| Module Code | B9AI101\_2122\_TMD2 |
| --- | --- |
| Module Name | Graph And AI |
| Date | 05/05/2022 |
| Student number | 10595858 |
| Student name | Sunil Judhistira Gauda |

*By uploading this exam from my Moodle account I Sunil Judhistira Gauda am confirming that this document is all my own work.*

*I understand that DBS will carry out checks such as text-matching (via Urkund), bench-marking, and viva voce exams in order to verify the authenticity of submissions.*

Please input your answers below. You may answer the questions in any order but you must ensure they are clearly labeled.

**Question 1.**

1. **Dijkstra's Shortest Path**

| Origin Vertex | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | Infinity |  |
| Node C | Infinity |  |
| Node D | Infinity |  |
| Node E | Infinity |  |
| Node F | Infinity |  |
| Node G | Infinity |  |

Visited Nodes : Null  
Unvisisted Nodes: A,B,C,D,E,F,G

| Origin Vertex | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | 2 | A |
| Node C | 6 | A |
| Node D | 8 | A |
| Node E | Infinity |  |
| Node F | Infinity |  |
| Node G | Infinity |  |

Visited Nodes : A  
Unvisisted Nodes: B,C,D,E,F,G

| Origin Vertex | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | 2 | A |
| Node C | 6 | A |
| Node D | 8 | A |
| Node E | Infinity |  |
| Node F | Infinity |  |
| Node G | 12 | B |

Visited Nodes : A, B  
Unvisisted Nodes: C,D,E,F,G

| Origin Vertex | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | 2 | A |
| Node C | 6 | A |
| Node D | 7 | C |
| Node E | 11 | C |
| Node F | 9 | C |
| Node G | 12 | B |

Visited Nodes : A, B, C, D  
Unvisisted Nodes: D, E,F,G

| Origin Vertex - D | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | 2 | A |
| Node C | 6 | A |
| Node D | 7 | C |
| Node E | 11 | C |
| Node F | 9 | C |
| Node G | 12 | B |

Visited Nodes : A, B, C, D

Unvisisted Nodes: E, F, G

| Origin Vertex - E | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | 2 | A |
| Node C | 6 | A |
| Node D | 7 | C |
| Node E | 11 | C |
| Node F | 9 | C |
| Node G | 12 | B |

Visited Nodes: A, B, C, D, E  
Unvisisted Nodes: F, G

| Origin Vertex - F | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | 2 | A |
| Node C | 6 | A |
| Node D | 7 | C |
| Node E | 10 | F |
| Node F | 9 | C |
| Node G | 12 | B |

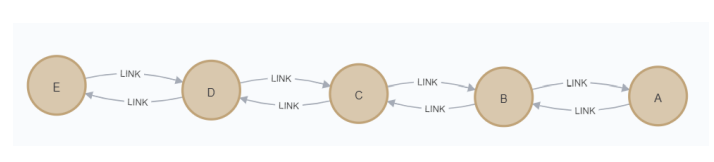
Visited Nodes: A, B, C, D, E, F  
Unvisisted Nodes: G

| Origin Vertex - G | Shortest Distance | Vertex Visited Previously |
| --- | --- | --- |
| Node A | 0 |  |
| Node B | 2 | A |
| Node C | 6 | A |
| Node D | 7 | C |
| Node E | 10 | F |
| Node F | 9 | C |
| Node G | 12 | B |

