18/6/2020

Online eBook House

SRH University of Applied Science Heidelberg

Submitted to:

Prof. Dr. Barbara Spick

Prof. Frank Hefter

Submitted by:

Kishansinh Rathod	(11013344)
Silky Sonali	(11013698)
Jhesta Narasimhamurthy	(11013658)
Shamanth Ravindra	(11013363)
Sharada Tumkur Vasudeva	(11013329)

Index

Index.	Topic	Pg.No
1	**Introduction**	**4**
2	**Organization** **2.1 Task and Responsibility** **2.2 Meetings** **2.3 Project Organization**	**5 - 7**
3	**Detailed Use-cases**	**8 - 14**
4	**Database**	**14**
5	**Data Model** **5.1 Mapping of requirements to data models** **5.2 Graph Data** **5.3 Document Data Model** **5.4 Key-Value Model**	**16 – 30**
6	**Implementation** **6.1 DATA MODEL IMPLEMENTATION**	**31 - 33**
7	**Queries**	**34 - 48**
8	**Evaluation**	**48**
9	**Bibliography**	**49**

Tables

Index.	Topic	Pg.No
1	Roles and Responsibility	5
2	Minuets of Meetings	6
3	Document data in MongoDB	17
4	Relation in Neo4j	18
5	Dataset in Redis	18

Figures :

Name	Pg.No
Fig 1-3 Sequence Diagram	10-12
Fig. 4-6 : Usecase Diagram	16
Fig 7 : database Schema in Neo4j	19
Fig 8 : Detail database Schema in Neo4j	20
Fig 9-16 : Sample Node	20-22
Fig 17-27 : Neo4j Sample Relation	23-25
Fig 28-37 : MongoDB Sample JSON	26-29
Fig 40 : Alternative data model	30
Fig 41-43 : Sample JSON MongoDB	32-33

1. Introduction

The world is becoming digital these days. Tablets and laptops are replacing pen and paper. With this digitization, people are also changing their habits of reading and moving towards digital reading and learning. From the increasing user's demand in the field, we came up with an idea to design and develop an Online Reading Platform for users where they can find various genres of books in a soft copy format and read them anytime, anywhere. Moreover, this platform gives users a chance to become an author by publishing their book online on the same platform.

While talking about the system, system will have 4 different portals. One is for Reader, one is for Author, one is for Book Reviewer and one is for Admin. There will be two types of reader one is free membership holder, and another is premium membership holder. Based on the member reader can read books. Premium member has access to all eBooks while free member has limited choice. However, if they want to read some book with premium tag then they will have the option to buy it without subscribing for a membership.

Any reader or end user can become the author. Author can make a request to publish their book which can be viewed by the Reviewer. Based on the content, Reviewer will submit the report to admin whether it should be publishing or not. If report is positive, then new book and Author will be added.

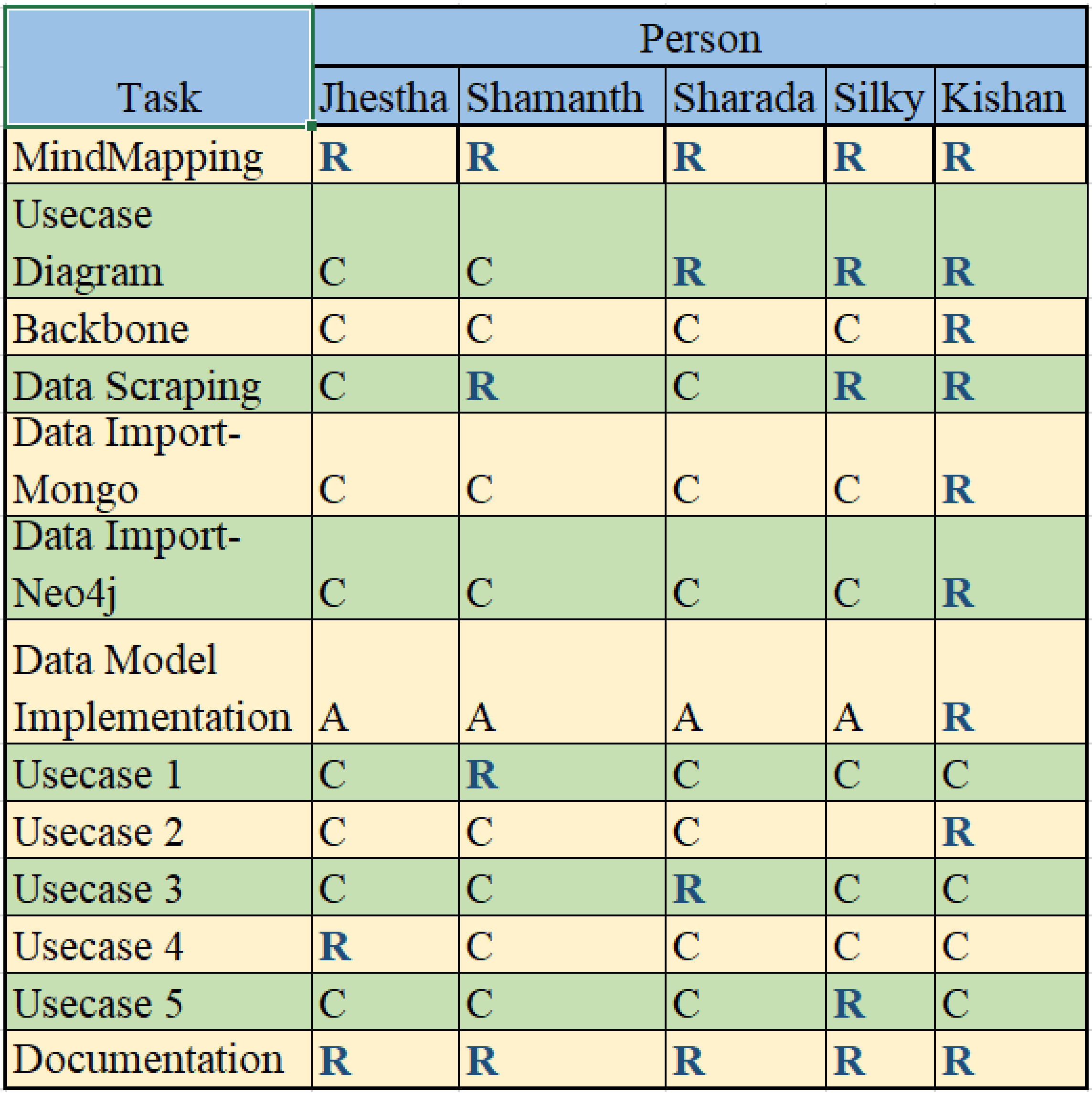
Coming to Admin part, admin will have all the access to this system. Admin can add, remove and update Reviewer, User, Author, Book, Publication House etc.

This platform has the following functionality - reading online or download offline, browse authors, browse books, browse category, publish their own book, trending by location, etc

2. Organization

2.1. Task and Responsibility

Tasks during the project was divided in such a way that every team member gets around one main dataset which they would have to search for, download them and then be responsible for their insertion and maintenance of the datasets. Also, every team member would work on 3 user stories each that would have difficulty level of easy, medium and hard. This would ensure that every member gets to research and build their knowledge through these user stories.



Task	Person				
	Jhestha	Shamanth	Sharada	Silky	Kishan
MindMapping	R	R	R	R	R
Usecase Diagram	C	C	R	R	R
Backbone	C	C	C	C	R
Data Scraping	C	R	C	R	R
Data Import-Mongo	C	C	C	C	R
Data Import-Neo4j	C	C	C	C	R
Data Model Implementation	A	A	A	A	R
Usecase 1	C	R	C	C	C
Usecase 2	C	C	C		R
Usecase 3	C	C	R	C	C
Usecase 4	R	C	C	C	C
Usecase 5	C	C	C	R	C
Documentation	R	R	R	R	R

Table 1: Roles and Responsibility

R= Responsible

A=Accountable

C= Consulted

2.2. Meetings

Meetings during the project happened after discussion of the availability of team members on messaging service “WhatsApp”. After confirmation from everyone regarding timing and place, team members would meet on Microsoft Teams or “WhatsApp”. Before ending the meeting, agenda for the next meeting was generally set to so that team members would know their deadlines and goals before the next meeting.

Date	Agenda	Decisions
15.05.2020	Project Title Idea	Discussion to come up with new ideas
18.05.2020	Project Title	Finalized two topics to discuss the scope
20.05.2020	Project Scope and finalizing topic	1. Discussed the scope of the two topics 2. Finalized online eBook Platform like kindle
22.05.2020	Basic data Model	Discussed how to create data model and what to use
23.05.2020	User stories	
25.05.2020	Use case	Decided to identify use cases
26.05.2020	Use case	All members came up with the use cases
27.05.2020	Use case	Again, discussed use cases and finalized them
29.05.2020	Discuss over the Use-case	Implement feedback and new Use-case
01.06.2020	New Use-Case	
03.06.2020	Use-Case Data Scrapping	Find a way to get dataset
04.06.2020	Backbone	Discuss to design data models for

		all database
06.06.2020	Data scrapping	Found some data and decide to go with manual data entry
08.06.2020	Use-case	See remaining thing in data model and discuss
10.06.2020	Formal meeting	
11.06.2020	Status Meeting	
12.06.2020	Wrapping up	Finalizing things.

Table 2 : Minuets of Meetings

2.3. Project Organization

Most of the tasks and project status discussions happened during project meetings where everyone would list out the status of their tasks and explain the work done till now. New tasks and ownerships were also decided in these meetings in presence of every team member

During development phase every team member used DEV repository to dump all their scripts and files they worked up.

For the task Scheduling at some point we also used **Azure Devops** which is very convenient to assign a task to every team member. By using this every member of team would be aware of their responsibility and deadline to finish it.

GitHub Final – Click Here

3. Use Cases

3.1. User Stories

3.1.1. Publish Book

Sharada has immense knowledge on Indian cooking from its various states and also knows recipes of spices and history of traditional dishes. She also has a hobby of writing and wants to publish a cookbook where people can have a look into the recipes and a little history about it. She desires to become a successful author.

This user story is about a reader who wants to register as an Author. The reader will register as an Author and make a request to publish their book by providing all necessary documents and then wait for the response from the Publisher. The Publisher reviews it and decides to publish (positive) or not (negative). If the decision is positive, the user is recognized as an Author on the system and their book will be published.

Use-case

Author: Kishan

Description: One system user wants to publish their own book. So, they will register and login as an Author (Profile verification done manually and not online till approval) and they submit their publication for the review. Reviewer will receive the request and will send approval or rejection report to Publisher. Publisher will receive the request and if approved then they add necessary information like ISBN number, book will be published, added to database, and will be available on portal as well as New authors is added or else Author gets rejection report
Users: Author, Publisher, Admin

Outcome: Reader get recognized as Author published their eBooks

Database: Neo4j, MongoDB

Dataflow: MongoDB is the database which store all the detailed information about books and Author and Publication House so it is to add new Author and Books andsame node will be created in Neo4j to use it in future.

3.1.2. Currently Reading, Rating and Comment

Jhesta read one book and she liked it very much. She wants to tell other readers also about the book and she is in search of something to do this. This user story is about a reader who wants to rate books and comment on them. The reader can do both. Comments and ratings will be stored but before this they must have to start reading book because without it, they can't do it. Whenever the reader starts reading a book, reading counter will increase by +1.

Use-case

Author: Kishan

Description: When reader starts reading a book, the book will be added to the reading list of the reader which can be accessible anytime and when they finish any book then it will be in finished books list. So, when the reader wants to rate any book or comment on then the book must be in the reader's Currently Reading List or in Finished List.

Users: Reader (Any)

Outcome: User gives Rating, Comments on Books, and Reading counter will be increased by +1.

Database: Neo4j, MongoDB

Dataflow: Neo4j has all the needed information of this use case and it is also storing the reading counter so only one database will be used to show the comments and rating and it will increase the performance while mongoDB is only using for reading count of book

3.1.3. Create group and share books within the group

Jhesta, Sharada, Silky, Shamanth and Kishan are friends who have great interest in reading books. Shamanth read a great book and he wants to share this book to his friends. He wants a platform where he can create a group of friends and share his books with them.

