity graph 2 based on Establishment and Cusine over "HasCuisine" Relationship.
- Calculate the cuisine similarity score based on graph 2.
- Sum atmosphere similarity score and cuisine similarity score, ranking the establishment with the highest sum score.

Influential User Finding

For each restaurant, they may want to find the most influential user in the social network who likes them. This information can be further used for advertisement. To find the user, we go with the following steps:

- Run Pagerank inside all the social network, and get the score of each user.
- Matching the restaurant with the user who likes them.
- For each restaurant, find the user with the highest score in Pagerank.

User Interaction

We believe it would be interesting to know which users are interacting the most with the app; this means adding tags, liking establishments, uploading images, etc. Since our aim is to make our recommendations as accurate as possible, we have to encourage users to give feedback. If we can find those who are posting the most, we would be able to give them rewards like getting the add-free version of the app, unlocking premium avatars, etc.

For this task we have decided to use the **outdegree** algorithm. It is a Centrality algorithm, using adjacency. In this case, we only consider those edges that are leaving the node *User*, which represent the activity of the user.

In order to do this, we perform the following steps inside a query:

- Count the number of edges following the three different paths as when a user interacts with the app (liking, recommending, etc.)
- Aggregate all of them as Outdegree of each user node.

M6. Metadata Description

This section is not applicable for our scenario.

M7. Proof of Concept

Environment Setting

The implementation environment setting is as following:

- System: Mac OS with M1 silicon.
- Neo4j DBMS System: 4.4.3
- Neo4j Plugin APOC 4.4.0.4
- Graph Data Science Library 1.8.3

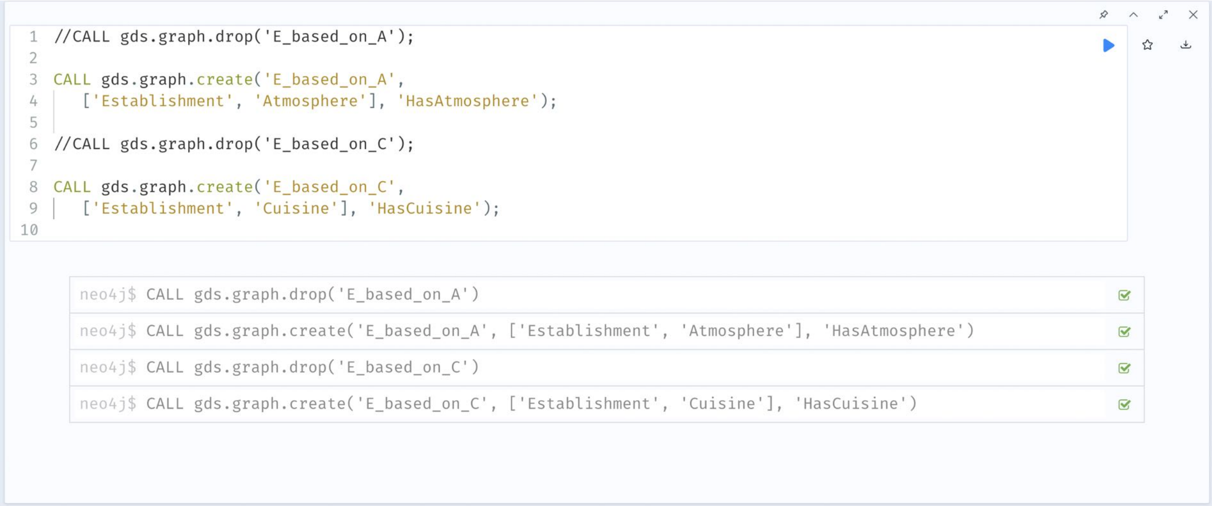
We used R to do this formatting. Code is available in the R script named *create_csvs.R*. And we use Python to integrate Neo4j query to populate the data into the graph.

Tool Justification

Neo4j and Cypher could support diverse multi-hop query and various graph algorithm, which is necessary for implementation our business idea.