Visited Nodes: A, B, C, D, E , G

Unvisisted Nodes: Null

**b. Calculate Closeness Centrality:**

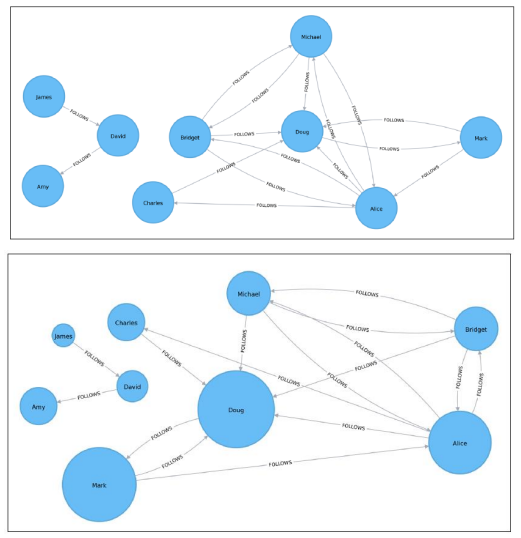
****

| Nodes | A | B | C | D | E |
| --- | --- | --- | --- | --- | --- |
| A | 0 | 1 | 2 | 3 | 4 |
| B | 1 | 0 | 1 | 2 | 3 |
| C | 2 | 1 | 0 | 1 | 2 |
| D | 3 | 2 | 1 | 0 | 1 |
| E | 4 | 3 | 2 | 1 | 0 |
| Total | 10 | 7 | 6 | 7 | 10 |
| n-1/Total | 0.4 | 0.57 | 0.66 | 0.57 | 0.4 |

The Centrality of Node C is 0.66 which is the highest of all nodes, hence node C is the closest to all nodes.

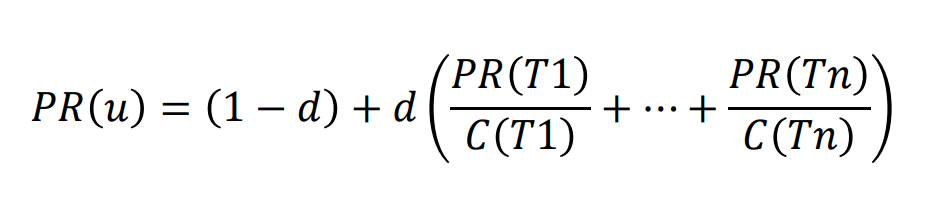
**Question 2.**

1. **Twitter Social Graph Algorithm Identification**



Pagerank is used in the above nodes, pagerank determines the strength of the node depends on influential nodes connected to it.

Formula :



Where, **PR** is the page rank, **PR(u)** is the pagerank of node, **d**  is the damping factor, and **C** is the total number of connections.

1. **Usecases**
   1. Page rank can be used in health care to study the effectiveness of protiens of genes in an DNA to study disease development and other advanced fields.
   2. Page Rank can be used to Identify Hostsopots for Crime in locality using Police Data.
   3. Disease Strentgh Prediction on How and where the disease spreads
2. **Network Types**

**Random Network** :



Random networks can be defined as networks which have independent edges, there are many different variations of the random network like the **scale free** , **SONET** and the **Small World**.

**Small World Network** :



It is a type of Random network in which it is easy to traverse to other nodes in the network with just few steps, as each node not being direct neghbours of each other but each neghbour of the desired node can be neghbours of each other, with increase in number of nodes the shortest path factor increases very slowly as compared to other networks.

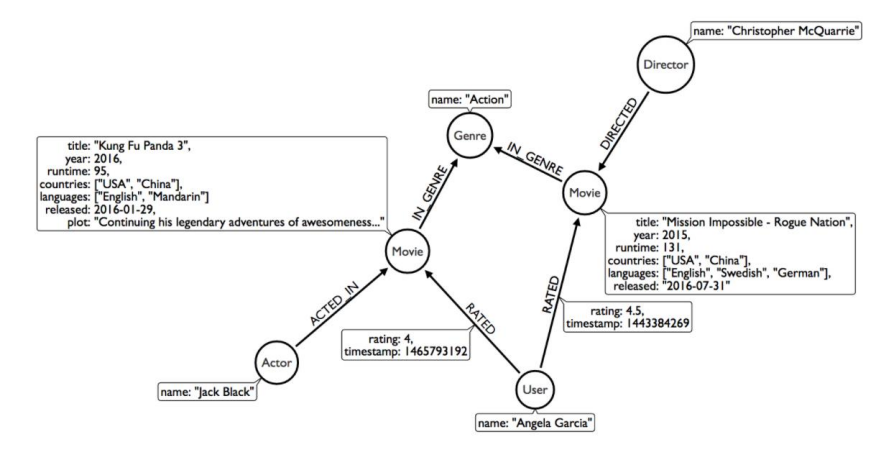
**Scale Free** :



This is a type of random network whose probability of distribution (degree distribution) obey power law.