This user story is about a user, who wants to create a group by inviting his friends. Later the people in the group can share books with comments among themselves instead buying it every time newly.

Use cases:

Create group and share a book within the group

Author: Sharada

1. **User creating a group**

Description: A reader can create a group and invite their friends (limited to 5). Once the invitation is accepted, he/she will be added into the group.

Users: Reader (Group Admin: one who will create the group)

Outcome: Any Reader can create a group. And only a group admin can add the members into the group

Database: MongoDB

Dataflow: MongoDB is the database which store all the detailed information about the group created.

Data Flow Diagram:

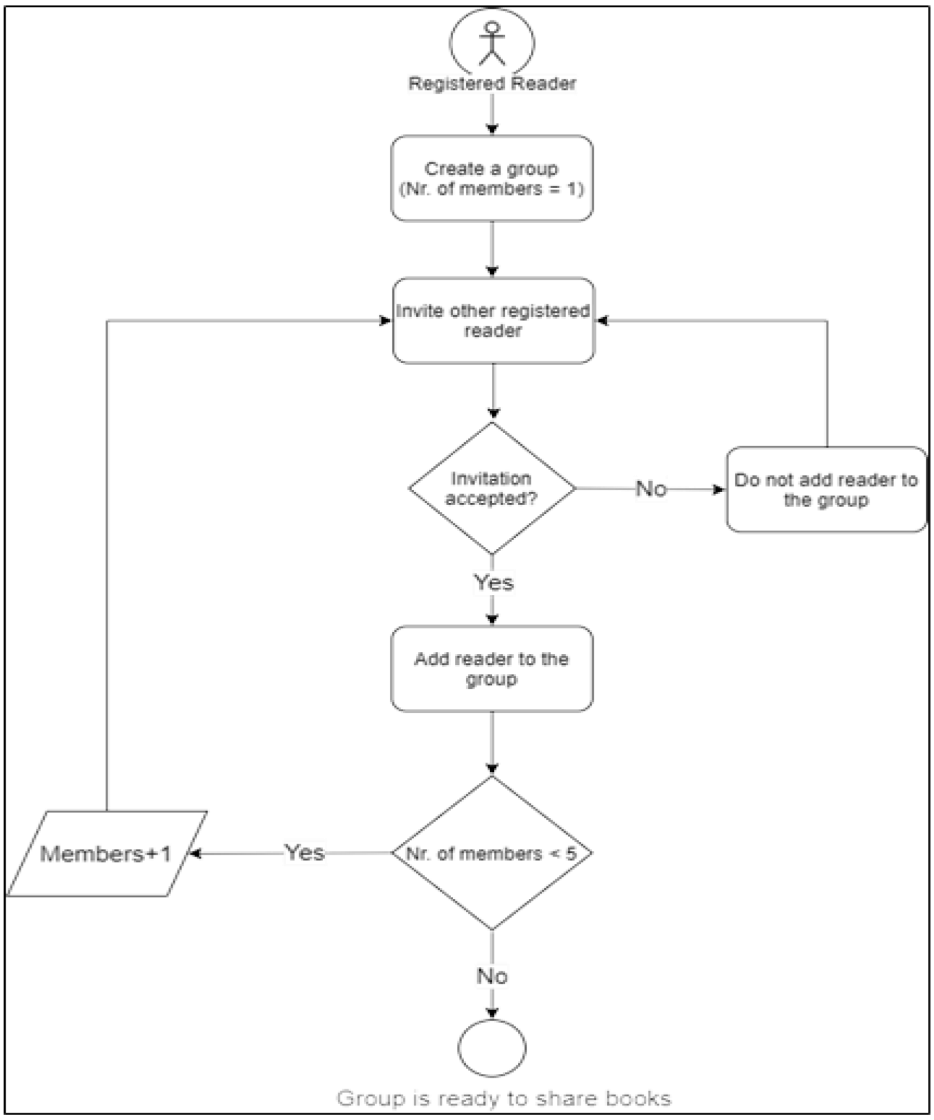

Registered Reader
Create a group (Nr. of members = 1)
Invite other registered reader
Invitation accepted?
No
Do not add reader to the group
Yes
Add reader to the group
Nr. of members < 5
Yes
Members+1
No
Group is ready to share books

Figure 1: Sequence diagram to create a group

2. User leaves the group

Description: A reader can leave group. Once he leaves the group, he will be removed from the group table, but the books shared by this user is still be available in this group.

Users: Reader (Group Member)

Outcome: This user will be no longer visible in the group and he/she cannot access the group.

Database: MongoDB

Dataflow: The user entry will be removed from the group document in the Database.

3. Admin leaves the group

Description: Group admin can leave the group at any point of time in this case, the second member of the group will be made as admin.

Users: Group Admin

Outcome: The admin will be no longer visible in the group and he/she cannot access the group. And the second member of the group will be made as admin.

Database: MongoDB

Dataflow: The user entry will be removed from the group table in the Database. The group document will be updated with new admin.

Data Flow Diagram

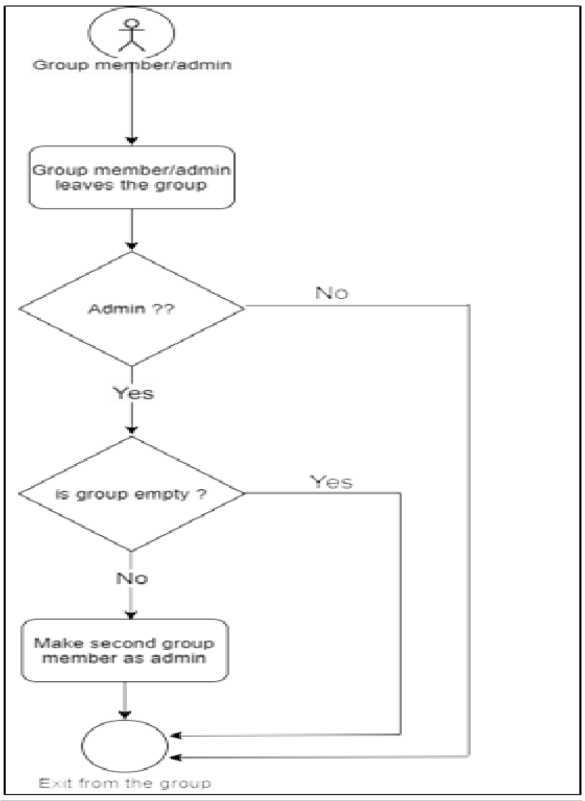


Figure 2: Sequence diagram for member/admin leaving the group

4. **Admin can delete the group**

   Description: Group admin can delete the group at any point of time and the group members will be notified. After this the group members cannot access the group.

   Users: Group Admin

   Outcome: The group will be deleted and none of the group members can access the group.

   Database: MongoDB

   Dataflow: The group document will be deleted.

   Data Flow diagram:

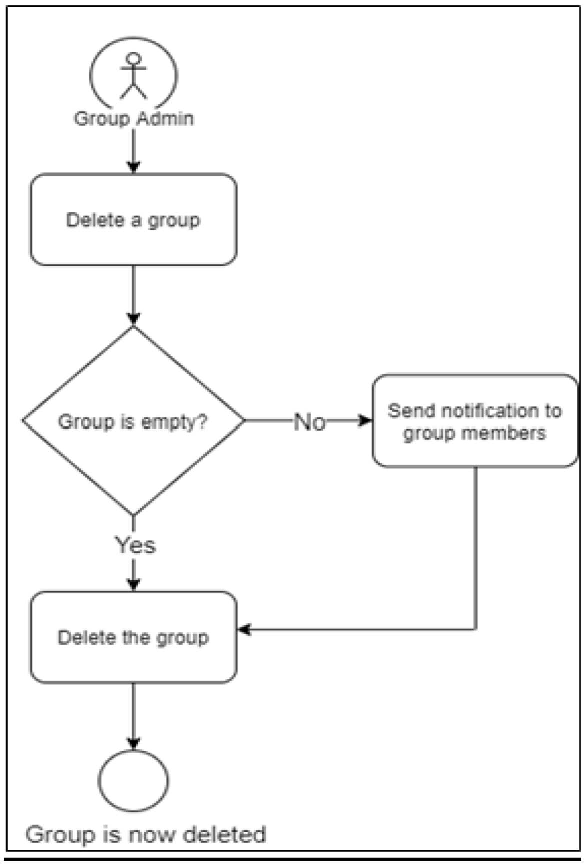


Figure 3: Sequence diagram for delete a group

5. **Share a book within the group**

   Author: Sharada

   Description: In a Group all members can share a book within the group. Date when the book shared will be stored.

   Users: Group Admin and Group Members

   Outcome: Any member of the group can share books and can give comments.

   Database: MongoDB

   Dataflow: MongoDB is the database which store the information of the books shared.

Use case Diagram:

3.1.4. Collect points on every book purchase

Kishan wants to have a discount on every purchase he makes in day to day life. Just like kishan, there are other users who are looking for various discounts, offers, sales and so on. Also, to attract more people and to make our eBook purchase graph upwards we have introduced a "points collection" system on every single euro the reader spends on the book purchase. So that Readers can redeem it later in their future purchases.

Use-case

Collect points on every book purchase

Author: Jhesta

Description: A reader gets points collected into their account on every purchase irrespective of free or premium reader. And these points are converted into euros, which can be redeemed on the next purchase.

Users: All registered readers

Outcome: Points gets collected on every purchase

Database: MongoDB

Dataflow: MongoDB is the database used to store the points collected by the user.

3.1.5. Affiliate Marketing

Silky loves to read but with reading she wants to earn some money by selling books and stuffs from home.

This user story is about a user who wants to join affiliate marketing where they have the chance to earn commission on every selling of a book to other users.

Use-case

Author: Silky

Description: User can refer other users to join this application with special referral code and on every successful referral joining and can earn marketing point. User can encourage other free account user to buy eBooks and on every successful purchase they will earn commission and marketing points based on their marketing club position.

Every user who are doing affiliate marketing are placed on marketing club which decide their commission rate. Marketing club are given based on marketing point, for example: Platinum is at top having highest commission rate followed by Gold, Silver and Bronze at last

Users: Any Reader

Outcome: Commission earned Marketing point increased

Database: MongoDB and Neo4j.

Dataflow: MongoDB to store all basic data and Neo4j to show who is joined by whose referral.

4. Database

We used 3 databases which are going to satisfy our user requirements and further coming optimized data model

1. MongoDB
2. Neo4j
3. Redis

MongoDB:

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is also having feature like Expressive querying, Flexible indexing against subsets of data, Updates in place, In Database aggregations and transformations.

Neo4j:

Neo4j is the open source graph database contains nodes and relationships and is the ACID compliant transactional database. Neo4j is agile, flexible and scalable graph database and with great performance than any rational and not rational databases. Traversal and built in algorithms for shortest path made Neo4j is the flavorsome choice for the relationships and location related data.

Redis:

Redis is the Remote Dictionary Server, it the in-memory data structure store and open source database with key and value pairs. As Redis is the in-memory database all the data stored in memory, but persistence can be achieved by two methods, which are RDB (Redis Data Base file) and AOF (Append-only File). With the nature of in-memory, Redis naturally perform well when it compared to other databases in terms of caching of huge data.

