Large-Scale and Multi-Structure Database

Reviook Application Report

Application developed by
Arancio Febbo Salvatore, Di Donato Mattia, Giorgi Matteo

Summary

Introduction

To implement our project, we use Java and Maven with the support of Scene Builder to manage fxml files for the interface of the application.

Data have been stored into MongoDB and Neo4J database. MongoDB was used for store and retrieve information about Users, Authors and Books while the task of storing and retrieving social information that associate Users and Books was assigned to Neo4J.

This application was developed to provide authors the possibility to share their titles and receive from readers the feedback on their works, for this reason the name of the application came from the union of Review and Book. More in detail a user/author can register/login, can search for other users/authors profiles and view statistics of them profile, view suggested users and books, view a book, and leave a review about it. Users and authors have on their profile a useful section for the ranking analytics.

An additional actor of the application is the administrator that have the possibility to search and view authors/users/books profile as the other actor, but has also the capacity to delete reviews, books and user's/author's account.

The application repository is on GitHub

Functional and Non-Functional requirements

This paragraph describes all the functional and not functional requirements that the Reviook application satisfy.

The application manages three kinds of actors: User, Author, Admin.

- A user has the access to basic function of the application
- An author can also add and delete his/her books
- An admin has all the function of di application except the social ones and can also decide to delete users/authors/reviews/books.

Functional Requirements

- An unregister user can only sign up to become a registered user.

A **user** is allowed to:

- Login/Register
- Logout
- View its homepage
- Delete or modify its account
- View Ranking statistics
- Browse information about:
  - Users
  - Authors
  - Books
- View users/authors profile
- Follow or unfollow another user/author
- View book profile
- Insert a book into read or to_read list
- Review a book
- Like a review
- Report books/reviews
- Modify delete its own review
- View suggested:
  - Users
  - Authors
  - Books

An **author** is a **user** who is also allowed to:

- Add new books
- Delete its books

An **admin** is allowed to:

- Browse information about:
    - Users
    - Authors
- Browse reviews and books reported
- Delete a book
- Delete a review
- Delete user/author account

Non-Functional requirements

- The application should be simple, intuitive and fast in response.
- The password must be protected and stored in an encrypted form and must be between 8-20 characters.
- The user's and author's username must be unique.
- The application should store locally information to reduce network traffic and DBs load.
- The code must be easy to maintain.
- Data may not be updated for up to 10 minutes