**Question 3**

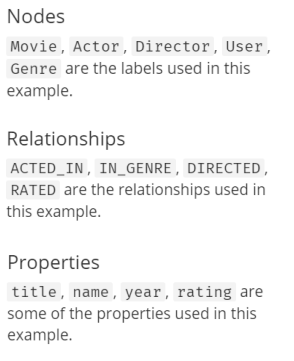
**Movie Lense**

****

**Recommendation on Movies Database**

Recommendation based on movies loded can be done in multiple ways, we will use cypher query’s and features provided by neo4j to extract desired results.

Our DB has following properties which we can work upon for further research, as our Cypher queries will largely depend on the structure of the DB.

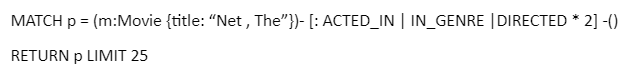


There are two ways in which we can filter movies for recommendation

1. **Content-Based Filtering**
2. **Collaborative Filtering**

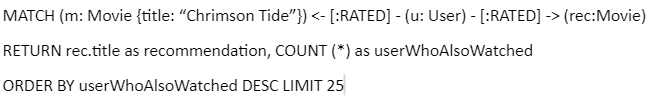
**Content Based Filtering:**

**“ Items Similar to what user is looking now”**



In the above Query we try to get the recommendations based on what user is currently watching, that can be done by considering movies with similar Actors, Genre, and Directors.

**“Customers who watched this also watched this”**

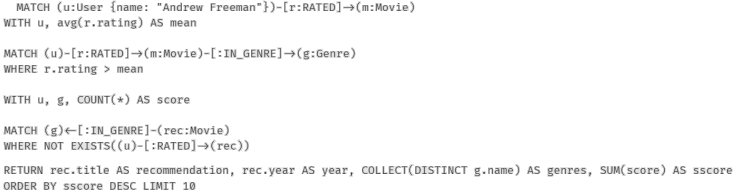


in the above query we are comparing based on RATED user to movies they would like to watch sharing similar traits.

**Collaborative Filtering:**

In this method we can use usernames to recommend movies to user, this is more personalised approach as it is different for every user.