5. Data Model

Use-case Diagram

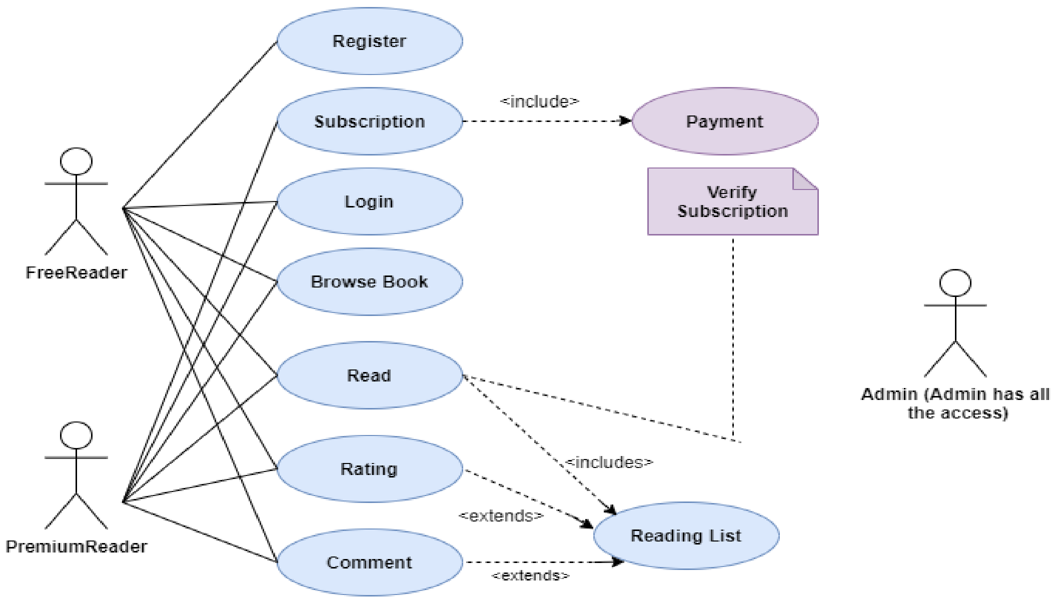


Figure 4: Use case diagram

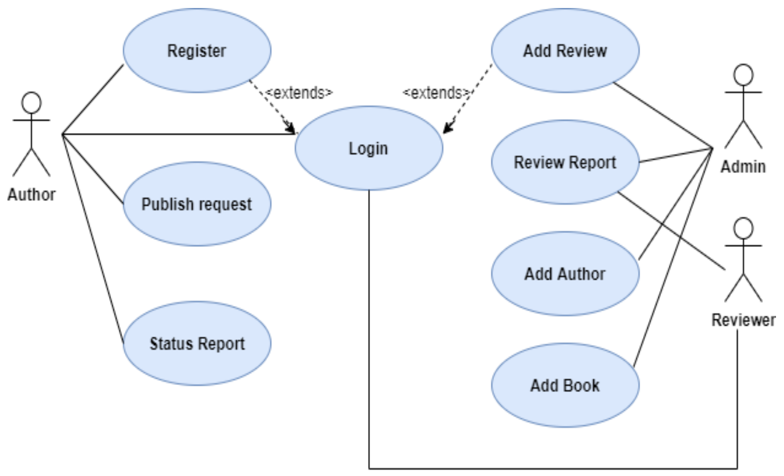


Figure 5: Use case diagram

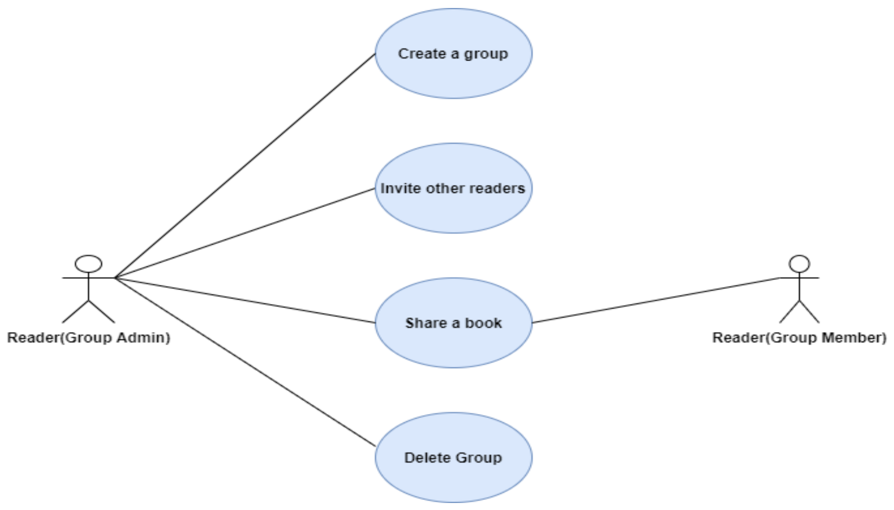


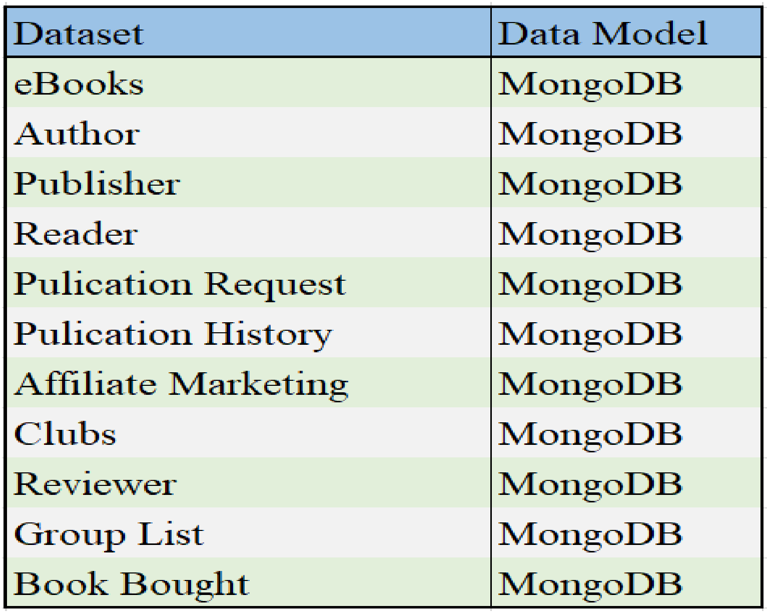
Figure 6: Use case diagram for "Create group" use case

5.1. Mapping of requirements to data models

We have 3 kinds of data that is needed for our project requirements and that should be fitted into the Data models, 1. Document data 2. Mappings/ Relationships between data 3. Key value data (Login Session, Search history and Historical data)

Document Model

Based on the requirements, the datasets have been downloaded or scraped from the web. The data in the dataset may or may not be homogeneous in terms of characteristics, thus may does not have all the fields in common. So, we thought of get the reference from scrapped data as well as from the web resources like Amazon Books, Google Books and ISBN data and do manual entry in excel file and made it to CVS so we can easily import into our Database system i.e MongoDB and Neo4j.



Dataset	Data Model
eBooks	MongoDB
Author	MongoDB
Publisher	MongoDB
Reader	MongoDB
Pulication Request	MongoDB
Pulication History	MongoDB
Affiliate Marketing	MongoDB
Clubs	MongoDB
Reviewer	MongoDB
Group List	MongoDB
Book Bought	MongoDB

Table3: Document data in MongoDB

Advantages:

- Performance Scalable
- Easy retrieval of information
- Information available at one place in document format

Disadvantage:

- Most of dataset would be independent from each other
- Increase complexity while trying to give relation between dataset

Mapping/Relationships:

We have relationships between the datasets such a eBook to Author, eBook to Publisher, Reader to eBook, eBook to Category, eBook to Language, Reader to Club etc. for these relation we have chosen Neo4j graph database to store. As Neo4J is very quick in traversal either way, getting the high-level filtration search results would be very promising for example: To find User reading which book or Author Wrote which book is very convenience in Neo4j with faster response.

Relationship	Datamodel
Reader to eBook	Neo4j
Reader to Marketing Club	Neo4j
Author to eBook	Neo4j
Publisher to eBook	Neo4j
eBook to Language	Neo4j
eBook to Category	Neo4j
GroupName to SharedBook	Neo4j
SharedBook to Comment	Neo4j

Table 4 : Relation in Neo4j

Advantages:

- Filtration of data is easy for complex level search
- Performance optimized
- Availability of difference algorithm

Disadvantages:

- For complete information of search result, we need to traverse to another database
- Query complexity is little more compared to other database systems

Key Value Data:

We have 2 kind of data as shown in the below table, which is needed for our further

Dataset	Data Model
Login Session information	Redis
Search keyword	Redis

Table 5: Dataset in Redis

So many users made login many times a day and it generated high volume of data. Apart from this session is encrypted and need to be check many times to perform Session Validation. For this we need faster database system to improve performance Here it comes the Redis database, which is in-memory, stores key value pairs and lightning fast in retrieval of data for further analysis.

Search history will be generated on minute and second basis in a day. Thus, it will generate huge volume of data with high velocity contains date, user and search values. To store huge data and further reading of this data for the analysis will need faster response database.

5.2. Graph Data

The below graph explains the overall structure of the nodes and their relationships with each other using few examples