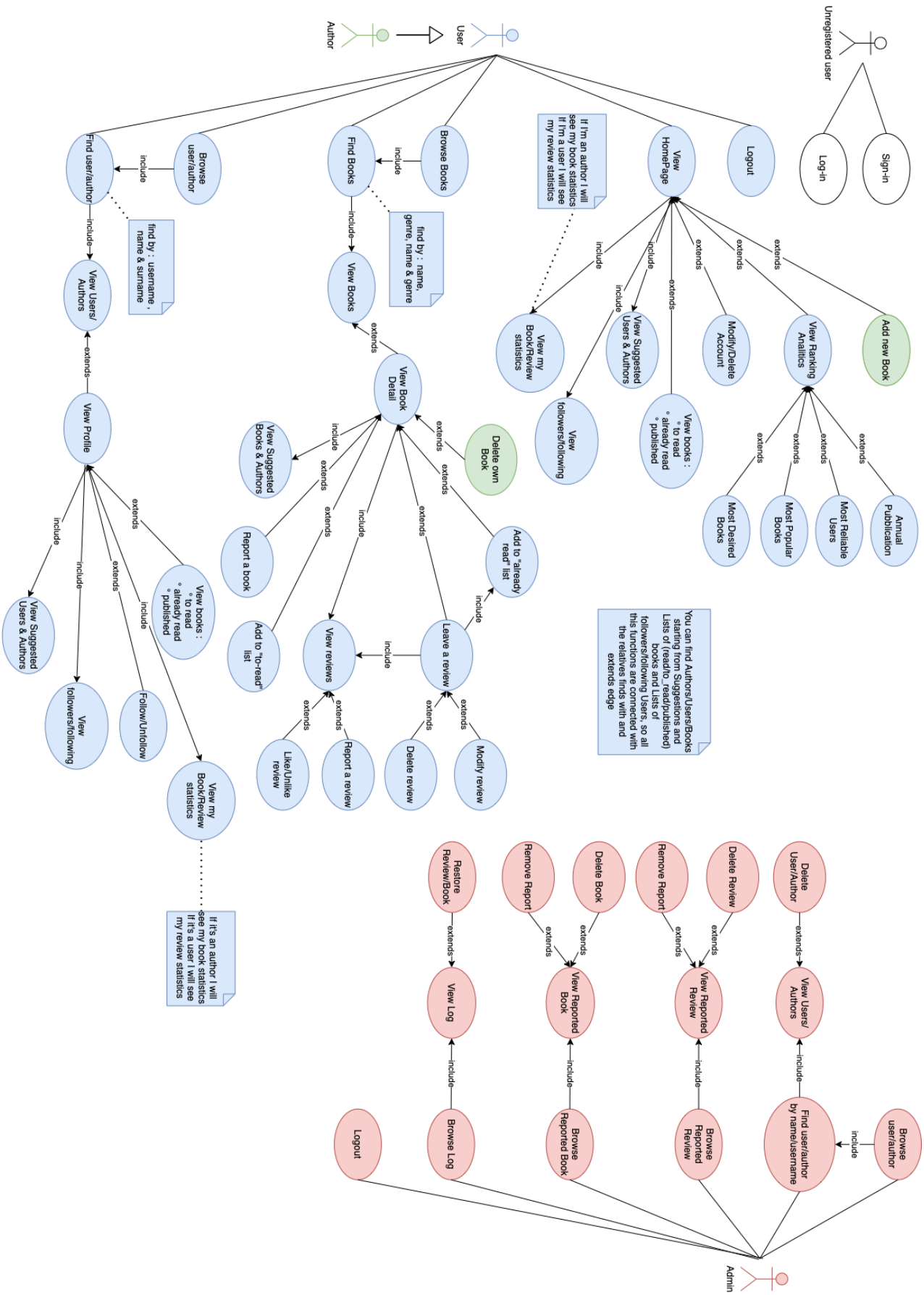
Dataset

We collect information about books, reviews, authors and users from two different sources :
https://snap.stanford.edu/data/amazon-meta.html
https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home .

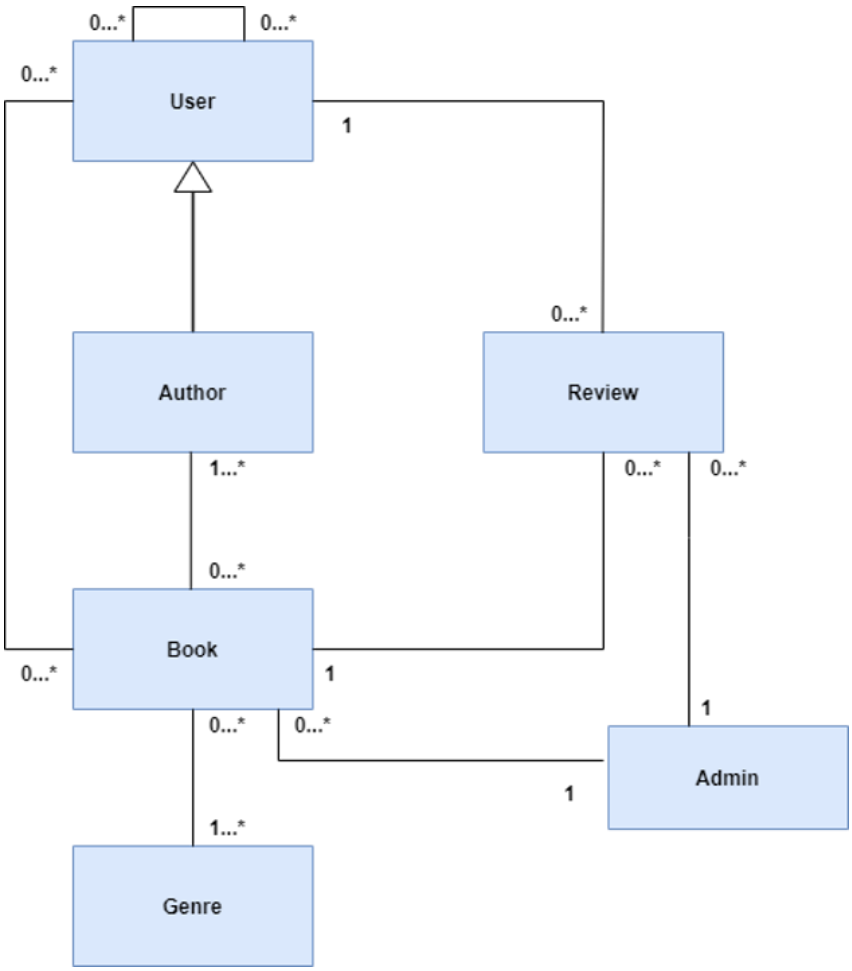
The first one is a plain text dataset of Amazon product. From it we extract information about books purchased and reviewed by users. We wrote a program (GitHub) to reorganize data in a json format and at the same time scrape some missing information. The second is a collection of data from goodreads site, it is already formatted in json and from it we extract similar information of the first one.

The reviews of two dataset were merged in case we found the same ISBN. The two datasets were quite big, so we decided to take from goodreads the books from 2013 to 2017 that have reviews. Some information about authors and users were generated randomly using java-faker. We also used the UUID in order to create a different unique identifier for new entity that can been created during the execution of the application.

Use Case Diagram



Class Analysis



Classes Relationship

1. Each User can create more than 1 Review for each Book
2. One Review is associated only to 1 User and 1 Book
3. An Author can publish more than 1 Book
4. A Book has 1 or many Authors
5. An Admin is connected to a Book or a Review if it has been reported
6. A Book ha 1 or many Genres
7. A User can follow 0 or many users
8. A User can have 0 or many books into his/her books list

Classes Definitions

Class	Description
User	A standard user who can interact with most of its functionalities
Author	A special user who can add new books
Admin	A user who has the task of checking the correct behavior of the users described above
Book	Contains information about a book published by an author
Review	Contains information about reviews left by users and authors
Genre	The genre of books

Classes Attributes

User		
Attribute	Type	Description
id	String	Unique string that identifies a user to match the review
name	String	User's name
nickname	String	Unique string that identifies a user in the login
email	String	User's e-mail
password	String	User's password in hash
interactions	Interactions	Contains the list of followers and followers of the user
listReviewID	ArrayList<String>	List of the ids of the reviews to which the likes have been put
follower_count	Integer	User's follower counter

statistics	ArrayList<Genre>	Average of reviews left by the user by genre
listBook	ListBooks	Contains the list of “read” and “to read” of the user