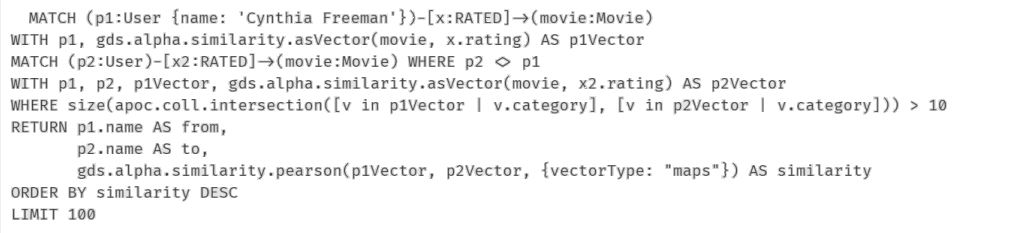
**Using Genere**



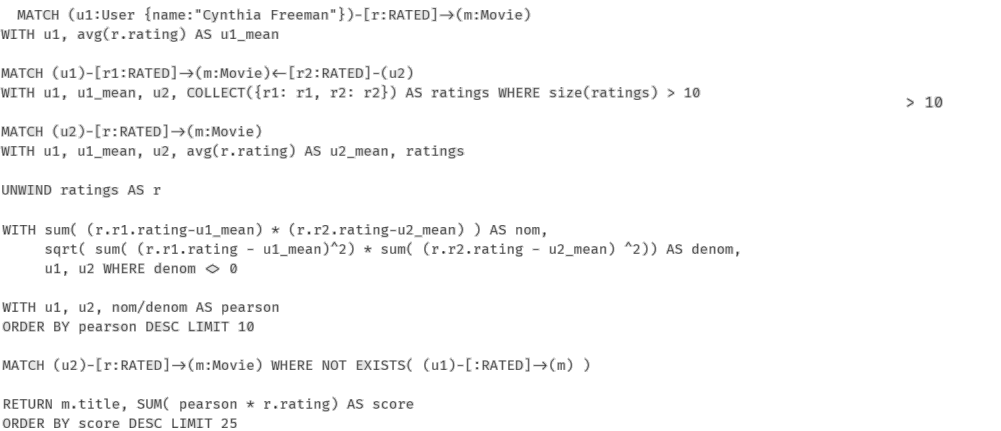
In the above query we used User and the Genere of movies watched by user to provide personalised recommendations, the query considers rating, in genre and has overall dependency on preferences of a particular user.

**Based on Similarity Metrics**

we can use similarity scores of nodes to get the strength of recommendations from the nodes



Neo4j provides us with implementation of Person Similarity algorithm we can extend this to get recommendation after combining it with KNN to recommend movies.



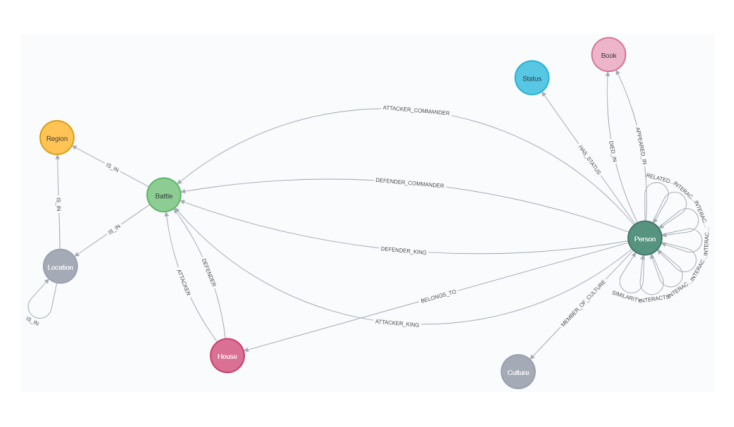
The Query above is modeled based on KNN -Movie recommendation using Pearson Similarity Metrics that will provide us with personalised recommendation for movies

**Compare Lable Propagation and Louvian Graph**

| **Louvian** | **Label Propagation** |
| --- | --- |
| Makes Optimal Choices Locally hence a Greedy algorithm | Iterative Algorithm which changes choices depending of closeness of the node |
| Requires no direction to traverse through the nodes as it is completely dependent on heuristic evaluation of the nodes | Requires Directed Graph to Operate as it traverses trough relationships to eliminate labels and consolidate nodes. |
| Aggregation takes place when the nodes of same community is detected, hence new node creation takes place which has all the properties of aggregate nodes | Just labels are reduced hence there is no new node ceration and so there is no real querying for aggregation , instead just labels which are variables are assigned to the designated community |
| It is a little difficult to Interpret the result of Louvian | Its easy to interpret the results as the end result is just labels that are remaining after the consolidation. |
| Considers graph as undirected so the results will be consistant | Results might differ on large graph between directed and undirected graph |
| Slower than Label propagation as it uses local optimization to provide results | Faster because the result is just elimination of labels when consolidated as per similarity |

**Question 4.**

**Game of thrones Graph**

****

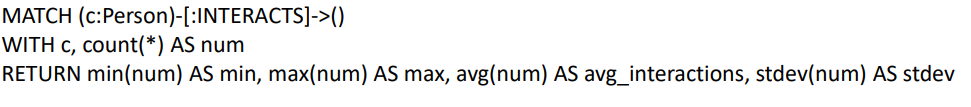
**Overview:**

The Graph Containes the characters of game of thrones as nodes(Person) , the relationship between the characters are defined on how frequently they interact with each other in book, which is considered as a span of 15 words.