Analysis 1. Restaurant similarity

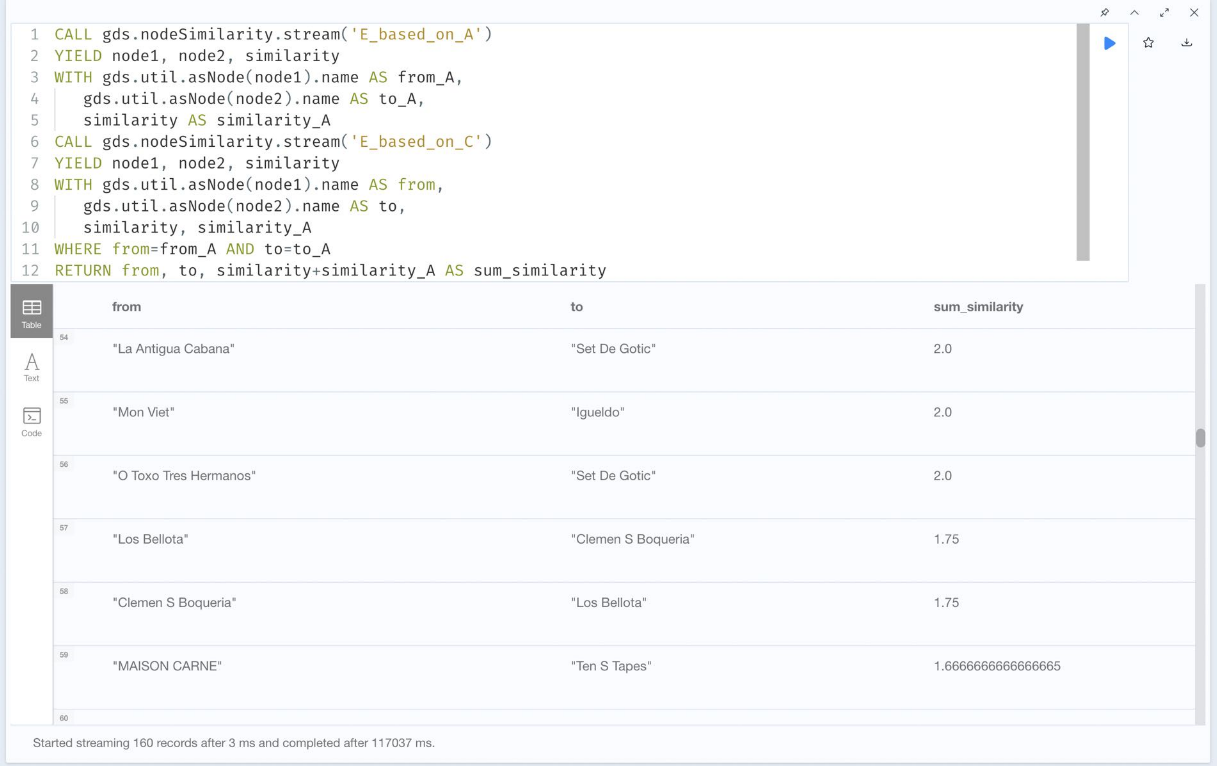
The creation of the Graph and the Similartiy Query can be seen from the Figure 3 and Figure 4.



```
//CALL gds.graph.drop('E_based_on_A');

CALL gds.graph.create('E_based_on_A',
   ['Establishment', 'Atmosphere'], 'HasAtmosphere');

//CALL gds.graph.drop('E_based_on_C');

CALL gds.graph.create('E_based_on_C',
   ['Establishment', 'Cuisine'], 'HasCuisine');

```

```
neo4j$ CALL gds.graph.drop('E_based_on_A')                                                      ☑
neo4j$ CALL gds.graph.create('E_based_on_A', ['Establishment', 'Atmosphere'], 'HasAtmosphere')  ☑
neo4j$ CALL gds.graph.drop('E_based_on_C')                                                      ☑
neo4j$ CALL gds.graph.create('E_based_on_C', ['Establishment', 'Cuisine'], 'HasCuisine')        ☑
```

Figure 3: Similarity Graph Initialization



```
CALL gds.nodeSimilarity.stream('E_based_on_A')
YIELD node1, node2, similarity
WITH gds.util.asNode(node1).name AS from_A,
  gds.util.asNode(node2).name AS to_A,
  similarity AS similarity_A
CALL gds.nodeSimilarity.stream('E_based_on_C')
YIELD node1, node2, similarity
WITH gds.util.asNode(node1).name AS from,
  gds.util.asNode(node2).name AS to,
  similarity, similarity_A
WHERE from=from_A AND to=to_A
RETURN from, to, similarity+similarity_A AS sum_similarity
```

	from	to	sum_similarity
54	"La Antigua Cabana"	"Set De Gotic"	2.0
55	"Mon Viet"	"Igueldo"	2.0
56	"O Toxo Tres Hermanos"	"Set De Gotic"	2.0
57	"Los Bellota"	"Clemen S Boqueria"	1.75
58	"Clemen S Boqueria"	"Los Bellota"	1.75
59	"MAISON CARNE"	"Ten S Tapes"	1.6666666666666665
60			