Author		
Attribute	Type	Description
same as the superclass		
published	ArrayList<Book>	List of books published by the authors with all the appropriate characteristics

Book		
Attribute	Type	Description
isbn	String	Unique string that identifies a book (sometimes missing)
language_code	String	Defines the language of the book
asin	String	Unique string that identifies a book (sometimes missing)
average_rating	Double	Average rating of reviews
description	String	Book’s description
num_pages	Integer	Book’s pages
publication_day	Integer	Publication day of the book
publication_month	Integer	Publication m month of the book
publication_year	Integer	Publication year of the book
image_url	String	Book cover
book_id	String	Unique string that identifies a user to be associated with the review and authors
ratings_count	Integer	Total reviews
title	String	Book title
authors	ArrayList<Author>	List of all possible authors and co-authors of a book
genres	ArrayList<String>	List of all possible genres of a book

reviews	ArrayList<Review>	List of all possible reviews of a book

Review		
Attribute	Type	Description
review_id	String	Review id
date_update	String	Date the review was last modified
user_id	String	User id
username	String	Username id
rating	String	Rating of the review
review_text	String	Text of the reviews
likes	Integer	Total number of likes for a review
liked	Boolean	It is set to true when the review id is present in the user's listReviewID attribute

Genre		
Attribute	Type	Description
type	String	Type of the genre
value	Double	Used for statistics

Query Analysis

Volume Table

Name	Type	Number of instances	Explanations
Book	Entity	275232	According to some online statistics in the 2013 the total amount of published books in US was 275232
Author	Entity	96008	In Goodreads dataset we have a total of 96008 authors that published during 2013
Review	Entity	1466018	In Goodreads dataset we have a total of 163752 books published in that year and a total of 15,7M of review in 18 years so if we decide

			to maintain this proportion we have that 163752 : 275232 (books) = 872222 : x (reviews) so 1466018 number of reviews
User	Entity	43450	Again, considering Goodreads we have a total number of users of 465323 registered in 18 years. Therefore 872222 : 1466018 (reviews) = 25851 : x (users) so 43450 number of users
Read / To-read	Relationship	24	An average reader can read 12 books per year
Follow / Following	Relationship	300	Very difficult to predict, but some online statistics says that it could be around 150

MongoDB

Read Operation		
Query	Frequency	Cost
Browse all books	Low (users/authors don't have much interest to find all books without any filter)	Very High (multiple reads) For instance, at the end of the first year we will have 275232 of books
Browse books by title	High (users/authors are usually interested to find a specific book)	Average (if you don't insert the complete title you will retrieve more than 1 book)
Browse books by genre	Average (users/authors are usually interested to find book that belongs to a genre which he/she likes)	High (multiple reads) For instance, if we consider 8 macro categories of books we have 275232 / 8 = 34404 books
Browse books by title and genre	Average (users/authors can easily find a book knowing it's category and a part or the complete title)	Low (if there aren't any other books with the same title, we have only 1 read)
Browse User/Author by name	High (users/authors usually search each other for following or for view other's profile)	Low (considering the full name we might find homonyms but the

		number of reads are quite low)
View book information and related review	High (each time a user/author search for a book, he/she probably also wants to learn more about it)	Average-High (multiple reads) We have to retrieve the reviews of a books. If we take in consideration the instances of a year we have 1466018/ 275232 = 5 reviews for each books
View user/author profile	Average-High (each time a user/author search for a another user, he/she probably also wants to learn more about him/her)	Low (1 read) We can easily find in all the information related to a user after the browse
Ranks users with more frequent valid reviews based on the likes received	Average	High (complex aggregations)
View for each author, average rating on each published category	High (these information are retrieved each time we wants to see a profile of an author)	High (complex aggregations)
View for each user, average rating given to each category	High (these information are retrieved each time we wants to see a profile of a user)	High (complex aggregations)
View for each year and category the number of published books	Low-Average	High (complex aggregations)

Write Operation		
Query	Frequency	Cost
Insert a Book	Low-Average 275232/365 = 754 books every day	Average (create new document)
Insert a User	Low 43450/365 = 119 users every day	Low (create new document)
Insert an Author	Low 96008/365 = 263 authors every day	Low (create new document)
Insert a Review	High 1466018/365 = 4017 reviews every day	Low (add new document)
Modify a Review	Low	Low (modify a document)
Delete Review	Low	Low (delete a document)
Add like to Review	Low-Average	Low (modify 2 documents)

Remove like from Review	Very Low	Low (modify 2 documents)
Delete Book	Very Low	Low (delete a document)
Delete User	Low	High (delete a document and related document reviews)
Delete Author	Very Low	Very High (delete a document and related books with its reviews)
Report a Review	Very Low	Low (add new document)
Report a Book	Very Low	Low (add new document)
Delete reported Review/Book	Average	Low-Average (2 documents deleted and 1 added)
Unreport Review/Book	Average	Low (1 documents deleted and 1 added)
Restore Log	Very Low	Low (1 document deleted and 1 added)
Insert an Admin	Very Low	Low (add new document)

Neo4J

Read Operation		
Query	Frequency	Cost
Browse Follower	Average-High (every time you visit a User/Author profile)	Average (multiple reads)
Browse Following	Average-High (every time you visit a User/Author profile)	Average (multiple reads)
Browse Published Books	Low-Average (every time you want to see the published list)	Average (multiple reads)
Browse Read Books	Low-Average (every time you want to see the read list)	Average (multiple reads)
Browse To Read Books	Low-Average (every time you want to see the to-read list)	Average (multiple reads)
View Suggested User	High	High

