### **Deliverable I**

### **Project Proposal (Updated)**

**Dataset** - Goodreads dataset (<https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home>)

The dataset contains - (1) meta-data of the books, (2) user-book interactions (users' public shelves) and (3) users' detailed book reviews. These datasets can be merged together by matching book/user/review ids. The entire dataset contains -

* 2,360,655 books (1,521,962 works, 400,390 book series, 829,529 authors);
* 876,145 users;
* 228,648,342 user-book interactions in users' shelves (include 112,131,203 reads and 104,551,549 ratings).

For the purpose of this project, I propose to filter the dataset by genre of book pertaining to ‘Poetry’, as the entire original dataset is too huge.

**Business use case**

Goodreads is like a social network platform like Facebook, but it is for books. Here, users review books, become friends, share books and keep status on their reading habits etc. Goodreads derives its revenues by [promoting book campaigns](http://www.goodreads.com/advertisers), where it [works with major publishers to promote titles](http://elitzr.com/elitzr-14-goodreads-otis-chandler/). The company’s data services mainly center on [an open API](http://www.goodreads.com/api) for utilizing its book data on third-party sites. Data from Goodreads could be utilized to understand how book characteristics, writers and other factors perform among certain reader sets. It would be interesting to apply the machine learning concepts to perform this task. But, graph networks draw more insights from the data that are hidden to regular machine learning techniques.

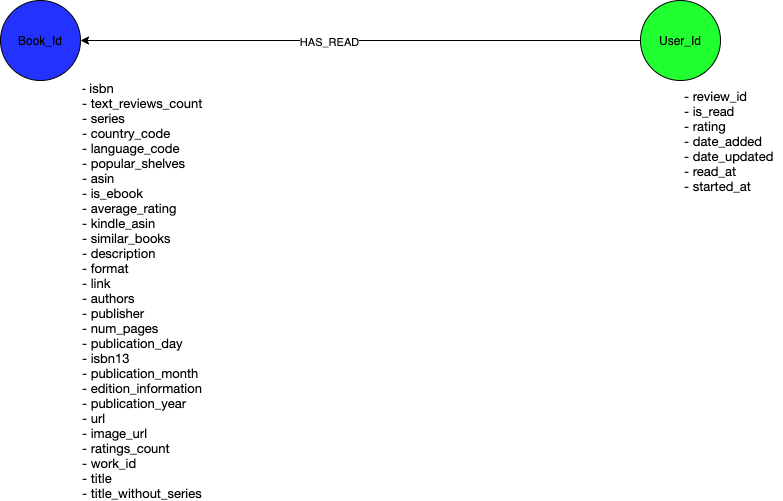
Analytics team of the book-based social network platform Goodreads could use graph analytics and build implicit book-user graphs:

* to visualize and analyze networks of interactions between users, books, and authors via their reading preferences [graph visualization]
* to identify and recommend books to users by engineering network features
* to identify influential users and user groups (communities) on the Goodreads social network

Essentially, the idea is to work on designing a network based recommender system for the Goodreads platform. Furthermore, to build a graph database and present results of the above use cases via visualization to the stakeholders.

Knowing that Goodreads generates its revenue by running Book promotion campaigns, running ads on books, it's important that it understands its users and their preferences well. Through the business use cases described above Goodreads can better understand its users and use it to build a rewarding relationship with the book companies to generate profit.

### **Graph Data Model (Version 1.0)**

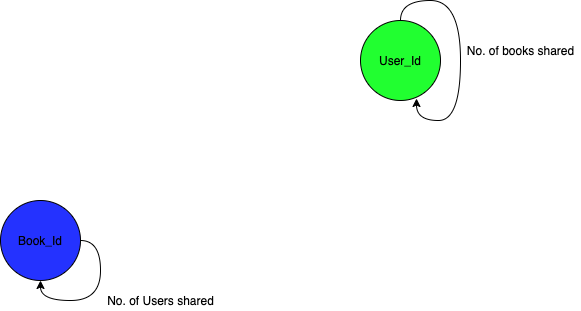


### **Graph Projections**

The two mono-partite graphs are as below.