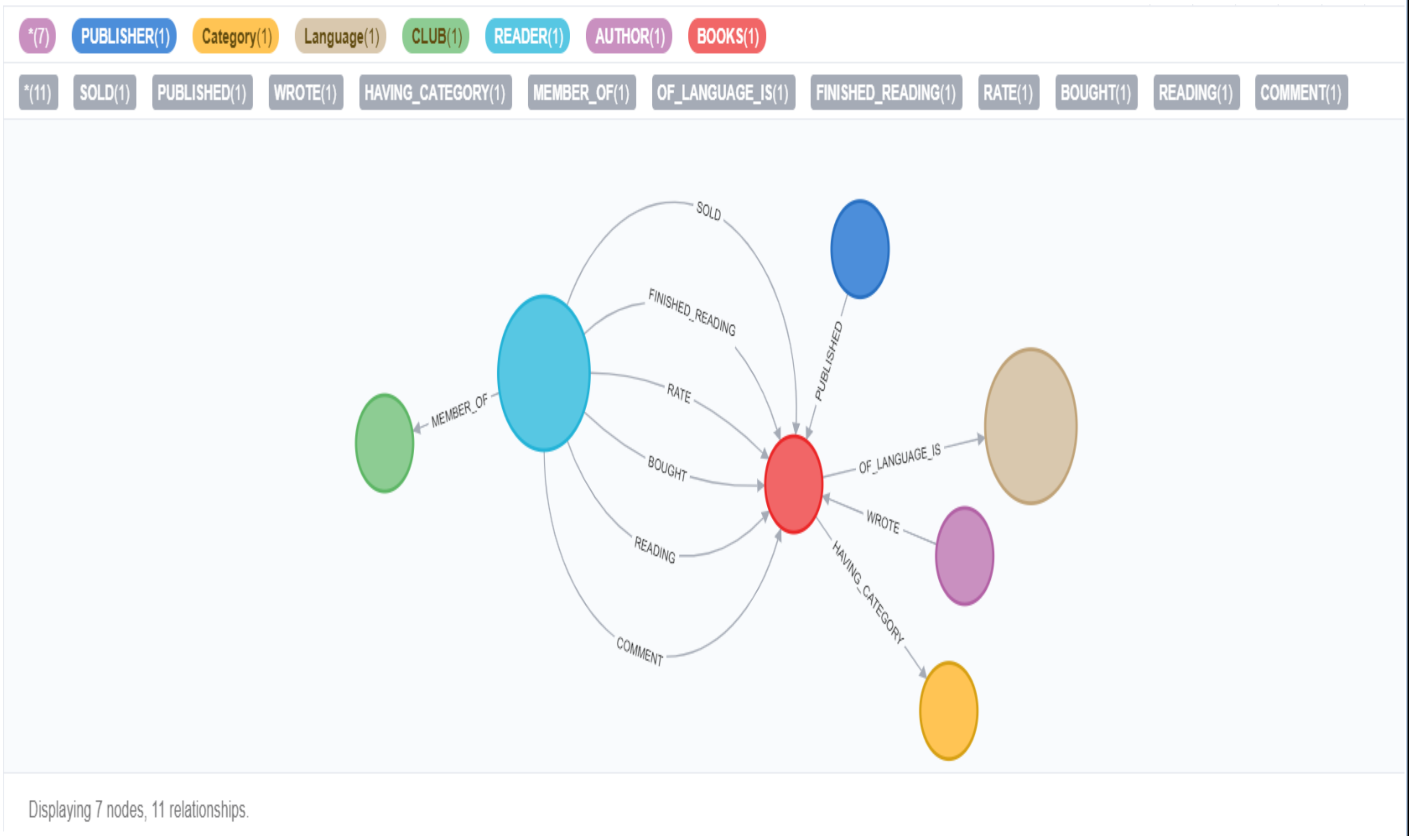


Figure 7: Database Schema of Neo4j

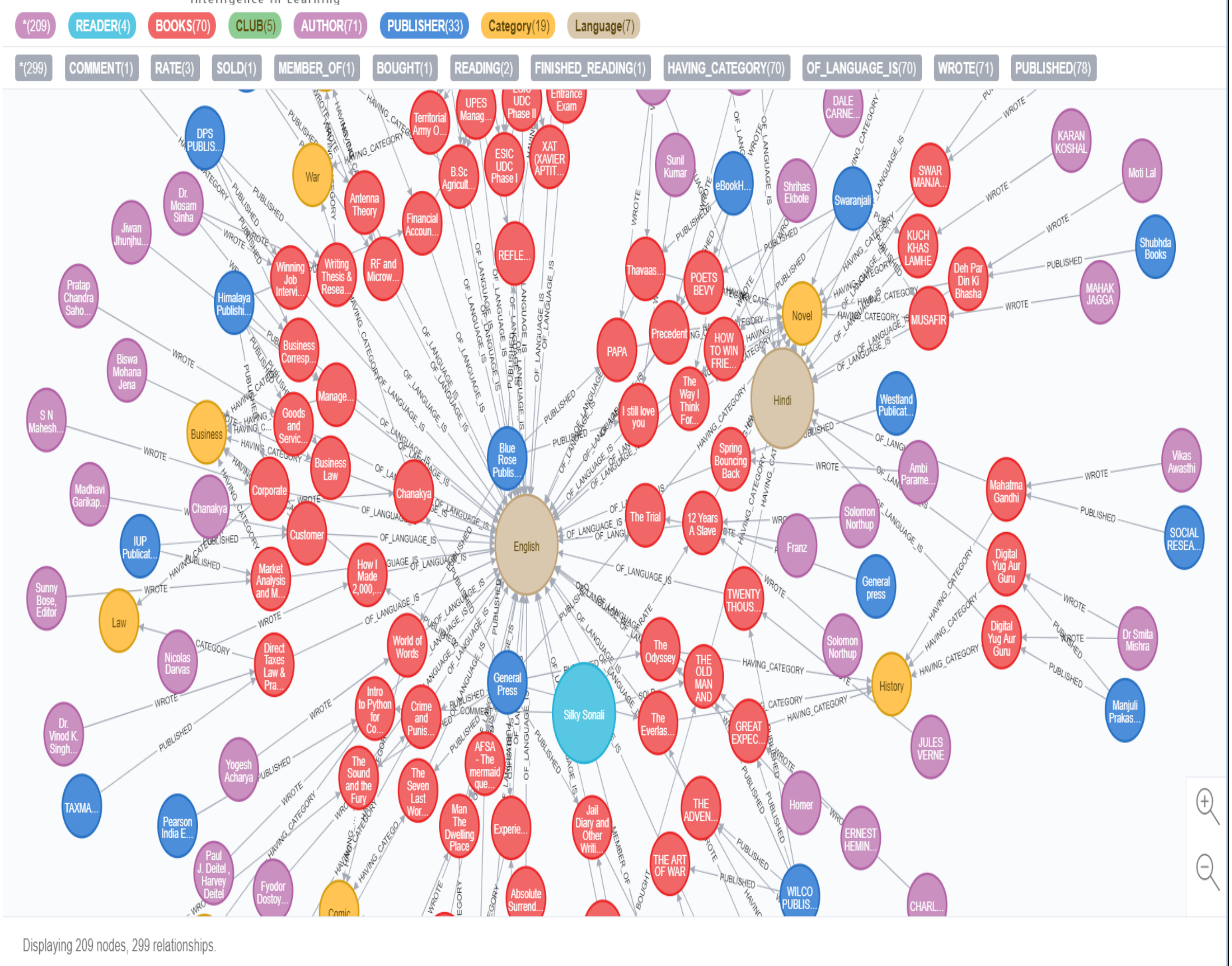
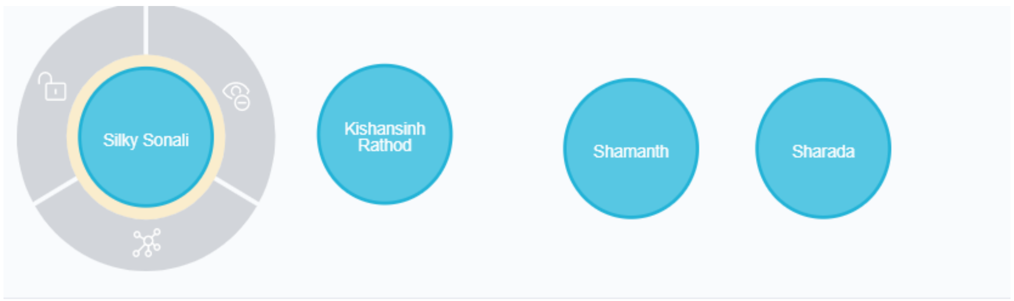


Figure 8 : Detail Database Schema in Neo4j

Nodes:

1. Reader:

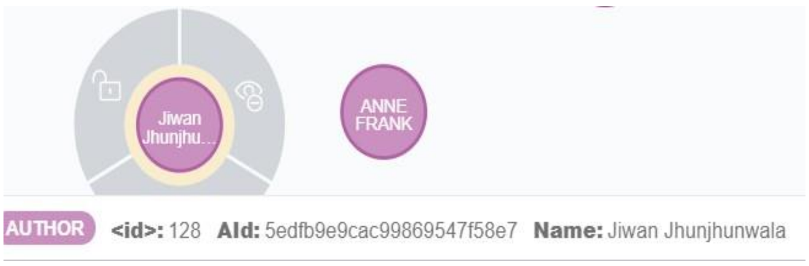


READER <id>: 0 AccType: 2 Name: Silky Sonali country: India email_ID: silkysonali38@gmail.com

Figure 9 : Sample node

This node Contains Information about the Reader of book and Have relations with node Books

2. Authors:

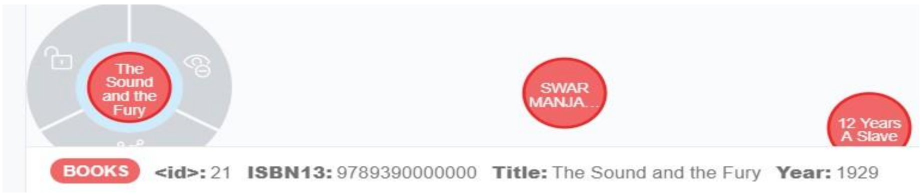


AUTHOR <id>: 128 AId: 5edfb9e9cac99869547f58e7 Name: Jiwan Jhunjhunwala

Figure 10:Sample node of Author

This node Contains Information about Authors of book and Have relations with node Books

3. eBooks:

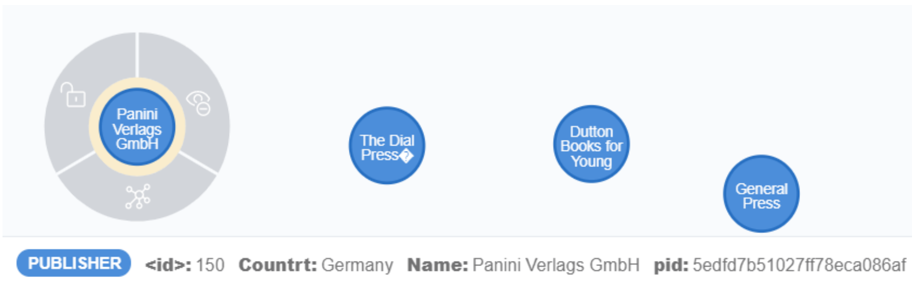


BOOKS <id>: 21 ISBN13: 9789390000000 Title: The Sound and the Fury Year: 1929

Figure 11 : Sample node of eBook

This node Contains Information about Authors of book and Have relations with node Language and Category

4. Publication House:

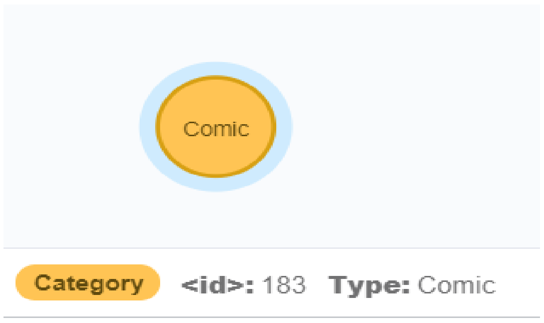


PUBLISHER <id>: 150 Countrt: Germany Name: Panini Verlags GmbH pid: 5edfd7b51027ff78eca086af

Figure 12: Sample Node of Publication House

This node Contains Information about Authors of book and Have relations with node Books

5. Comic:



This node Contains Information about Category of book and Have relations with node Books

Figure 13: Sample Node of book Category

6. Language:

This node Contains Information about Language of book and Have relations with node Books

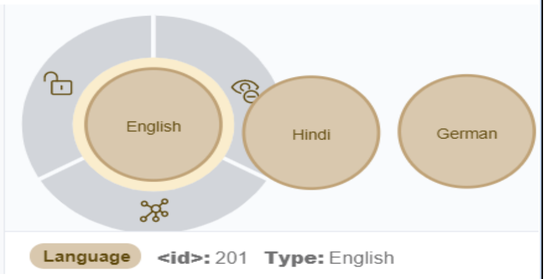
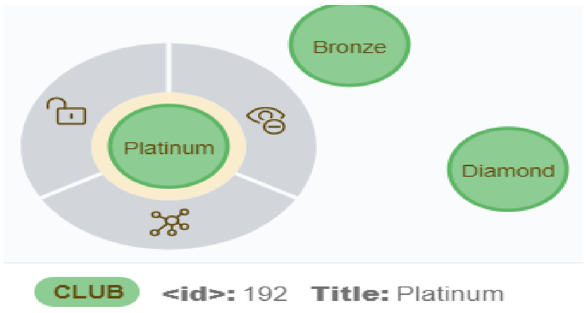


Figure 14: Sample Node of Language

7. Marketing Club:



This node Contains Information about Different Marketing Club of user which is later used for Affiliate Marketing Purpose

Figure 15: Sample Node of Marketing Club

8. Share a book with comment within a group:

These three nodes contain information about group name, shared book, and comment for this book respectively.

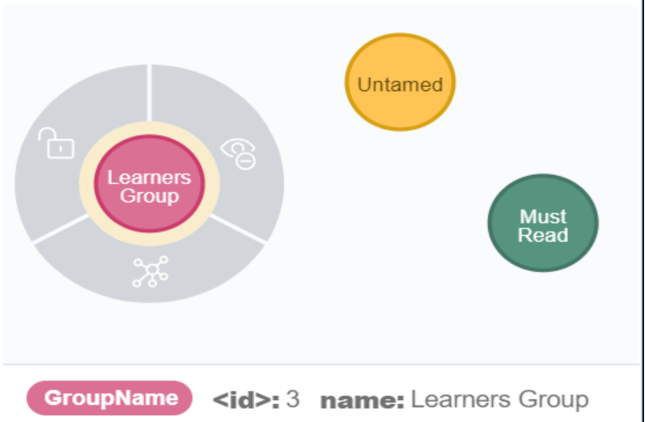
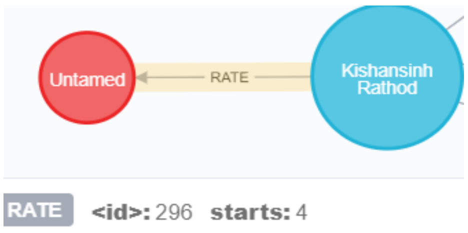


Figure 16 : Sample Node of Share Book

Relations

1. Rating

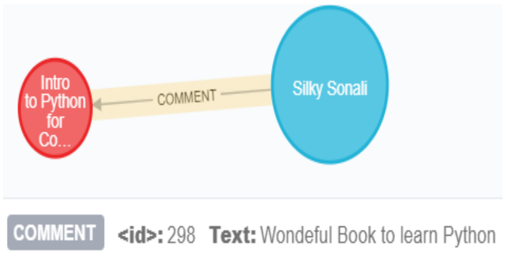


RATE <id>: 296 starts: 4

This is relation between users and Books. This relation will be created when they rate any book. It has property which contains rating in numeric value

Figure 17: Sample Relation Rating

2. Comment



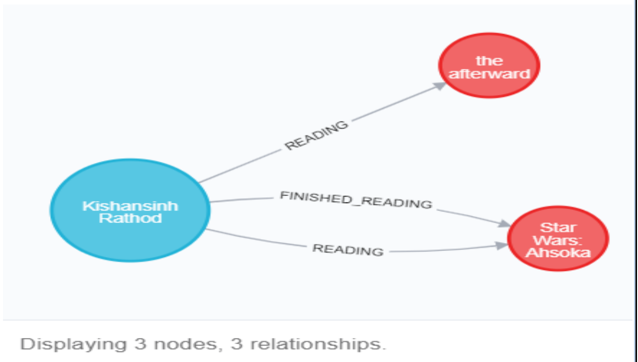
COMMENT <id>: 298 Text: Wondeful Book to learn Python

This is relation between users and Books. This relation will be created when user comment on any book. It has property which contains Comment Text

Figure 18 : Sample Relation Comment

3. Reading and Finished Reading

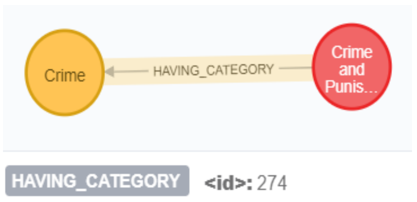
When user start reading any book then new relationship will be created with the Name “READING” to maintain users currently reading list. When they finish any book then READING relation will be deleted and Finished Reading will be created to maintain user’s reading history.