	(each time you visit a User/Author homepage)	(multiple reads)
View Suggested Author	High (each time you visit a User/Author/Book page)	High (multiple reads)
View Suggested Books	High (each time you visit a Book page)	High (multiple reads)
Browse Most Popular/ Desired Books	Low (every time you want to see ranking statistics)	High (multiple reads)

Write Operation		
Query	Frequency	Cost
Insert a Book	Low-Average	Low-Average (insert 1 node and 1 ore more WROTE relationship)
Insert a User	Low	Low (insert 1 node)
Insert an Author	Low	Low (insert 1 node)
Delete Book	Very Low	Low (delete 1 node)
Delete User	Low	Low (delete 1 node)
Delete Author	Very Low	Low (delete 1 node and detach it's related books)
Add Follow	Average-High	Low (add new relationship)
Remove Follow	Very Low	Low (delete relationship)
Add to read/read list	Average	Low (add new relationship)
Remove from to read/read list	Very Low	Low (delete relationship)
Restore Book	Very Low	Low-Average (insert 1 node and 1 ore more WROTE relationship)

Data Model

This chapter discusses organizing databases. MongoDB and Neo4j were used for the design.

MongoDB – Document Organization

In mongoDB we are stored five different collections:

- **Authors Collection:** ~ 59.6k
- **Users Collection:** ~ 151.0k
- **Books Collection:** ~ 131.1k
- **Genres Collection:** 38
- **Report Collection**
- **Log Collection**

Authors Collection


```
{
  "_id": {
    "$oid": "61e18e83d02c1399618e2a55"
  },
  "password": "D1B9446EF187CE708F9D2DF4A83359C5",
  "name": "Bernard Knight",
  "author_id": "37778",
  "email": "kristofer.mayert@unipi.it",
  "username": "kristofer.mayert",
  "follower_count": 3,
  "liked_review": [
    ""
  ]
}
```


It contains the fundamental information of each author.

Users Collection


```
{
  "_id": {
    "$oid": "61e18e6b76de9638c596445e"
  },
  "password": "4FC9618AD8C4CFB7EFB9525402CF7E1C",
  "user_id": "A3MTOVLB0TSQXL",
  "name": "Dylan Kuphal II",
  "email": "tawana.fisher1@unipi.it",
  "username": "tawana.fisher1",
  "liked_review": [
    ""
  ],
  "follower_count": 0
}
```


It contains the fundamental information of each user.

Books Collection


```
{
    "_id": {
        "$oid": "61e844dc8a28488dd721f550"
    },
    "isbn": "0001713353",
    "language_code": "eng",
    "description": "A king runs into the trouble when he tries to protect his cheese from the palace mice.",
    "num_pages": 64,
    "publication_day": 1,
    "publication_month": 3,
    "publication_year": 1986,
    "image_url": "https://s.gr-assets.com/assets/nophoto/book/111x148-bcc042a9c91a29c1d680899eff700a03.png",
    "book_id": "446759",
    "title": "The King, the Mice and the Cheese",
    "average_rating": 4,
    "ratings_count": 1,
    "genres": [
        "Children"
    ],
    "authors": [
        {
            "author_id": "341748",
            "author_name": "Eric Gurney"
        },
        {
            "author_id": "251091",
            "author_name": "Nancy Gurney"
        }
    ],
    "reviews": [
        {
            "user_id": "9eb40f941680c78a10f10b15c5fe0ec2",
            "review_id": "b130642005d9cc0159b3b1b80e89dd5d",
            "rating": 4,
            "review_text": "Loved by my children and myself. A simple story told with fun and accompanied by
            "date_updated": {
                "$date": "2013-12-29T22:52:01.000Z"
            },
            "likes": 0,
            "username": "anderson.schaden"
        }
    ]
}
```


Contains the basic information of each book where some fields can be omitted. The book id, the title and at least one author are certainly required.

Genres Collection


```
{
  "_id": "Crime"
},{
  "_id": "Mystery"
},{
  "_id": "Romance"
},{
  "_id": "Graphic"
},{
  "_id": "Children"
},{
  "_id": "Fantasy"
},{
  "_id": "Horror"
},{
  "_id": "Parenting & Families"
},{
  "_id": "Science"
},{
  "_id": "Arts & Photography"
},{
  "_id": "Biography"
},{
  "_id": "Reference"
},{
  "_id": "Home & Garden"
},{
  "_id": "Biographies & Memoirs"
},{
...
}
```


It holds all possible genres that can be used for books. It is the only static connection, as it cannot be modified by the application.