For the first, User\_Id - two users are said to be connected if they have read (shared) the same books. Number of shares becomes the weight here.

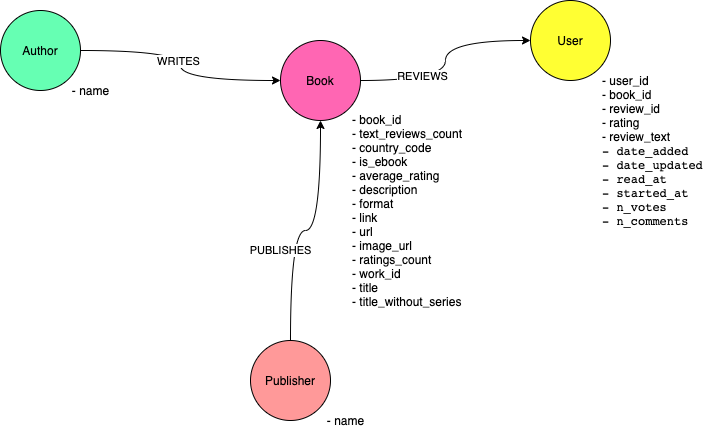
Second, Book\_Id - two books are said to be connected if they have the same users and the number of users that read the book becomes the weight. Say if user 1 and user user 2 have read that book, the weight becomes 2 here.



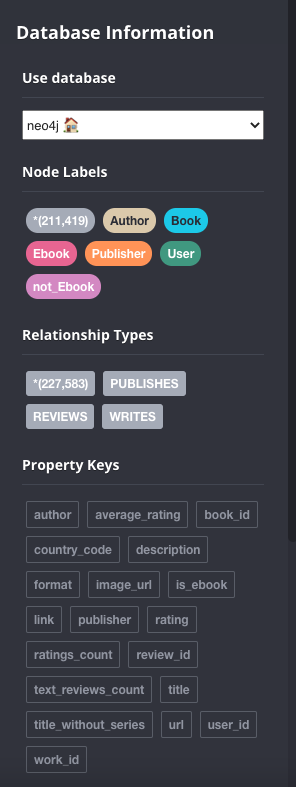
### 

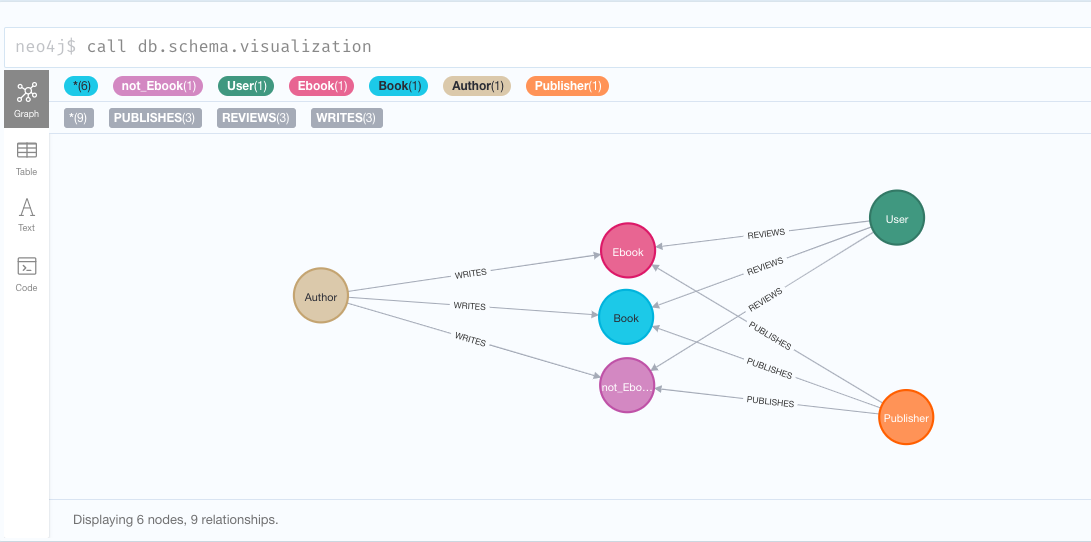
### **Graph Data Model (Updated)**

Having just 2 node types hinders full understanding of users behaviour. So in order to expand the scope of understanding of user characteristics, few other node types have been added.