Displaying 3 nodes, 3 relationships.

Figure 19 : Sample Relation: Reading and Finished Reading

4. Book HAVING_Category



HAVING_CATEGORY <id>: 274

Having_category relation is used to show books category.

Figure 20 : Sample Relation Category

5. WROTE

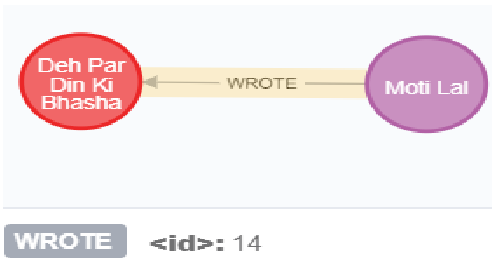


Figure 21 : Sample Relation Wrote

Connection between Author and eBook node like who is Author of which book

6. Published

Connection between Author and eBook node like who published what.

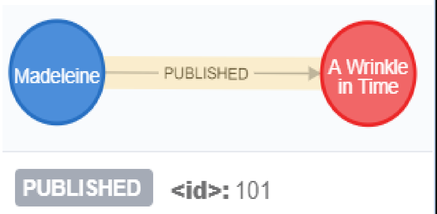
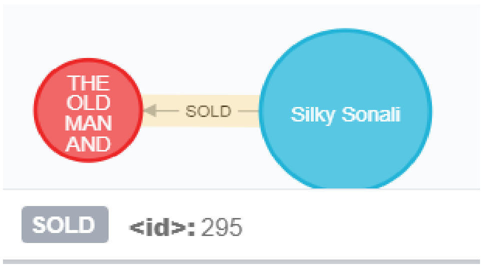


Figure 22: Sample Relation Published

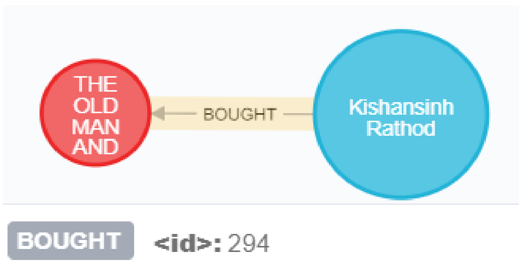
7. Book Sold



When user sold any book book then new relation with name SOLD will be created which indicate which user sold this book

Figure 23 : Sample Relation Book Sold

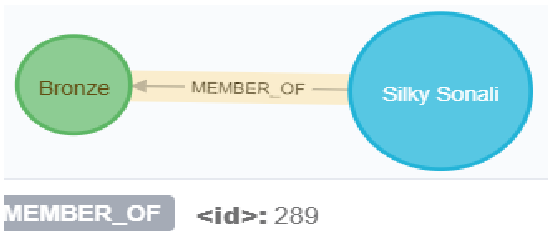
8. Book Bought by User



Having_category relation is used to show books category.

Figure 24 : Sample Relation Book Bought

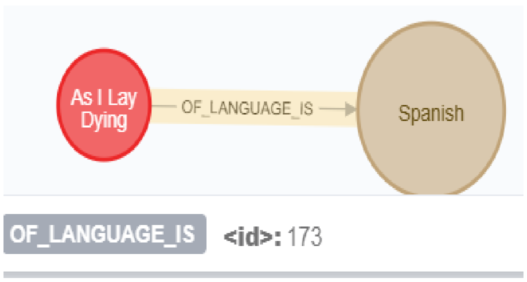
9. User Member of Club



This relation indicates user member ship of marketing club, in which marketing club they belongs.

Figure 25 : Sample Relation Marketing club member

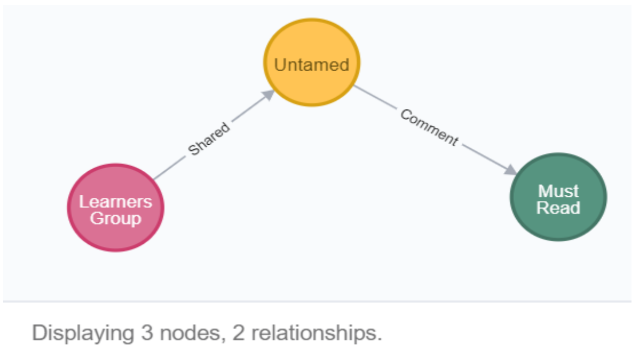
10. Language of Book



This relation indicates Language of eBook.

Figure 26: Sample Relation Book Language

11. Share book with a comment within a group



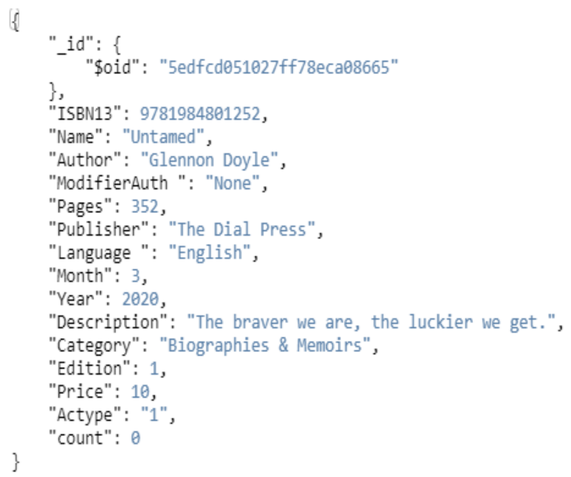
When user shares a book with comment then the relationship between the group, the book shared, and the comment is as represented in this node diagram.

Figure 27 : Sample Relation

5.3. Document Data model

Listed below collection we stored in MongoDB

1. eBooks

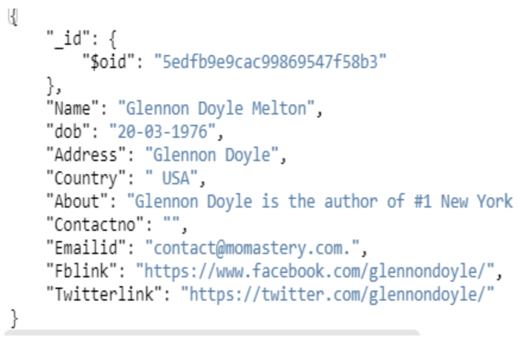


```
{
    "_id": {
        "$oid": "5edfcd051027ff78eca08665"
    },
    "ISBN13": 9781984801252,
    "Name": "Untamed",
    "Author": "Glennon Doyle",
    "ModifierAuth ": "None",
    "Pages": 352,
    "Publisher": "The Dial Press",
    "Language ": "English",
    "Month": 3,
    "Year": 2020,
    "Description": "The braver we are, the luckier we get.",
    "Category": "Biographies & Memoirs",
    "Edition": 1,
    "Price": 10,
    "Actype": "1",
    "count": 0
}
```

eBook Collection contains all the information related eBooks. Sowing in figure 5.3.1 is the sample JSON document of collection eBooks having some main field like ISBN13 which is unique and using for querying purpose, Title, Author, Publisher, Language, Category etc..

Figure 28: Sample JSON document of eBooks collection

2. Authors

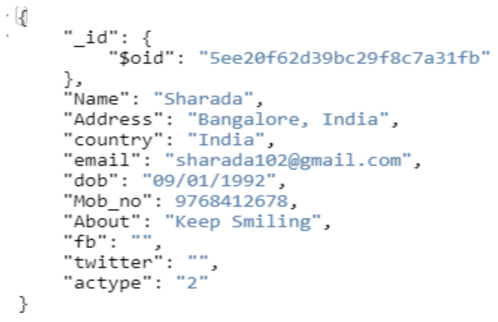


```
{
    "_id": {
        "$oid": "5edfb9e9cac99869547f58b3"
    },
    "Name": "Glennon Doyle Melton",
    "dob": "20-03-1976",
    "Address": "Glennon Doyle",
    "Country": " USA",
    "About": "Glennon Doyle is the author of #1 New York
    "Contactno": "",
    "Emailid": "contact@momastery.com.",
    "Fblink": "https://www.facebook.com/glennondoyle/",
    "Twitterlink": "https://twitter.com/glennondoyle/"
}
```

Author Collection contains all the information related to the Authors of eBooks. Showing in figure 5.3.2 is the sample JSON document of collection Authors having some main fields like:
Name, Email_Id, Country.

Figure 29 : Sample JSON document of Author collection

3. Book Readers

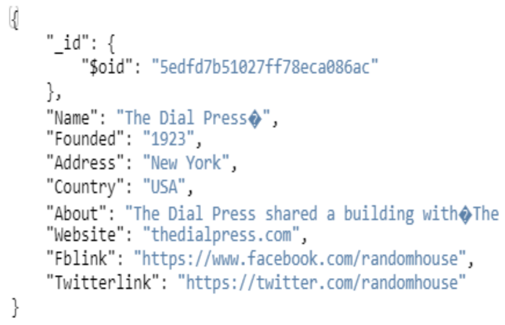


```
{
    "_id": {
        "$oid": "5ee20f62d39bc29f8c7a31fb"
    },
    "Name": "Sharada",
    "Address": "Bangalore, India",
    "country": "India",
    "email": "sharada102@gmail.com",
    "dob": "09/01/1992",
    "Mob_no": 9768412678,
    "About": "Keep Smiling",
    "fb": "",
    "twitter": "",
    "actype": "2"
}
```

Reader Collection contains all the information related to the User who is going to read eBooks. There are two type of reader Free Membership (Limited books) holder and Premium (Have all books access). Showing in figure 5.3.3 is the sample JSON document of collection Readers.