Report Collection


```
[{
  "_id": {
    "$oid": "61fabb6b5883c545819f50fa"
  },
  "report_id": "9c69a794-3bb5-4970-928e-d949f7926153",
  "type": "review",
  "review_id": "8e7767db6ad37d8039737ec91acd54b2",
  "review_text": "The story is set in 1953 and the main character is an
  "user_id": "13fd7d3894016d252573ad254df6adc3",
  "username": "gary.hyatt1",
  "book_id": "2328359"
}]
```


It includes all the reports made by users that will be viewed by the admin

Log Collection


```
{
  "_id": {
    "$oid": "6203f66fb78dd61d9cd39a94"
  },
  "id": "9931d9c3-90dd-4797-89b8-cf6737008900",
  "report_id": "e68038e7-3335-4513-a8c0-826c4a63fa23",
  "date": {
    "$date": "2022-02-09T17:14:23.899Z"
  },
  "operation": "unreport",
  "admin": "admin",
  "type": "review",
  "review_id": "5a114aff4c0047b4c79203fd64d8221b",
  "review_text": "Synopsis: \n For fifteen years Luke Moore has lived by thr
lovers want to bind him tighter? \n My Thoughts: \n I enjoyed this one. It w
itically. I knew what the deal was with him and that kid from his past from
atthew. Those two are really good guys, but Richard's controlling side was a
  "rating": "3",
  "user_id": "3b8e93a8e7b1e70cb62e04edb0cffcb3",
  "username": "marlon.mcdermott1",
  "book_id": "8709531"
}
```

```
{
  "_id": {
    "$oid": "6203f67cb78dd61d9cd39a98"
  },
  "id": "f1812062-e6b8-4a63-9619-e06c76d1d084",
  "report_id": "8b94ff0c-28a5-4bb0-9a08-37a45b0de5e2",
  "date": {
    "$date": "2022-02-09T17:14:36.332Z"
  },
  "operation": "unreport",
  "admin": "admin",
  "type": "book",
  "isbn": "",
  "asin": "B003N3V61S",
  "num_pages": 308,
  "publication_day": 23,
  "publication_month": 3,
  "publication_year": 2010,
  "image_url": "https://s.gr-assets.com/assets/nophoto/book/111x148-bcc042a9c91
  "book_id": "8709531",
  "title": "More (More #1)",
  "description": "For fifteen years Luke Moore has lived by three rules: stay c
ind him tighter?",
  "authors": [
    {
      "author_name": "Ralph Cosham",
      "author_username": "",
      "author_id": "105442"
    },
    {
      "author_name": "Richard Adams",
      "author_username": "",
      "author_id": "7717"
    },
    {
      "author_name": "Sloan Parker",
      "author_username": "",
      "author_id": "3413928"
    }
  ],
  "genres": [
    "Romance"
  ]
}
```


It allows admins to retrieve reviews and books that they or other admins have deleted.

Neo4J – Nodes Organization

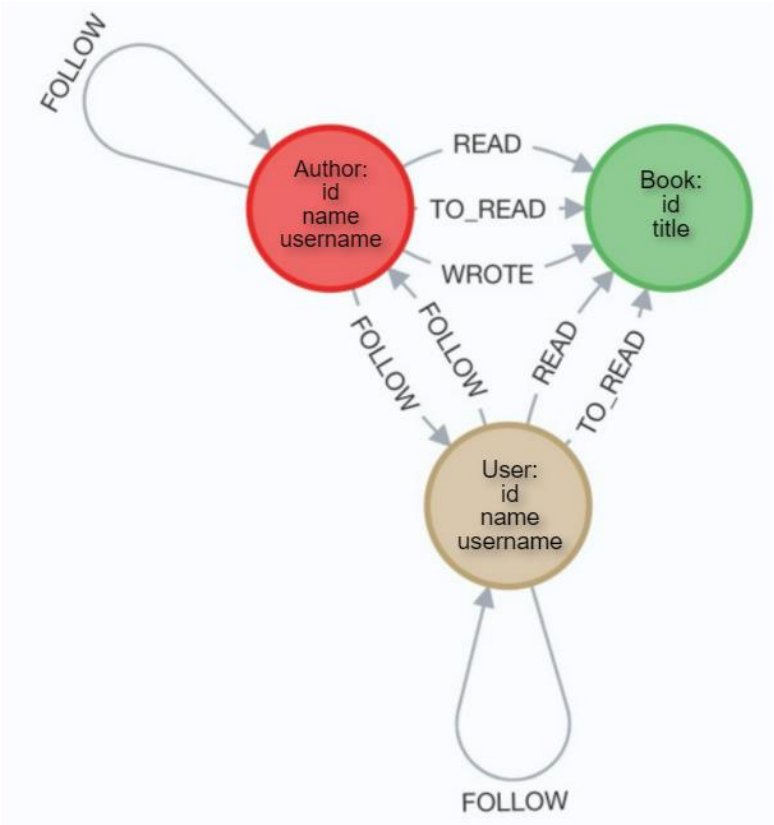
In neo4j there are ~ 340.000 nodes, ~ 240.000 relationships.