**Interactions :**

To get the average number of interactions between the character we need to query the Graph DB as follows

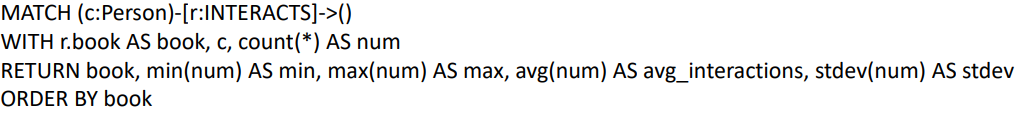
****

In the query above we have selected the person node and added our clause over it, we have selected **Person** nodes as per INTERACTS relationship to query out minimum, maximum, average, and standard deviation of interactions overall of Game Of Thrones DB.

**Interaction as per Book**

We can also calculate interaction per book by adding book to the clause which will separate the number of interactions as per book and provide us the average interactions per book.

To get the average number of interactions per book we need to execute the query below



In the query above we can see that we have added clause of book instead of directly selecting everything, the book property of the graph will help us separate the INTERACTS as per book and we can fetch the minimum, maximum, average interactions per book, and standard deviation.

**Applying Graph Theories:**

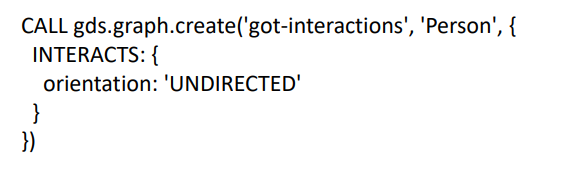
Before we start using “Graph Data Science” to analyse the data on the neo4j,we will be creating named graphs to ease the process of our implementaion.

**Creating Named Graphs :**

Named Graphs are created in a way to execute graph which will be faster and more easy to repeat, when we create a named graph we predefine the layout of our Nodes and Relationships we need to explore, we can create multiple named graphs , each graphs are represented as a projection in the neo4j environment and hence it is cached. so it is necessary to delete the graph when we are done with it, as it will free up space.

In addition it helps us play with the nature of the graph elements by Sub-graphing, where the small graphs are part of the large graphs and we can use it feely. editing or merging types of relationships or node labels, changing direction of relationship, collecting parallel relationships and properties, extracting relationships from larger patterns.

eg.



**Using GDS Library to explore Importance of Charaters**

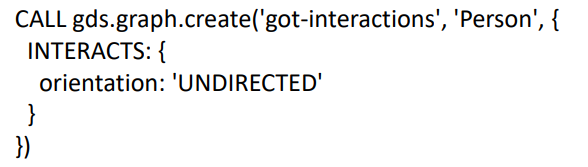
We will be using Graph Data Science Library to explore importance of the characters

**Page Rank for Importance of Characters:**

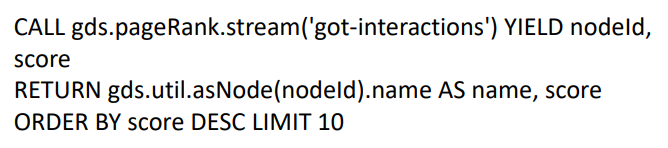
Pagerank is used to measure influence of node using connectivity of the direct nodes connected to the target node, it is an iterative algorithm and it provides a score that can be used to determine how well connected a particular node is.

We will implement the pagerank algorithm below in following steps.