Figure 30 : Sample JSON document of Reader collection

4. Publication House

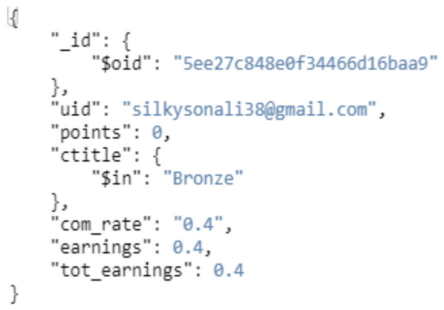


```
{
    "_id": {
        "$oid": "5edfd7b51027ff78eca086ac"
    },
    "Name": "The Dial Press�",
    "Founded": "1923",
    "Address": "New York",
    "Country": "USA",
    "About": "The Dial Press shared a building with�The
    "Website": "thedialpress.com",
    "Fblink": "https://www.facebook.com/randomhouse",
    "Twitterlink": "https://twitter.com/randomhouse"
}
```

Publisher Collection contains all the information related Publication House which are published to eBooks. Showing in figure 5.3.4 is the sample JSON document of collection Readers

Figure 31 : Sample JSON document of Reader collection

5. Affiliate Marketing

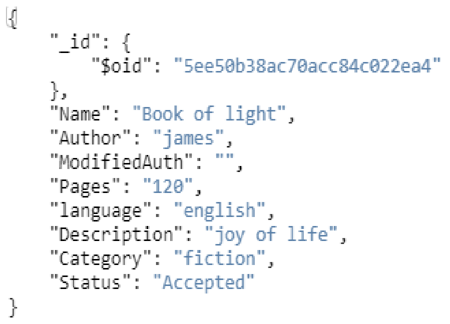


```
{
    "_id": {
        "$oid": "5ee27c848e0f34466d16baa9"
    },
    "uid": "silkysonali38@gmail.com",
    "points": 0,
    "ctitle": {
        "$in": "Bronze"
    },
    "com_rate": "0.4",
    "earnings": 0.4,
    "tot_earnings": 0.4
}
```

This Collection contains all the information related User who joins Affiliate marketing and start earing. Showing in figure 5.3.5 is the sample JSON document of collection Readers. It contains information like userid, earnings, points

Figure 32 : Sample JSON document of Reader collection

6. Publication Request

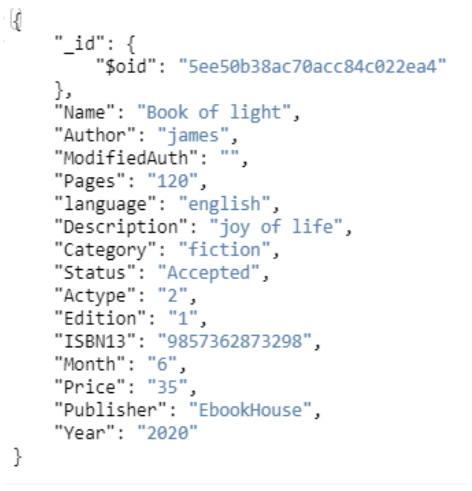


```
{
    "_id": {
        "$oid": "5ee50b38ac70acc84c022ea4"
    },
    "Name": "Book of light",
    "Author": "james",
    "ModifiedAuth": "",
    "Pages": "120",
    "language": "english",
    "Description": "joy of life",
    "Category": "fiction",
    "Status": "Accepted"
}
```

This Collection contains all the information related to the Users who have made a publication request to publish their book. Showing in figure 5.3.6 is the sample JSON document of collection Publication request. It mentions the status of the request which can either be pending or accepted.

Figure 33 : Sample JSON document of Reader collection

7. Publication History

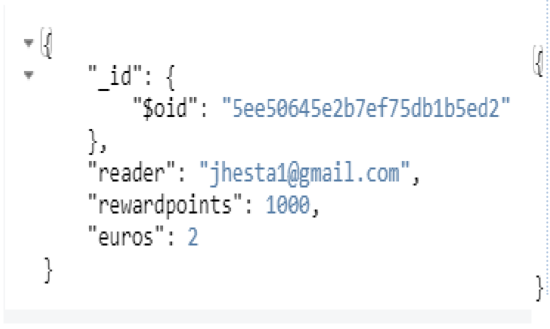


```
{
    "_id": {
        "$oid": "5ee50b38ac70acc84c022ea4"
    },
    "Name": "Book of light",
    "Author": "james",
    "ModifiedAuth": "",
    "Pages": "120",
    "language": "english",
    "Description": "joy of life",
    "Category": "fiction",
    "Status": "Accepted",
    "Actype": "2",
    "Edition": "1",
    "ISBN13": "9857362873298",
    "Month": "6",
    "Price": "35",
    "Publisher": "EbookHouse",
    "Year": "2020"
}
```

This Collection contains all the information related to the Users whose publication requests have been accepted. Showing in figure 5.3.7 is the sample JSON document of collection Publication History.

Figure 34 : Sample JSON document of Reader collection

8. Rewards Point

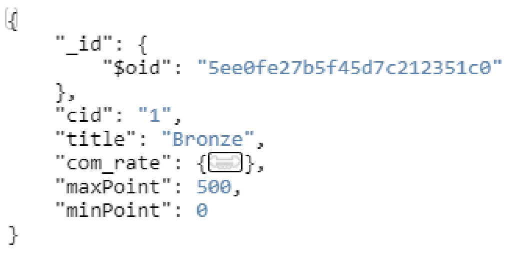


```
{
    "_id": {
        "$oid": "5ee50645e2b7ef75db1b5ed2"
    },
    "reader": "jhesta1@gmail.com",
    "rewardpoints": 1000,
    "euros": 2
}
```

This Collection contains all the information related to the points a reader can make when he buys a book. These points are converted to Euros, which can be redeemed the next time he makes a purchase. Showing in figure 5.3.8 is the sample JSON document of collection Points. It contains information like userid, rewardpoints and euros.

Figure 35 : Sample JSON document of Reader collection

9. Marketing Clubs

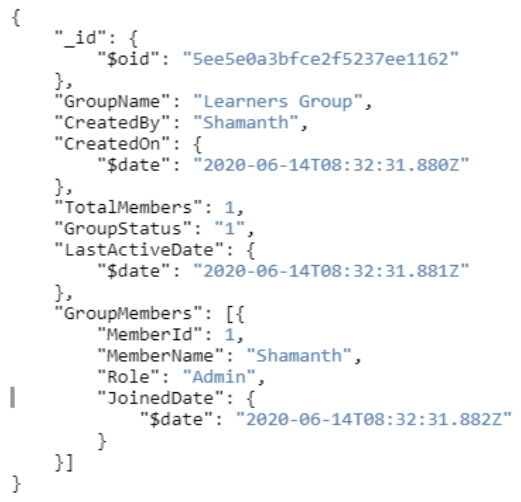


```
{
    "_id": {
        "$oid": "5ee0fe27b5f45d7c212351c0"
    },
    "cid": "1",
    "title": "Bronze",
    "com_rate": {…},
    "maxPoint": 500,
    "minPoint": 0
}
```

This Collection contains information of the commission rate a user is going to receive when he markets a book. Showing in figure 5.3.9 is the sample JSON document of collection Marketing Clubs.

Figure 36 : Sample JSON document of Marketing collection

10.Group List



```
{
    "_id": {
        "$oid": "5ee5e0a3bfce2f5237ee1162"
    },
    "GroupName": "Learners Group",
    "CreatedBy": "Shamanth",
    "CreatedOn": {
        "$date": "2020-06-14T08:32:31.880Z"
    },
    "TotalMembers": 1,
    "GroupStatus": "1",
    "LastActiveDate": {
        "$date": "2020-06-14T08:32:31.881Z"
    },
    "GroupMembers": [{
        "MemberId": 1,
        "MemberName": "Shamanth",
        "Role": "Admin",
        "JoinedDate": {
            "$date": "2020-06-14T08:32:31.882Z"
        }
    }]
}
```

When a registered reader creates a group all the details of the group is stored in this collection. Whenever a new reader is invited into the group his entry will be added as a new field in the "GroupMembers" array. Every time a new member is added to the group the total members is incremented by 1.
The group description of the collection group_list is as shown in the figure 5.3.10 in JSON format.

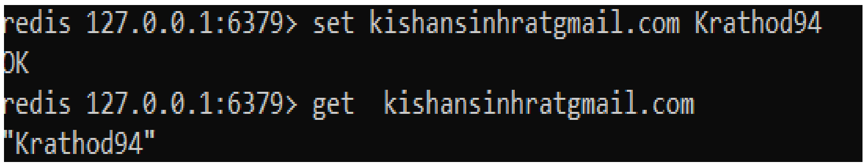
Figure 37: Sample JSON document of Group List collection

5.4. Key-Value Model

Redis data base has been used in our data model to store the data of,

<u>User's Login Data</u>

To store user's login

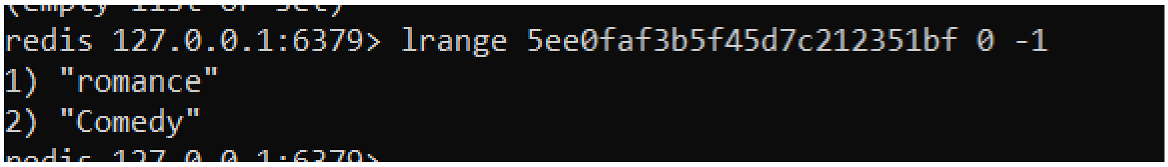


```
redis 127.0.0.1:6379> set kishansinhratgmail.com Krathod94
OK
redis 127.0.0.1:6379> get  kishansinhratgmail.com
"Krathod94"
```

<u>User's Search cache</u>

To store users search key word we are storing Lits type of Redis.

We are adding keyword to list by using LPUSH then user's objectID (from mongodb) as a reference of list Key and then user's search Keywords



```
redis 127.0.0.1:6379> lrange 5ee0faf3b5f45d7c212351bf 0 -1
1) "romance"
2) "Comedy"
```

5.5. Alternative: Integrated Data Model

During development of our data model itself we created around 10-15 data models which we eventually discarded due to different reason. But we have one alternative Memory Optimized model for current data model

Memory Optimized model

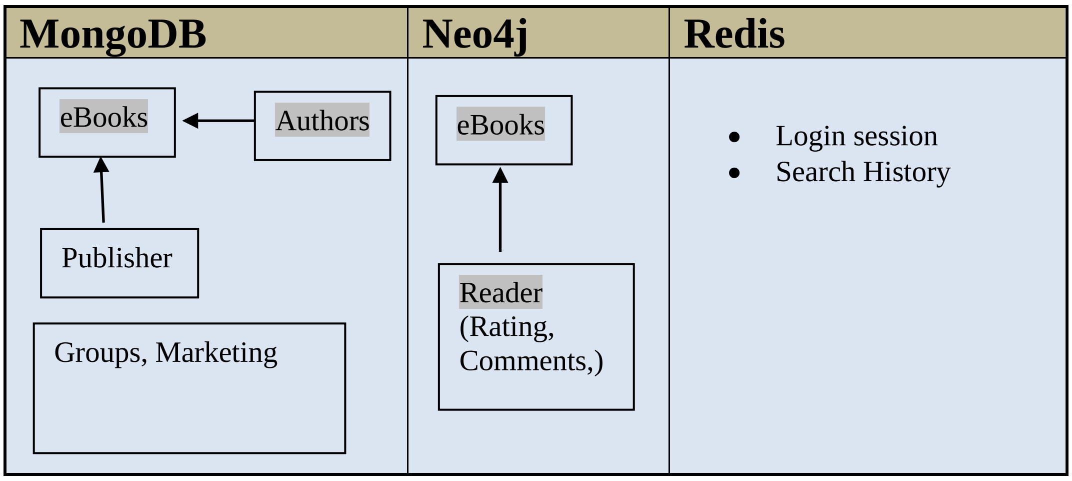


Figure 40 : Alternative Model : Memory Optimized Model

Advantages:

- Memory usage of this model is very less in comparison to the current model as most of the data are stored in MongoDB which requires less memory compare to Neo4j.
- As we can find the data all at one place so this can avoid unnecessary jumping between database to maintain performance.

Disadvantage:

- Execution of mongodb is comparatively slow when we try to implement search usecase
- Increase complexity while implementing relation.
- Compromised performance

6. Implementation

6.1. DATA MODEL IMPLEMENTATION

Our data model is being implemented by systematic insertion of all our datasets. Since some datasets have dependencies on other datasets like eBook to Author, Publisher to eBook, eBook to Category etc. At first we scrapped data from different web resources and then manually entered in CSV file in our required format.