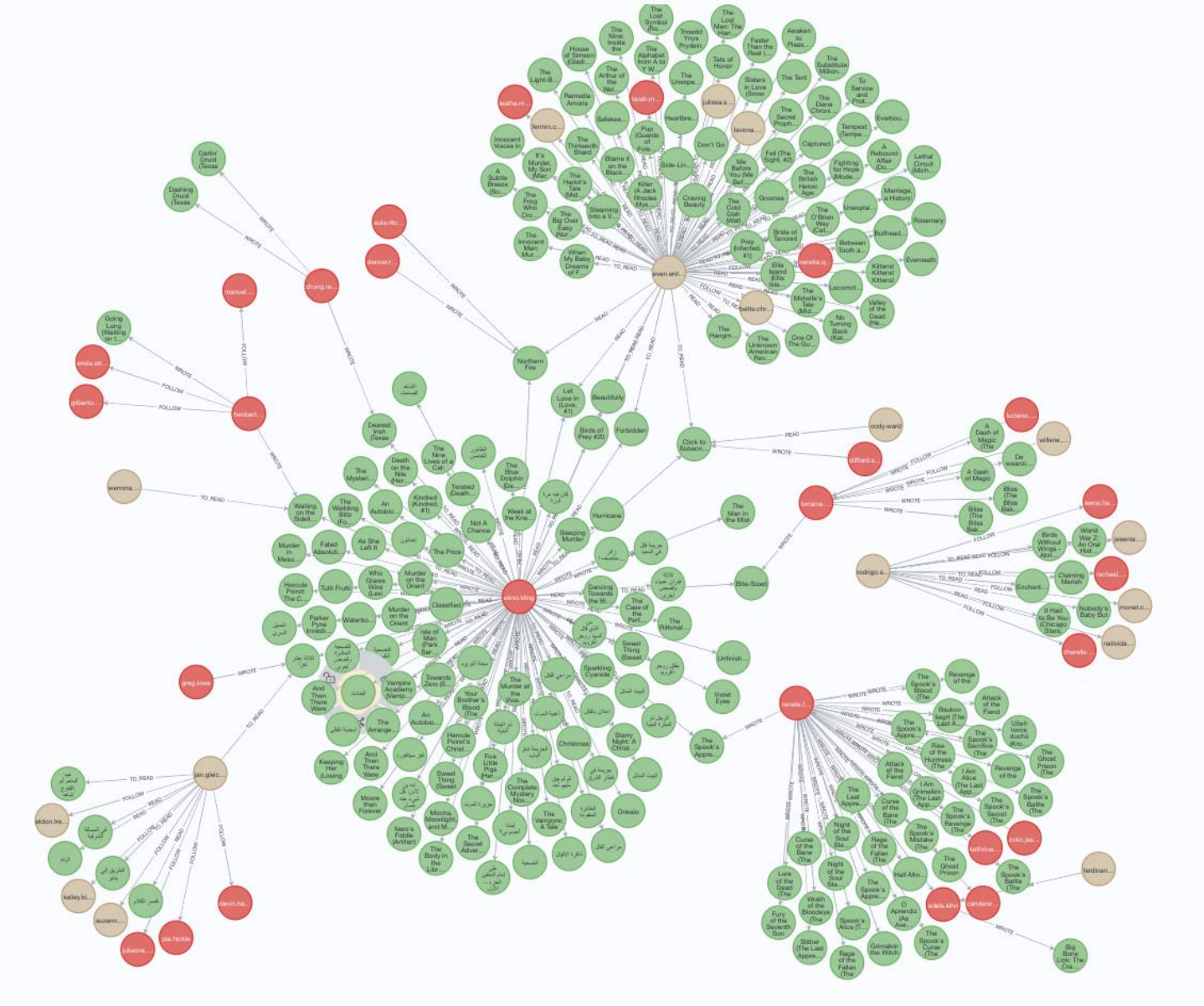
The structure of the nodes with their relations is described below:

- The "**User**" nodes represent the registered user and contain the user's id, username and name
- The "**Author**" nodes represent the registered user and contain the author's id, username and name
- The "**Book**" nodes represent the registered user and contain the book's id and title

Relationships:

- **User →: FOLLOW → User**, a User following a User who can be added and removed from the application. This relationship has no attributes.
- **User →: FOLLOW → Author**, a User following an Author who can be added and removed from the application. This relationship has no attributes.
- **Author→: FOLLOW → User**, an Author following a User who can be added and removed from the application. This relationship has no attributes.
- **Author→: FOLLOW → Author**, an Author following an Author who can be added and removed from the application. This relationship has no attributes.
- **User->: TO_READ -> Book**, represents a User who wants to read a book.
- **Author->: TO_READ -> Book**, represents an Author who wants to read a book.
- **User->: READ -> Book**, represents a User who has read wants to read a book.
- **Author->: READ -> Book**, represents an Author who has read wants to read a book.
- **Author ->: WROTE ->Book,** represents an author who has written a book.





Implementation

The Reviook application is composed by four main package that guarantee order and code organization.

- **Package Manager:** this package is in charge of managing the main crud operations to communicate with the database.
- **Package Controllers:** this package is in charge of providing the methods that manage graphical interfaces and internal navigation of the application.
- **Package Components:** this is the package used for building custom elements needed for the GUI.
- **Package Entity:** this package contains all objects used to structure the application.

In detail:

UserManager contains all the java methods used in the User or Author experience, this includes the crud operations to interact with other users.

BookManager contains all the java methods used in the exploration of the collection of books.

SearchManager contains specific methods used in the browsing experience from the Reviook search GUI.

AdminManager contains specific methods used by admins to moderate the application.

AuthorInterfaceController, UserInterfaceController and AdminController contains methods that manage the interaction between user and the GUI concerning authors, users and admin.

BookDetailController, AddBookController, DialogNewReviewController and PreviewReviewController contains methods that are used to handle everything related to books in the GUI.

LoginController, UpdateController, RegisterController and AddAdminController contains methods that are used in the user experience to handle operations related to sign-in and sign-up.

SearchInterfaceController contains methods used in the GUI to search users, authors or books.

Cache

In this application we immediately faced the problem of the slowness of crud operations. To solve this problem and make navigation in the application faster, we have decided to implement a sort of cache for our data.

In this way the logged user will not have to wait every time for the data to be loaded from the database because if the information is cached, the display will be instantaneous.