1. Creating a Named Graph



2 . Applying GDS Page Rank Algo using the created Named Graph

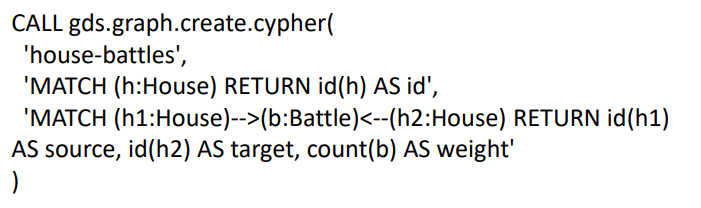


The above implementation above code should provide us with the top 10 influential characters ordered by descending.

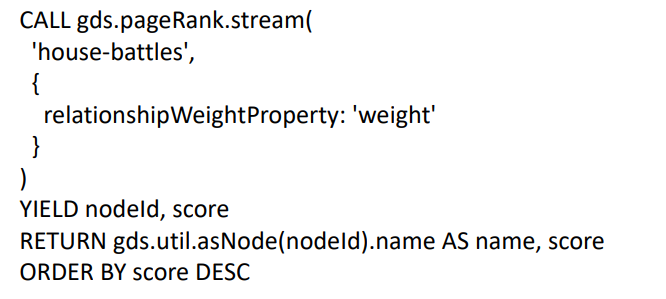
**Exploring Themes**

We can use Page Rank to further navigate into the story of Game of Thrones by exploring themes other themes.

1. **Page Rank for Battles fought together**
   1. **Create a named graph**

****

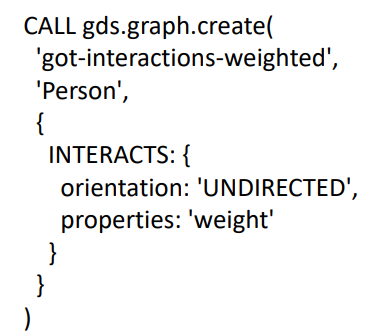
* 1. **Querying the Battles fought together**

****

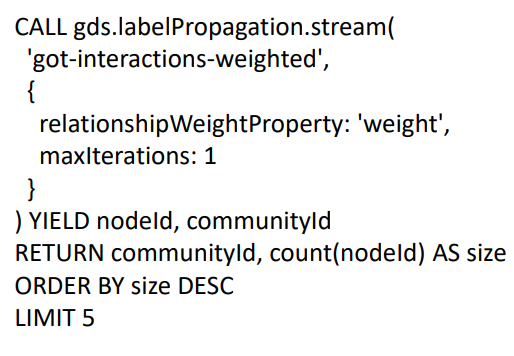
1. **Community Detection using Label Propagation**

Label Propagation is also an iterative algorithm, label propagation assignes a unique label to common nodes in the graph using its connected negbours, then using convergence of node it renames the labels to fit in close fir communities.

1. **Creating a Named Graph for Label Propagation Community Detection**



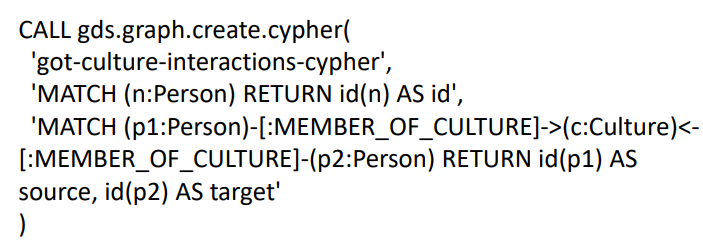
1. **Querying the GDS algorithm**



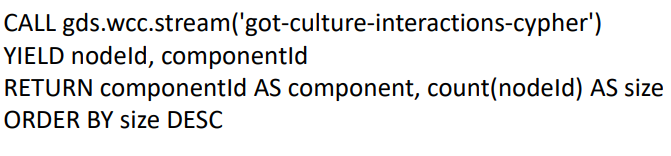
**3. Culture detection using Weakly Connected Componets(Union Find)**

Weakly connected Components uses nodes but not direction of relationships so the entire graph is considered as directionless, which helps us detect the disconnected subgraphs.