1. Let us assume one user want to find book with Category Comic then want to rate and comment on that.

For that we first must enter data in eBook collection

```
{
"ISBN13": "9781984801252",
   "Name": "Untamed",
   "Author": "Glennon Doyle",
   "ModifierAuth ": "None",
   "Pages": 352,
   "Publisher": "The Dial Press",
   "Language ": "English",
   "Month": 3,
   "Year": 2020,
   "Description": "The braver we are, the luckier
we get.",
   "Category": "Biographies & Memoirs",
   "Edition": 1,
   "Price": 10,
   "Actype": "1",
   "count": 0
}
```

Figure 40 : Sample Json

After adding data in MongoDB we need to create node in Neo4j then relation of category will given with category node.In this case following relation will be given

a) (:BOOKS{ISBN13:"9781984801252", Title:"Untamed"})-[:HAVING_CATEGORY]->(:Category{Name:"Comic"})

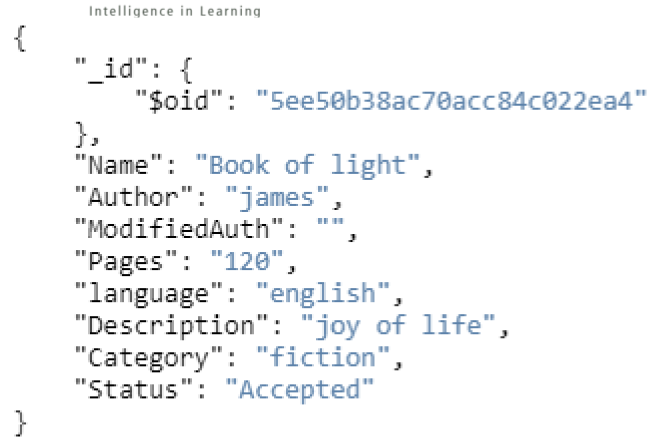
b) (:READER {email:"kishansinhr@gmail.com", Name:"Kishan"})

c) (:READER)-[:COMMENT{Text:"Good Book"}]->(:BOOKS)

d) (:READER)-[:RATE{star:4}]->(:BOOKS)

Part (a) will give relation between book node and category node and (b) will create node of user So using (c) and (d) Comment and rating can be possible and will be shown in neo4j

2. Consider case where user want to publish a book and want to sell book to other user to earn some commission.



```
{
    "_id": {
        "$oid": "5ee50b38ac70acc84c022ea4"
    },
    "Name": "Book of light",
    "Author": "james",
    "ModifiedAuth": "",
    "Pages": "120",
    "language": "english",
    "Description": "joy of life",
    "Category": "fiction",
    "Status": "Accepted"
}
```

Figure 41: Sample Json

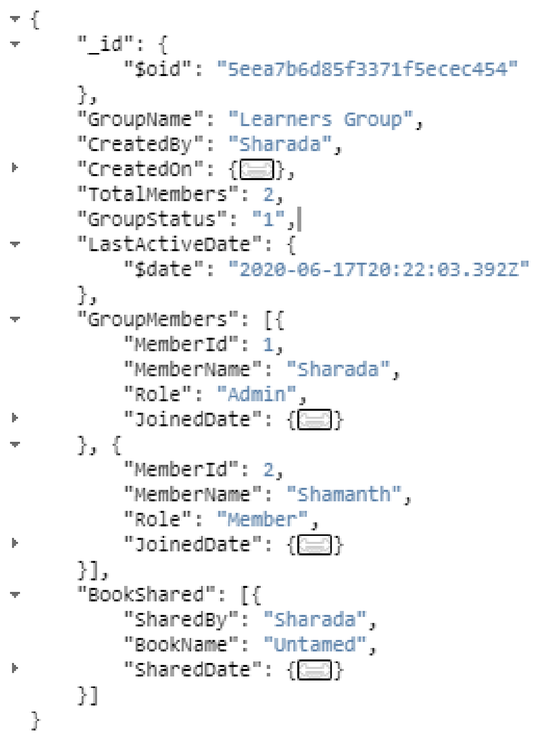
First any user will make request to system that he/she wants to publish a book and record will be stored in respective collection. When request is accepted node of new book will be created in Neo4j as well it will be added in Mongo's Collection. Author will be also added if it is not existing in record and relation

While user want to sell book they can share it to another user and when they purchase it Marketing collection in MongoDB will be updated as well as graph representation will be shown in Neo4j. Consider following

a) (:READER)-[:SOLD] ->(:BOOKS)

b) (:READER)-[:BOUGHT]->(:BOOKS)

3. Consider a case wherein a user wants to create a group, invite other registered users to the group and share a book and comments within.

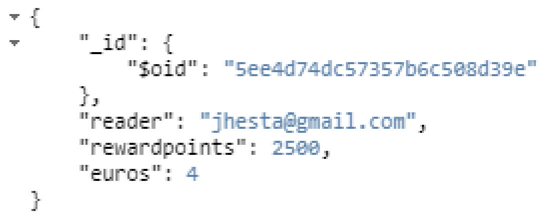


```
{
    "_id": {
        "$oid": "5eea7b6d85f3371f5ecec454"
    },
    "GroupName": "Learners Group",
    "CreatedBy": "Sharada",
    "CreatedOn": {…},
    "TotalMembers": 2,
    "GroupStatus": "1",
    "LastActiveDate": {
        "$date": "2020-06-17T20:22:03.392Z"
    },
    "GroupMembers": [{
        "MemberId": 1,
        "MemberName": "Sharada",
        "Role": "Admin",
        "JoinedDate": {…}
    }, {
        "MemberId": 2,
        "MemberName": "Shamanth",
        "Role": "Member",
        "JoinedDate": {…}
    }],
    "BookShared": [{
        "SharedBy": "Sharada",
        "BookName": "Untamed",
        "SharedDate": {…}
    }]
}
```

In this case the user who is logged in, can create a group. Further the user can invite and add other users into this group. Once the group is created the group details are stored in group_list Collection in MONGODB. Any member of the group can share books within the group. The book shared is also stored in the same Collection.

In addition to this, the group members can also give comment on the shared book and this information is stored in NEO4j.

4. For every purchase of book user gets reward points and the user can redeem it later in his next purchase.



```
{
    "_id": {
        "$oid": "5ee4d74dc57357b6c508d39e"
    },
    "reader": "jhesta@gmail.com",
    "rewardpoints": 2500,
    "euros": 4
}
```

Figure 43 : Sample Json

Everytime a user purchases a book a reward points of 500 will added to points collection in MONGODB against this user. Further the user gets discount of 1 euro for every 500 points which the user can redeem in next purchase. The equivalent discount amount in euros for the user is also stored in this points Collection.

7. Queries

All use-case queries have been stored in GitHub along with python scripts and CSV file as well as README.txt file to guide on how to execute the queries along with output file.

Github Link : Click here

7.1. CSV Import in Neo4j

- **Create Node of Book, Author and give relation**

```
load csv with headers from "file:///E:/AD/Dataset/MongoOut/eBooks.csv"
as row
with row
Merge (B:BOOKS{ISBN13:row.ISBN13})
Merge (A:AUTHOR{Name:row.Author})
Create (A)-[:WROTE]-> (B)
```

- **Create node of Publisher and join it with Book**

```
load csv with headers from "file:///E:/AD/Dataset/MongoOut/eBooks.csv"
as row
with row
Merge (B:BOOKS{Title:row.Name})
Merge (P:PUBLISHER{Name:row.Publisher})
Create (P)-[:PUBLISHED]-> (B)
```

- **Create node of Language and join it with Book**

```
load csv with headers from "file:///E:/AD/Dataset/MongoOut/eBooks.csv"
as row
with row
Merge (B:BOOKS{Title:row.Name})
Merge (L:Language{Type:row.Language})
Create (B)-[:OF_LANGUAGE_IS]-> (L)
```

- **Create node of Category and join it with Book**

```
load csv with headers from "file:///E:/AD/Dataset/MongoOut/eBooks.csv"
as row
with row
Merge (B:BOOKS{Title:row.Name})
Merge (C:Category{Type:row.Category})
Create (B)-[:HAVING_CATEGORY]-> (C)
```

- **Create a node of user**

```
load csv with headers from
"file:///E:/AD/Dataset/MongoOut/Readers.csv" as row
with row
Merge
(R:READER{email_ID:row.email,Name:row.Name,country:row.country,AccType
:row.actype})
```

7.2. Queries for Use-case 2: Currently Reading, Rating, Comment

Flow:

- User have choice, Taking input from user on choice
- User can see their Currently reading List and mark any book in as Finished and book will go in Finished List
- User can see their Finished Reading list
- User can get list on books and start reading new book and it will be in currently reading list
- User can see total Ratings of book and can rate as well as comment on them

Github: Click Here

Queries in Neo4j

- **Add Book in user's currently reading list**

```
MATCH (R:READER{email_ID:"kishansinhr@gmail.com"})
MATCH (b:BOOKS{ISBN13:"9783833234507"})
MERGE (R)-[:READING]->(b)
```

- **Finished Reading and add it in to Finished List**

Remove Relation

```
MATCH (R:READER{ email_ID:"kishansinhr@gmail.com"})
-[r:READING]-> (b:BOOKS{ISBN13: "9783833234507"})
DELETE r
```

Add Book to Finished list

```
MERGE (R:READER{email_ID:"kishansinhr@gmail.com"})
MERGE (b:BOOKS {ISBN13:"9783833234507"})
MERGE (R)-[:FINISHED_READING]->(b)
```

- **Checked Finished Reading List**

```
MATCH (R:READER{ email_ID:"kishansinhr@gmail.com"})
-[:FINISHED_READING]-> (b: BOOKS{ISBN13:" 9780735231917"})
RETURN b.Title
```

- **Browse Book to start reading**

```
MATCH (b:BOOKS) RETURN b.Title as Title, ID(b) as ID LIMIT 10
```

- **Retrieve ratings and comments**

```
MATCH (R:READER{ email_ID:"kishansinhr@gmail.com"})
-[R:RATE]->(B:BOOKS{ISBN13:"9783833234507"}) Return R.starts as
starts

MATCH (R:READER{ email_ID:"kishansinhr@gmail.com"})
-[R:COMMENT]->(B:BOOKS{ISBN13:" 9783833234507"}) Return R.Text
as Text
```

- **Add Rating**

```
MERGE (R:READER{ email_ID:"kishansinhr@gmail.com"})
MERGE (B:BOOKS{ISBN13:"9783833234507"})
MERGE (R)-[:RATE{starts:4}]->(B)
```

- **Add Comments**

```
MERGE (R:READER{ email_ID:"kishansinhr@gmail.com"})
MERGE (B:BOOKS{ISBN13:"9783833234507"})
MERGE (R)-[:COMMENT{Text:”Good”}]->(B)
```

- **Overall Rating**

```
MATCH (p:READER)
MATCH (b:BOOKS{ISBN13:"9789390239726"})
MATCH (p)-[r:RATE]->(b) return Count(r) as count

return SUM (r.starts) as tot
MATCH (p:READER)
MATCH(b:BOOKS{ISBN13:"9789390239726"})
MATCH(p)-[r:RATE]->(b) return SUM(r.start) as Total
```

MongoDB Queries

- **Increase Reading Count**

```
db.eBooks.update({ISBN13:" 9781984801252"}, {$inc: {count: 1}})
```

7.3. Queries for Use-case 5 Affiliate Marketing

- **On joining Affiliate marketing, user's data is inserted in the collection Affiliate Marketing by taking reference from User and Clubs collections**

```
db.affil_marketng.insertOne({"uid" :"silkysonali38@gmail.com",
points:0,
"ctitle":{$in:db.Clubs.find({cid:"1"}).map(function(dir){return
dir.title})[0]},
"com_rate":{$in:db.Clubs.find({cid:"1"}).map(function(dir){retur
n dir.title})[0]}, earnings:0 })
```

- **To find a user with its club category:**

```
db.affil_marketng.find({"ctitle": {$nin:["Bronze"]} })
```

Alternet Query in Neo4j

```
match (R:READER{email_ID:"silkysonali38@gmail.com"})
-[r:MEMBER_OF]->(C:CLUB{Title:"Bronze"})
 return R,C
```

- **To calculate earning (on every successful selling of book) commission rate and book price will be fetched, earning calculated and total earning will be incremented:**

```
db.affil_marketng.find({uid:"silkysonali38@gmail.com"},
{com_rate:1,_id:0}) db.eBooks.find({ISBN13:"9788437608341"},
{Price :1})

db.affil_marketng.update({uid:"silkysonali38@gmail.com"},
{$inc:{earnings:0.4, tot_earnings:0.4} } )
```

- **To show book sold by a user:**

```
match(p:Person{Name:"S
ilky"})
match
(b:Book{Title:"Titanic
"}) create (p)-
[:SOLD]->(b)
```

- **To show a user joining by the reference of another user:**

```
merge
(P:Person{Name:"Jhestha
"}) merge
(P2:Person{Name:"Silky"
}) merge (P)-
[:JOINED_BY]-(P2)
```