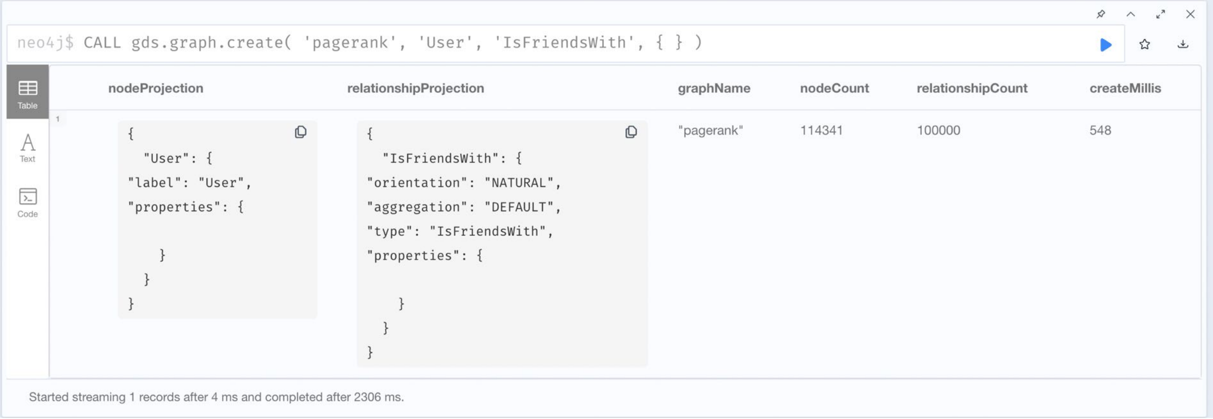
Started streaming 160 records after 3 ms and completed after 117037 ms.

Figure 4: Similarity Graph Query and Result

As shown in the result of figure, the restaurants pairs we extracted can be used to recommend to user. We could assume that if use like one of them, they are very likely to like the other one.

Analysis 2. Influential User Finding

The creation of the Graph and the Query of Influential User Finding can be seen from the Figure 5 and Figure 6.



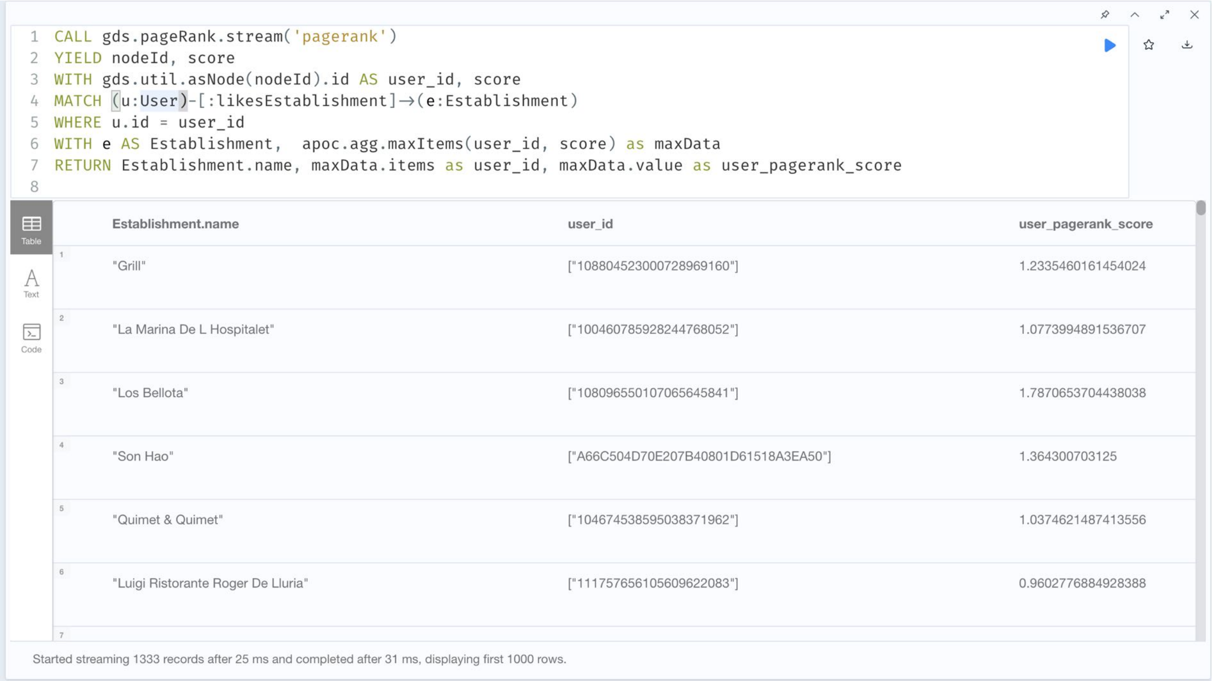
```
neo4j$ CALL gds.graph.create( 'pagerank', 'User', 'IsFriendsWith', { } )
```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	createMillis

1	{
	"User": {
	"label": "User",
	"properties": {
	}
	}
	}
		{
		"IsFriendsWith": {
		"orientation": "NATURAL",
		"aggregation": "DEFAULT",
		"type": "IsFriendsWith",
		"properties": {
		}
		}
		}
			"pagerank"	114341	100000	548

Started streaming 1 records after 4 ms and completed after 2306 ms.