**Graph Model in Neo4j**

****

****

**CYPHER Queries**

1. **Find the books that has the maximum number of reviews**

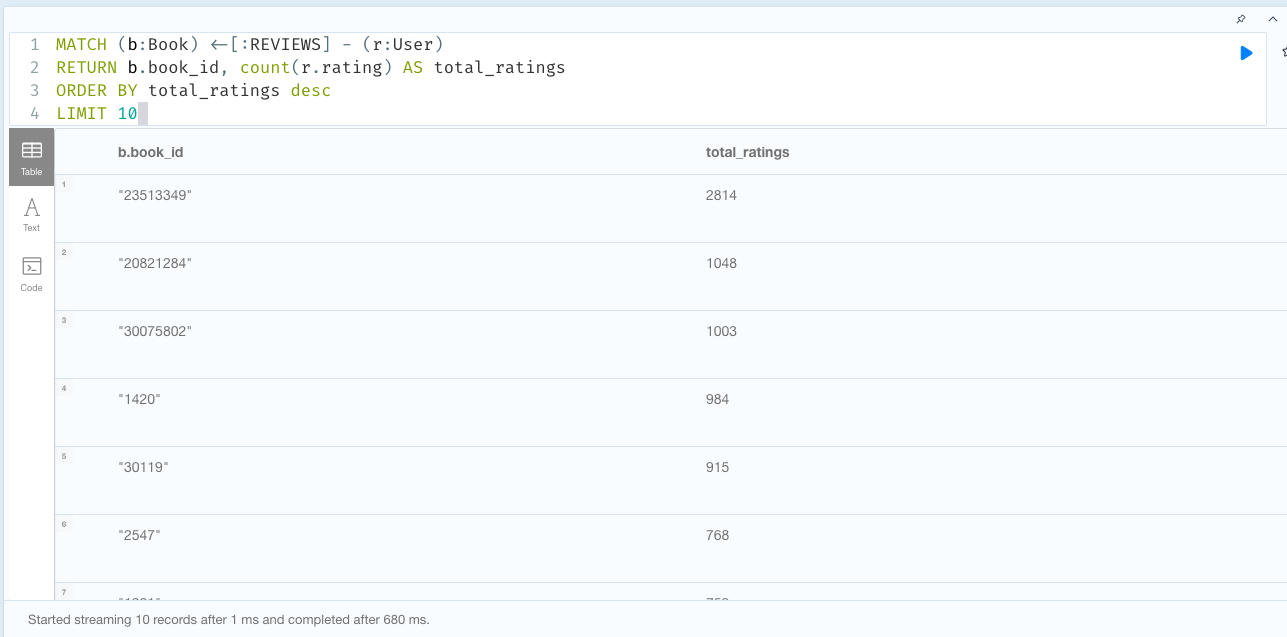
MATCH (b:Book) <-[:REVIEWS] - (r:User)

RETURN b.book\_id, count(r.rating) AS total\_ratings

ORDER BY total\_ratings desc

LIMIT 10

Popular books tend to receive more reviews. Whether it is popular for negative or positive reasons, a popular book receives more reviews. It is important to know the top few popular books to understand user characteristics related to those popular books.



1. **Find the top 10 users who have read the most number of books of different authors.**

MATCH (p1:User)-[:REVIEWS]->(b:Book)<-[:WRITES]-(a:Author)

RETURN p1.user\_id AS user, count(distinct a.author) AS authors\_read

ORDER BY authors\_read DESC

LIMIT 10

Users who have read more books are important when it comes to finding the relationship between users' preference to authors. Users who have read different authors could also be related to other users through the books. So, understanding the relationship between users to users through books will help Goodreads better tune their recommendations.