With this feature we solved the slow response time in Reviook, but this led to inconsistency problems since Reviook is a multi-user application.

To avoid this issue, we use a cache object into each session, this object contains the usual data with an added timestamp that will invalidate the cache if too old.

JAVA Entity

Class	Description
User	User registered in the application
Author	Author registered in the application
Book	A Book added in the application

Review	A review added in the application
Log	Entity used to keep track of admin operation
Report	Entity used to allow to report books or reviews
Cache	Entity used to speed up the application

```
 public class User {
    private String id;
    private String name;
    private String nickname;
    private String email;
    private String password;
    private Interaction interactions;
    private ArrayList<String> listReviewID;
    private Integer follower_count;
    private ArrayList<Genre> statistics;
    private ListBooks listBooks;
    /* Constructors, Getters and Setters */
 }


public class Author extends User {
    private ArrayList<Book> published;
    /* Constructors, Getters and Setters */
}


public class Book {
    private String isbn;
    private String language_code;
    private String asin;
    private Double average_rating;
    private String description;
    private Integer num_pages;
    private Integer publication_day;
    private Integer publication_month;
    private Integer publication_year;
    private String image_url;
    private String book_id;
    private Integer ratings_count;
    private String title;
    private ArrayList<Author> authors;
    private ArrayList<String> genres;
    private ArrayList<Review> reviews;
    /* Constructors, Getters and Setters */
}


public class Review {
    private String review_id;
    private String date_update;
    private String user_id;
    private String username;
    private String rating;
    private String review_text;
```

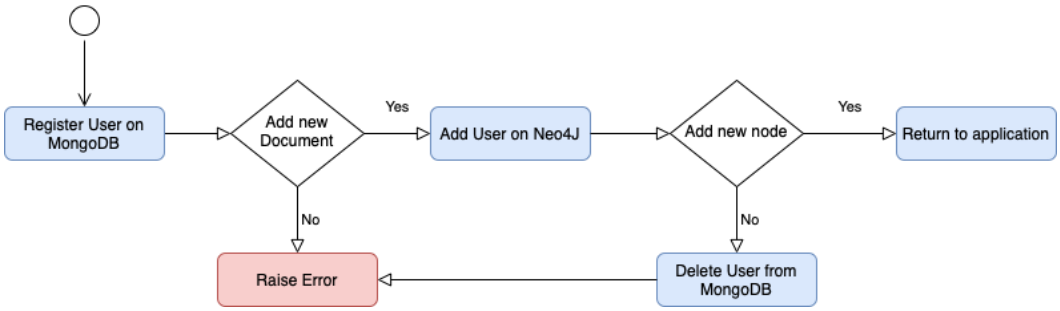
```
    private Integer likes;
    private Boolean liked;
    /* Constructors, Getters and Setters */
}


public class Log extends Report {
    private String id;
    private String operation;
    private String admin;
    private Date date;
    /* Constructors, Getters and Setters */
}

public class Report {
    private String report_id;
    private String type;
    private String isbn;
    private String asin;
    private String book_id;
    private String title;
    private String description;
    private Integer num_pages;
    private Integer publication_day;
    private Integer publication_month;
    private Integer publication_year;
    private String image_url;
    private ArrayList<Author> authors;
    private ArrayList<String> genres;
    private String review_id;
    private String review_text;
    private String rating;
    private String user_id;
    private String username;
    /* Constructors, Getters and Setters */
}

public class Cache {
    private ArrayList<Book> searchedBooks;
    private ArrayList<User> searchedUsers;
    private ArrayList<Author> searchedAuthors;
    private String searchedTitle;
    private String searchedGenres;
    private String searchType;
    private Date lastUpdate;
    private int count;
    /* Constructors, Getters and Setters */
}
```

Database Consistency Management

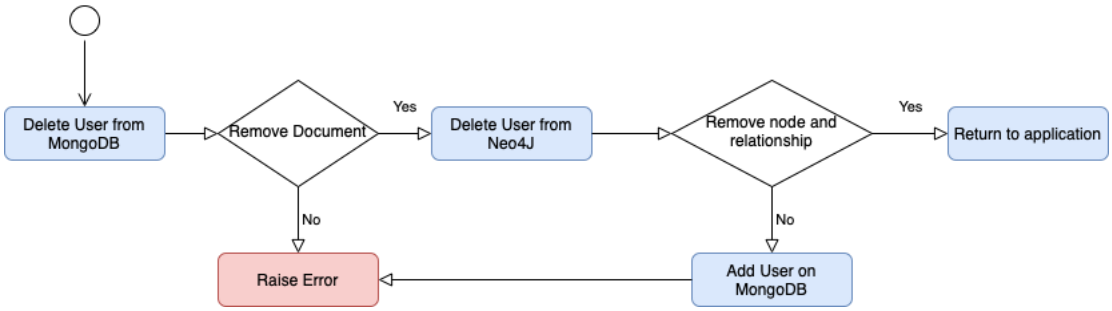
Some of the information contained into the two databases are duplicated. Some of the CRUD operations have been organized as follows, to maintain a state of consistency between the documents in MongoDB and the nodes/relationships in Neo4J.

The first write is committed on MongoDB, if it fails an exception is raised, otherwise the same operation is done on Neo4J. If the result of this last operation is positive the CRUD operation returns the control to the main program, otherwise it will try to undo the first operation on MongoDB.