1. **Creating Named Graph for Weakly Connected Componets**

****

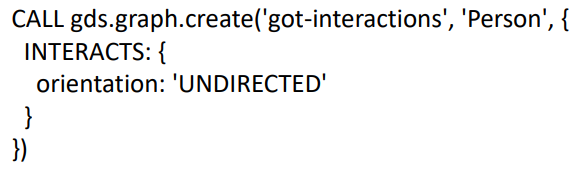
1. **Query WCC using GDS library**

****

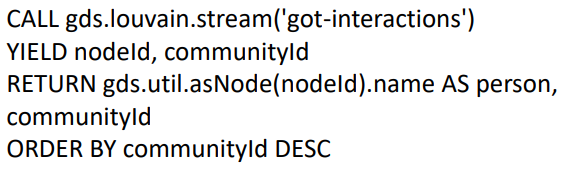
**4. Using Louvian algorithm to detect hierarchy of community**

Louvian is also a community detection algorithm which uses heuristic evaluation to detect community, and we can get scales of community by hierarchy, it is a greedy algorithm.

1. **Named Graph for Louvian**

****

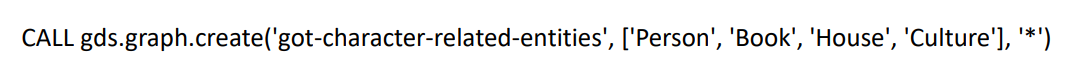
1. **Using GDS to Query with Louvian**

****

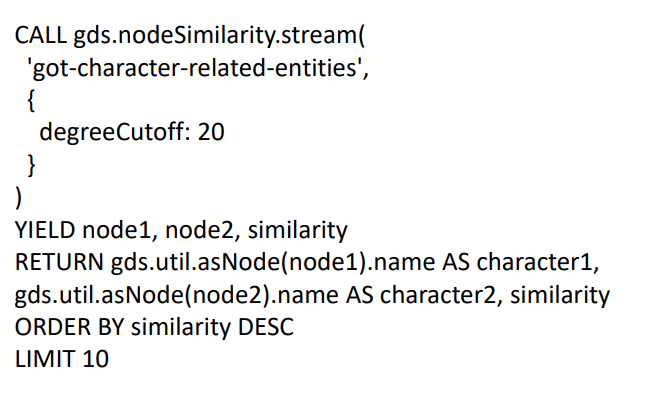
**6. Finding Relationship between nodes using Node Similarity**

Nodes which have connections which determine the negbours and sharing relationships between nodes Node Similarity uses the connection between nodes and shared negbours to define similarity between them.

1. **Named Graph for Node Similarity**

****

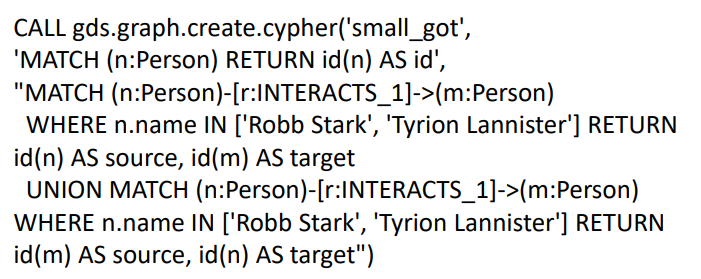
1. **Query**

****

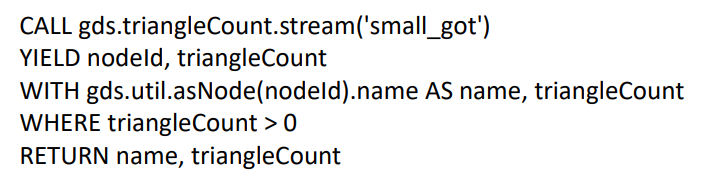
**7. Triangle Count**

Triangle count is used to identify set 3 nodes that are connected to each other, it can be used only in undirected graphs.

**a. Named Graph for Iteraction between “Robb Stark” and “Tyrion Lannister”**

****

**b. Query**

****