1. **Find the top 10 users connected to users through books.**

MATCH (p1:User)-[:REVIEWS]->(b:Book)<-[:REVIEWS]-(p2:User)

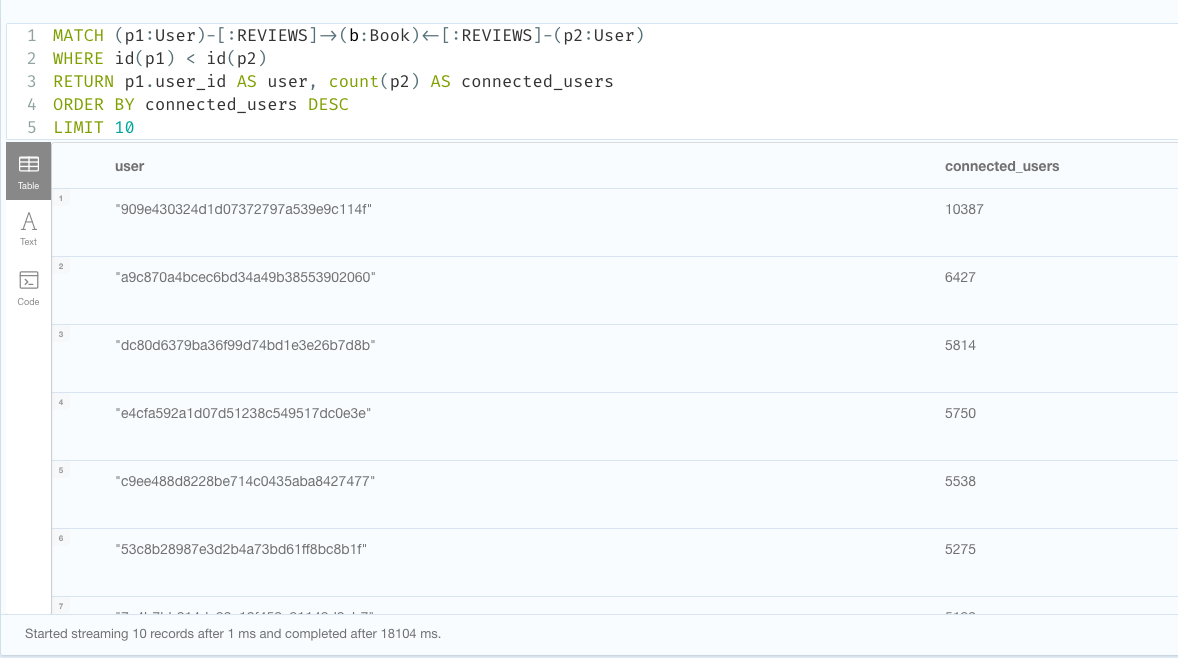
WHERE id(p1) < id(p2)

RETURN p1.user\_id AS user, count(p2) AS connected\_users

ORDER BY connected\_users DESC

LIMIT 10

Users with maximum connection to other users through books tend to be the influencers. So targeting these users through campaigns could yield better results to the campaign. Knowing these top few user nodes that are maximum connected to other users is important to the use case to find influencers.



**Projections and Algorithms**

Before jumping to identify influencers in the graph, it is necessary to first define and lay out the assumptions.

So, starting with a definition for influencer -

*A Social Media Influencer is a user on social media who has established credibility in a specific industry. A social media influencer has access to a large audience and can persuade others by virtue of their authenticity and reach.*

Next step would be to create a projection such that it captures the relationship between users based on the book they have read. So, user 1 is said to be related to user 2 if they both have read the same book.

As this dataset does not provide any information on friendship or follower-followee relationship between users, for the sake of convenience it is assumed that people who have reviewed the same book are friends.

Below is the cypher query to create the projection discussed.

**Projection 1 -**

CALL gds.graph.create.cypher(

'user-to-user-relationship',

'MATCH (p:User) RETURN id(p) AS id',

'MATCH (p1:User)-[:REVIEWS]->(b:Book)<-[:REVIEWS]-(p2:User) WHERE id(p1) < id(p2) RETURN id(p1) AS source, id(p2) AS target')

**Algorithm 1 - PageRank**

From the projection, the idea is to find people who are influencers. But, to identify influencers, what makes a user influential? One can say -

* A book is relevant for him/her if other similar people liked it.
* A person is similar to him/her if they like books that are relevant to him/her.