- **Give membership of club**

```
merge
(P:Person{Name:"Jhestha
"}) merge
(P2:Person{Name:"Silky"
}) merge (P)-
[:JOINED_BY]-(P2)
```

7.4. Queries for Use-case 4 <u>Collecting Reward Points</u>

- **For every eBook purchased by a reader, 500 Reward points get inserted into his account**

```
var uname = db.Books_bought.findOne(
           {reader:"jhesta@gmail.com"}
           ).reader
db.Points.findAndModify({
           query:{reader:uname},
           update:{
                $inc:{rewardpoints:500}
                  },
           upsert:false
           })
```

- **500 reward points equates to 1 Euro**

```
db.Points.updateMany({},
      [{$set:{euros:{
           $switch:{
            branches:[
             {case:{$eq:["$rewardpoints",500]},then:1},
             {case:{$eq:["$rewardpoints",1000]},then:2},
             {case:{$eq:["$rewardpoints",1500]},then:3},
             {case:{$eq:["$rewardpoints",2000]},then:4}
             ],
             default:0}
             }
            }}
      ])
```

- **The reader can utilize the Euros in his account and redeem them on his next purchase of a book**

```
var get_euros = db.Points.findOne(
                {reader:"jhesta1@gmail.com"}).euros

db.Books_bought.findAndModify(
           {query:{reader:"jhesta1@gmail.com"},
           update:
                {$inc:{price:-get_euros}}
           }
      )
```

- **To show an eBook bought by a Reader**

```
merge (u:Users{name:"Jhesta"} )
merge (b:Books_bought{name:"Book of light"})
create (u)-[:BUYS]->(b)
```

- **To show the reward points a Reader receives on the purchase of 1 book**

```
merge (u:Users{name:"Jhesta"} )
merge (p:Points{rewardpoints:500})
create (u)-[:GETS]->(p)
```

- **To show the conversion of reward points to Euros**

```
merge (u:Users{name:"Jhesta"} )
merge (e:Euros{euros:1})
create (u)-[:CAN_REDEEM]->(e)
```

7.5 Queries for Use-case 1 <u>Publish Book</u>

- **For Query will place the request sent by the Reader or the Author to Pub_request collection, the status of all the requests will be "Pending" before the reviewer approves or rejects the request**

```
db.Pub_request.insert({
        Name:"Book of light",
         Author:"james",
        ModifiedAuth:"",
        Pages:"120",
        language:"english",
        Description:"joy of life",
        Category:"fiction"
})

db.Pub_request.update({},
        {$set:{"Status":"Pending"}},
        {upsert:false,multi:true}
)
```

- **Reviewer checks for the requests and examines them, also will set the status to Accepted or Rejected based on the books he needs from his publication and the response will be sent to Pub_history collection.**

```
db.Pub_request.update(
        {"Name":"Book of light"},
        {$set: {Status: "Accepted"}
})

db.Pub_history.save(db.Pub_request.find(
        {Status:"Accepted"}).toArray()
)
```

- **When a book is Accepted and Published, all the necessary details like ISBN, Price,**

Edition will be included by the Publication house and will be sent to eBooks collection where the Reader, once he logs in, can view it.

```
db.Pub_history.update(
   {Name:"Book of light"},
   {$set:{
        "ISBN13":"9857362873298",
        "Publisher":"EbookHouse",
        "Month":"6",
        "Year":"2020",
        "Edition":"1",
        "Price":"35",
        "Actype":"2"}},
   {upsert:false,multi:true})

db.eBooks.save(db.Pub_history.find(
   {Name:"Book of light"}.toArray())
```

- **Query will show the Book Published by the Publication House**

```
    merge (B:eBooks{eBName:"Book of light"} )
    merge (P:Publisher{PName:"eBookHouse"} )
    create (P)-[:PUBLISHED]->(B)
```

- **Query shows the category of the book.**

```
    merge (eB:eBooks{eBName:"Book of light"} )
    merge (C:Catogery{Catname:"Fiction"} )
    create (eB)-[:HAVING_CATEGORY]->(C)
```

- **Query shows the Book written by the Author.**

```
    merge (eB:eBooks{eBName:"Book of light"} )
    merge (Au:Autname{AuName:"Jameson"} )
    create (Au)-[:WROTE]->(eB)
```

- **Query shows the Language of the Book**.

```
    merge (eB:eBooks{eBName:"Book of light"} )
    merge (L:Language{LName:"English"} )
    create (eB)-[:Language]->(L)
```

7.6 Queries for Use-case 3: Create Group and share book within the group

1. User wants to create a group

Description: When a user creates a new group, the group details are added into the group_list collection. This user is the first member of the group and role is "Admin". When a new group is created the "GroupStatus" is set to 1= active. If no books shared or comments made in this group for more than 90 days, then the "GroupStatus" is set to 0= inactive.

Query: The query to add the group details for the first time when a user creates a group is as given below: In this example user email_id is "sharada102@gmail.com" and this user wants to create a group naming "Learners Group".

- To find the name from the Users Collection using email_id

```
➢ var uname = db.Users.findOne({"email":
  "sharada102@gmail.com"}).Name
```

- To create a group

```
➢ var insert_grouplist_creategroup = {
       GroupName: "Learners Group", CreatedBy: uname, CreatedOn :
       new Date(), TotalMembers: 1, GroupStatus:"1",
       LastActiveDate: new   Date(),
       GroupMembers:[{ MemberId: 1, MemberName: uname, Role:
       "Admin",JoinedDate: new Date() }]
     }
```

- To insert newly created group into group_list Collection

```
➢ db.group_list.insert(insert_grouplist_creategroup)
```

2. User can add other users to the group

Description: In a real scenario, when a user accepts the invitation to join the group a response message is received from this user. This response message consists of two information:

a. User email_id
b. Group_id (to which he/she has accepted)

Using these details, the below queries are constructed to add this member into the group. In this example, we consider group name. (just for example)

Query: In this example "uname" is username found by corresponding email_id. Group Id is found from the corresponding group name, using this unique group Id corresponding group details will be updated with new member details and incremented total members.

- To get name using email_id from Users Collection.

➢
```
var uname = db.Users.findOne({"email":
"shamanth654@gmail.com"}).Name
```

- To get current total members using GroupName from “group_list” Collection and increment it by 1.

➢
```
var Tmembers = db.group_list.findOne({"GroupName": "Learners
Group"}).TotalMembers + 1
```

- To get Group_id(gid) using group name (GroupName) from Group_list Collection.

➢
```
var gid = db.group_list.findOne({"GroupName": "Learners
Group"})._id
```

- To update the group_list collection with incremented total members and new group members with all the details.

➢
```
db.group_list.update(
          {id: gid },
          {$inc: {TotalMembers: 1},
           $push: {GroupMembers: {MemberId: Tmembers, MemberName:
          uname, Role: "Member", JoinedDate: new Date()}}
          })
```

3. User wants to leave the group:

Description: At any point, any member of the group can leave the group and corresponding user details are removed from the group_list collection and Total members count is reduced by 1.

Query: Find the username from the “Users” collection using the user’s login email ID. In this example I have taken username as “Shamanth”. Delete the user’s details from the Group_list and decrease the TotalMembers count by 1.

- To get name using email_id from Users Collection.

➢
```
var uname =  db.Users.findOne({"email":
"shamanth654@gmail.com"}).Name
```

- To update the group_list by removing the user and to decrement the Total Members count

➢
```
db.group_list.update(
     { },
     {$pull: {GroupMembers: {MemberName: uname}}},
     {multi: true})
```

- To update group_list collection with decrease in total member count by 1

```
➢ db.group_list.findOneAndUpdate({"GroupName": "Learners Group"},
  {$inc: {TotalMembers: -1}})
```

4. Admin wants to leave the group:

Description: Admin can leave the group at any point of the time. When the admin leaves the group, the role of the second member of the group is updated from group member to group admin and total members count is decreased by 1.

Query: Find the username from the “Users” collection using the user’s login email ID. In this example username taken is “Sharada”. Remove the Admin member details from the group_list Collection. Update the second member of the group as admin and decrease the total members count by 1.

- To get name using email_id from Users Collection.

```
➢ var uname = db.Users.findOne({"email":
  "sharada102@gmail.com"}).Name
```

- To remove the admin from the group_list

```
➢ db.group_list.update(
   { },
   {$pull: {GroupMembers: {MemberName: uname, Role: "Admin"}}},
   {multi: true}
 )
```

- To update the second member of the group as the group_admin

```
➢ db.group_list.findOneAndUpdate({"GroupName": "Learners Group"},
     {$inc: {TotalMembers: -1},
      $set: {"GroupMembers.0.MemberId" : 1, "GroupMembers.0.Role" :
    "Admin"}})
```

5. User wants to share a book

Description: Any member of the group can share a book within the group. And all the details of the book shared within the group is updated in group_list collection.

Query: Find the username from the “Users” collection using the user’s login email ID. In this example username taken is “Sharada”. Book name is found from the “ebooks” Collection using unique “ISBN number”. Update the shared book details such as- book shared by, book name, date on which book is shared.

- To get name using email_id from Users Collection.

```
➢ var uname = db.Users.findOne({"email":
  "sharada102@gmail.com"}).Name
```

- To get Book name by using ISBN book number

```
➢ var bname = db.eBooks.findOne({"ISBN13": "9781984801252"}).Name
```

- To update the group_list Collection with book shared details

```
➢ var gid = db.group_list.findOne({"GroupName": "Learners
  Group"})._id
➢ db.group_list.update(
        {id: gid},
        {
          $push: { BookShared: {SharedBy: uname, BookName: bname,
          SharedDate: new Date()} }
        })
```

6. Admin wants to delete the group:

Description: If Admin wants to delete the group a notification will be sent to the members of the group and the group entry in the group_list Collection gets deleted.

Query: Find the group name using the unique objectID of the group entry from the group_list collection. In this scenario, the group name is “Learners Group”, which is removed from the group_list Collection.

- To delete the group details from the group_list Collection

```
➢ db.group_list.remove({_id: db.group_list.findOne({"GroupName":
  "Learners Group"})._id })
```

7. User sharing a book with comments within a group:

Query in Neo4j:

- To create nodes for GroupName, Bookshared and Comments

```
CREATE (:GroupName {name: "Learners Group"})
CREATE (:BookName {name: "Untamed", sharedby: "Shamanth"})
CREATE (:Comment_1 {name: "Must Read", commentby: "Shamanth"})
```

- To create Relationship between nodes

```
MATCH (a:GroupName),(b:BookName)
CREATE (a)-[:Shared]->(b)

MATCH (a:BookName),(b:Comment_1)
CREATE (a)-[:Comment]->(b)
```

- When a new comment is made within the group

```
merge(a:BookName {name: "Untamed"})
merge(b:Comment_2 {name:"Thanks",commentby:"Sharada"})
CREATE (a)-[:Comment]->(b)
```

8. Evaluation

On Data model:

- We can store some more information on Redis like Trending Books, Reading Counter and Overall rating to make execution fast on loading Home page of application because this is some basic information that needed on startup of application
- We could have store Publisher data in MongoDB only and not in Neo4j as it is less searched parameter.

On usecase :

- In usecase 2, we could have store overall rating in Redis when there is new rating done on on any book.
- In usecase 5, we could have avoid storing some information in Neo4j like club details of user to make Memory optimization
- In usecase 5 and usecase 3, we could have shift selling data to MongoDB only instead on Neo4j to optimize memory

On User Experience

- We would make a whole working system on web application with REST Apis and Server with interactive Web UI.

On Project Management :

- We would have manage detailed information of minutes of meetings

9. Bibliography

1) https://books.google.com/

2) https://www.amazon.com/amazon-books/b?ie=UTF8&node=13270229011

3) http://isbn.gov.in/Recently_Published_Books.aspx

4) https://openlibrary.org/

5) https://www.wikipedia.org/

6) https://neo4j.com/docs/

7) https://docs.mongodb.com/

8) https://redis.io/documentation

9) https://pandas.pydata.org/

10) https://pandas.pydata.org/

11) https://www.tutorialspoint.com/mongodb/index.htm

12) https://www.guru99.com/mongodb-tutorials.html

13) https://medium.com/neo4j/neo4j-graph-algorithms-release-memory-requirements-concurrency-settings-bug-fixes-37c501df105d