Figure 5: Influential User Finding Initial



```
CALL gds.pageRank.stream('pagerank')
YIELD nodeId, score
WITH gds.util.asNode(nodeId).id AS user_id, score
MATCH (u:User)-[:likesEstablishment]→(e:Establishment)
WHERE u.id = user_id
WITH e AS Establishment,  apoc.agg.maxItems(user_id, score) as maxData
RETURN Establishment.name, maxData.items as user_id, maxData.value as user_pagerank_score

```

Establishment.name	user_id	user_pagerank_score
"Grill"	["108804523000728969160"]	1.2335460161454024
"La Marina De L Hospitalet"	["100460785928244768052"]	1.0773994891536707
"Los Bellota"	["108096550107065645841"]	1.7870653704438038
"Son Hao"	["A66C504D70E207B40801D61518A3EA50"]	1.364300703125
"Quimet & Quimet"	["104674538595038371962"]	1.0374621487413556
"Luigi Ristorante Roger De Lluria"	["111757656105609622083"]	0.9602776884928388

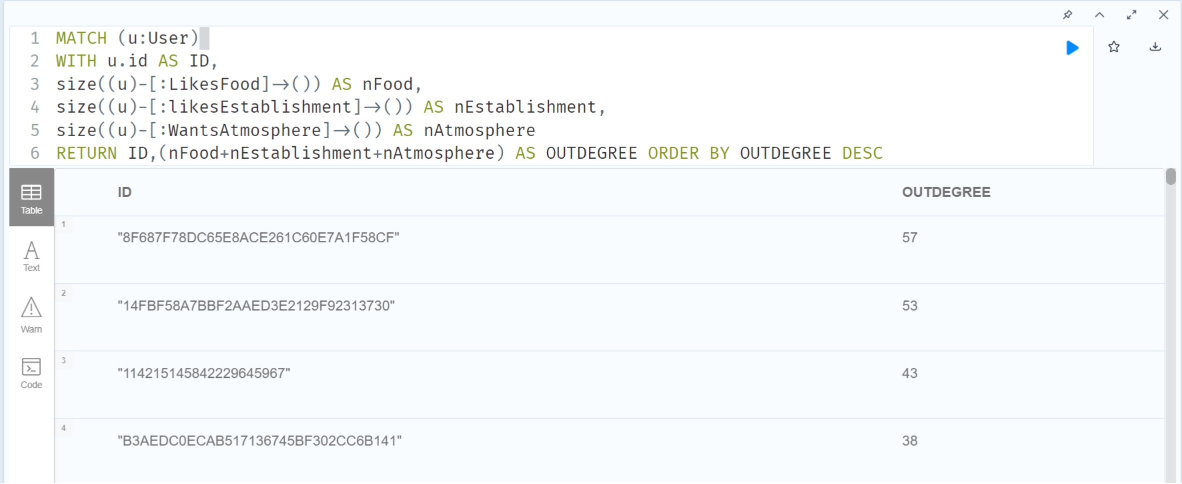
Started streaming 1333 records after 25 ms and completed after 31 ms, displaying first 1000 rows.

Figure 6: Influential User Finding Query

As shown in the result of figure, we could find the most popular user(with the most friends) for each restaurants. Thus, we could target on advertise the restaurant on the user to help them achieve higher popularity in community.

Analysis 3. User Interaction

The implementation of the outdegree algorithm for counting user interaction can be seen in Figure 7 below.



```
MATCH (u:User)
WITH u.id AS ID,
size((u)-[:LikesFood]→()) AS nFood,
size((u)-[:likesEstablishment]→()) AS nEstablishment,
size((u)-[:WantsAtmosphere]→()) AS nAtmosphere
RETURN ID,(nFood+nEstablishment+nAtmosphere) AS OUTDEGREE ORDER BY OUTDEGREE DESC
```

ID	OUTDEGREE
"8F687F78DC65E8ACE261C60E7A1F58CF"	57
"14FBF58A7BBF2AAED3E2129F92313730"	53
"114215145842229645967"	43
"B3AEDC0ECAB517136745BF302CC6B141"	38

Figure 7: Outdegree Algorithm

As shown in the result of figure, we could know the user id which have interact most on our application(choose the tags for restaurant). We would therefore provide the reward to them to encourage more user engagement