So to find these influencers, centrality algorithms can be used.

But, if there are two people who are equally similar to a given user and say if both the people like the book this user likes. But if one of these people likes every single book in the market, and the other one likes only a handful of books, the set of books from the second person is much more informative about the first person’s interests. Which is why it is more important to use Pagerank instead of other centrality approaches, as the [PageRank algorithm](https://neo4j.com/docs/graph-algorithms/current/algorithms/page-rank/) measures the transitive influence or connectivity of nodes.

If Goodreads wants to post some content or do some marketing campaign on books and have it spread across a large number of users/accounts, the users that rank highest for PageRank would be the best place to post that content.

**Cypher query-**

**CALL gds.pageRank.stream('user-to-user-relationship')**

**YIELD nodeId, score**

**RETURN gds.util.asNode(nodeId).user\_id AS user, score**

**ORDER BY score DESC, user ASC**

****

**Algorithm 2 - Betweenness Centrality**

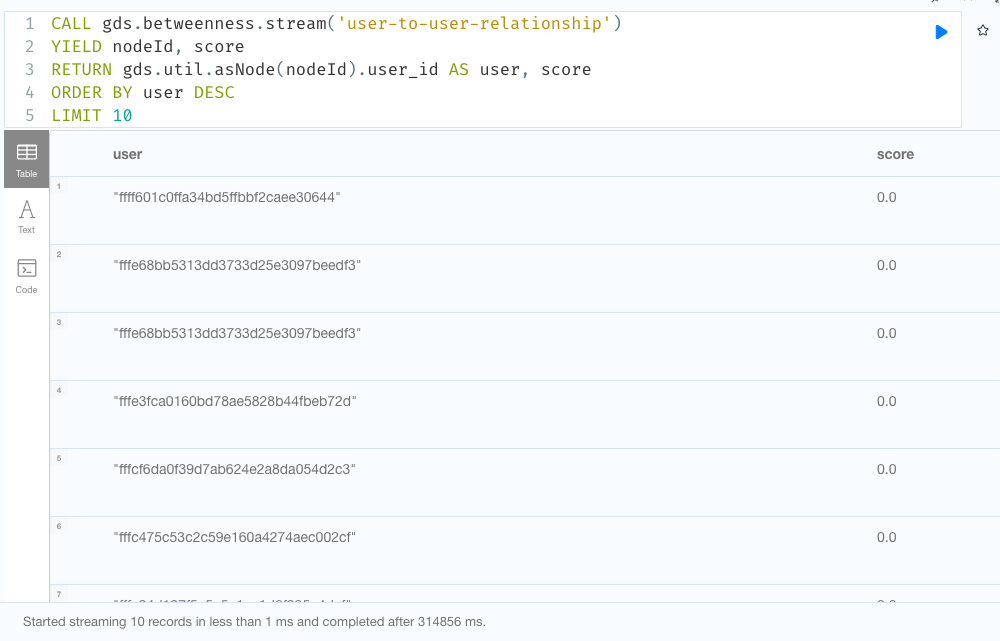
PageRank can give us influencers. But, there can be a disjoint set of groups that have users with disjoint interests, connected by very few users that connect groups. These users are important to know of to maximise the spread of book campaign success. Such users can be found using betweenness centrality as betweenness centrality is a way of detecting the amount of influence a node has over the flow of information in a graph. So this algorithm can be used to find nodes that serve as a bridge from one part of a graph to another.

**CALL gds.betweenness.stream('user-to-user-relationship')**

**YIELD nodeId, score**

**RETURN gds.util.asNode(nodeId).user\_id AS user, score**

**ORDER BY user ASC**

****

**Algorithm 3 - Louvain Modularity**

How are the users similar? Are there any groups in between users? Answers to such questions might be important to recommend books or to make a book launch/marketing campaign successful. Users in the same community might react the same way like the other user in the group towards a book. To know and capture this user's reaction towards the book would be necessary and reasonable before launching a book campaign as part of market research.

To do this, community detection algorithms can be used to evaluate how groups of nodes (users) are clustered or partitioned, as well as their tendency to strengthen or break apart. The Louvain Modularity algorithm detects communities in networks, based on maximising a modularity score, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network.

This algorithm can be used to find sub communities in the larger graph community.

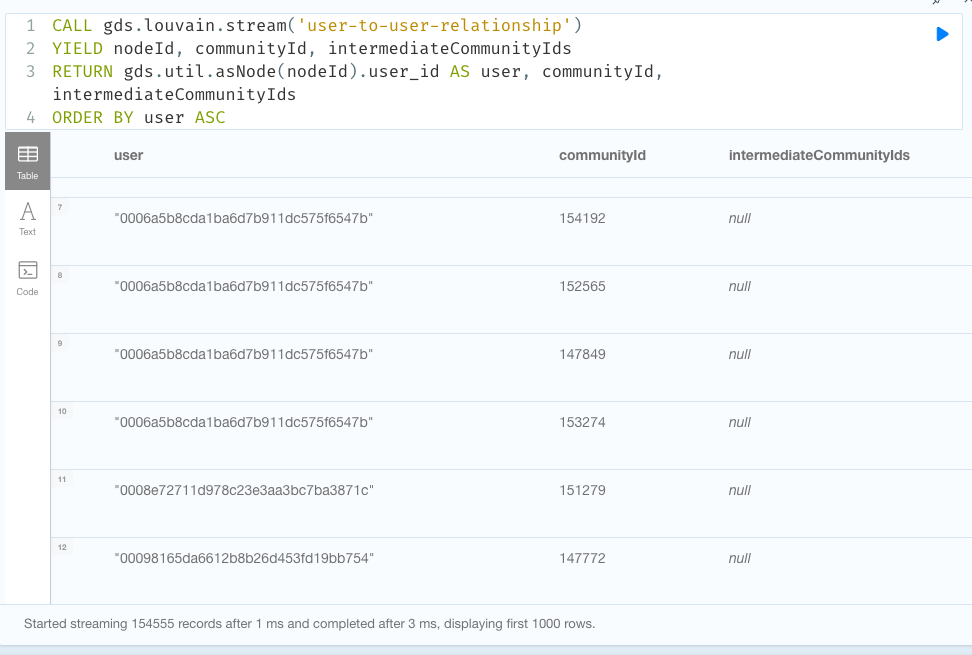
**CYPHER query -**

**CALL gds.louvain.stream('user-to-user-relationship')**

**YIELD nodeId, communityId, intermediateCommunityIds**

**RETURN gds.util.asNode(nodeId).user\_id AS user, communityId, intermediateCommunityIds**

**ORDER BY user ASC**

****

**Projection 2 -**

Projection 1 connects users to users folding on the edge- book. Similar projection can be done by folding on Authors. It is generally true in the book readers community that readers are divided by authors they follow and fans/followers of the same author are so closely knit. To understand the influencers and their characteristics based on Authors, in order to make recommendations based on authors, we can use this projection.

CALL gds.graph.create.cypher(

'user-to-author-relationship',

'MATCH (p:User) RETURN id(p) AS id',

'MATCH (p1:User)-[:REVIEWS]->(b1:Book)<-[:WRITES]-(a:Author)-[:WRITES]->(b2:Book)<-[:REVIEWS]-(p2:User) WHERE id(p1) < id(p2) RETURN id(p1) AS source , id(p2) AS target')

**Algorithm 4 - Closeness Centrality**

Closeness centrality is a way of detecting nodes that are able to spread information very efficiently through a graph. The closeness centrality of a node measures its average farness (inverse distance) to all other nodes. Nodes with a high closeness score have the shortest distances to all other nodes.

So this algorithm can be used to find users that have a high closeness centrality score to influence the fan followers of that author. It would come handy when the book campaigns or book releases are planned. During book release the influencers with high scores from this algorithm can be specifically identified and used to propagate the information within the community. As the communities are tightly knit and well trusted among its followers, it would serve as a good tool to reach each and every member of the community.

**CYPHER query**

CALL gds.alpha.closeness.stream('user-to-author-relationship')

YIELD nodeId, centrality

RETURN gds.util.asNode(nodeId).user\_id AS user, centrality

ORDER BY user DESC

LIMIT 10