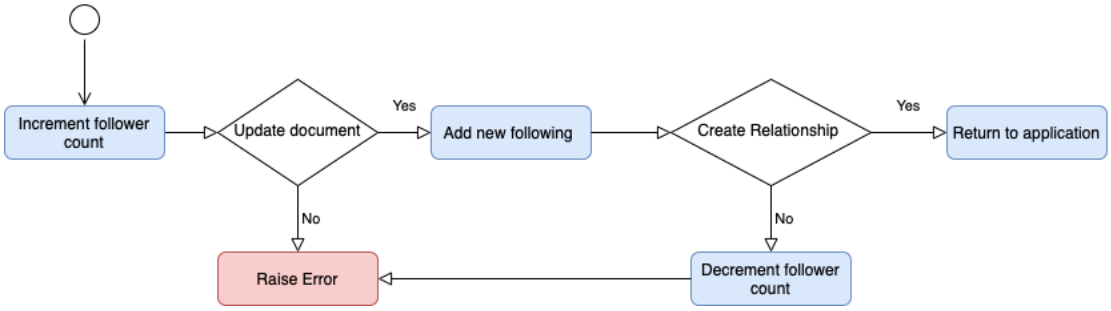
- Add new User/Author



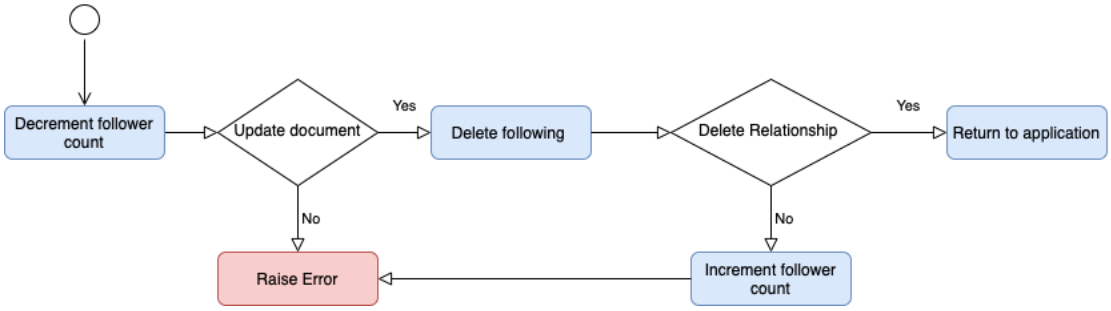
- Delete User/Author



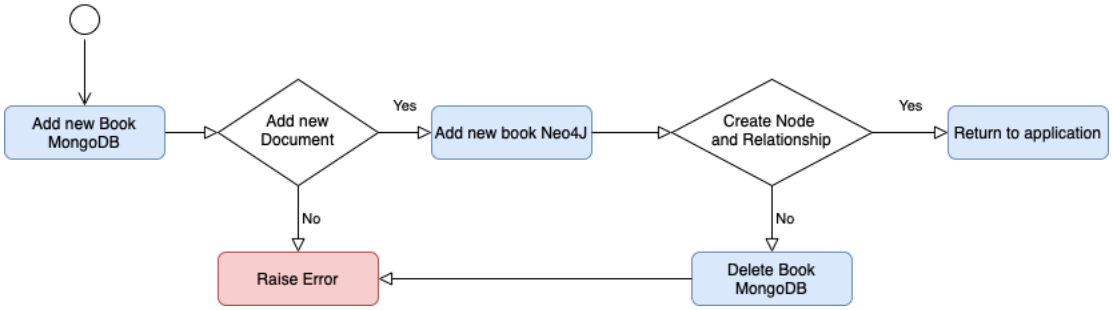
- Add Following



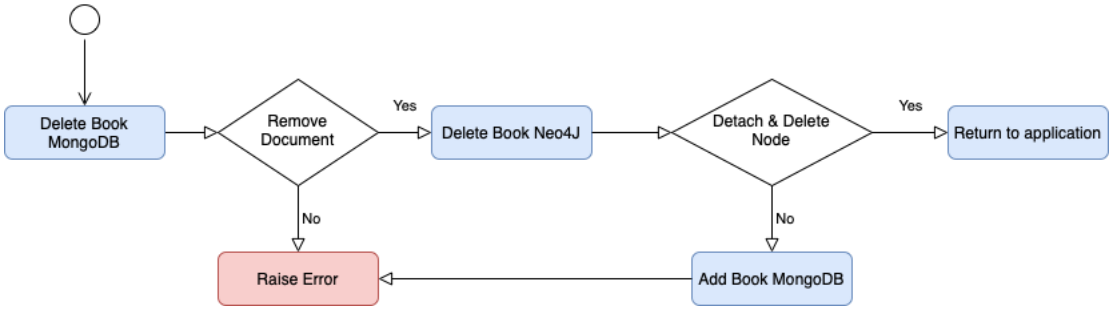
- Delete Following



- Add new Book



- Delete Book



Database Organization on Machines

We set up a MongoDB cluster to avoid the problem of the single point of failure and to improve the availability of our service. The cluster is composed by three replicas set hosted in different servers. In each of them, the mongod daemon runs at the same port, the primary server receives the client requests and manage communication into the cluster, on the other side the secondary servers maintain the replica updated to be ready to substitute the primary when it will crash.

As follows the organization of the virtual machines

VM	IP Address & Port	OS
Replica-0	172.16.4.102:27020	Ubuntu 18.04.2 LTS
Replica-1	172.16.4.103:27020	Ubuntu 18.04.2 LTS
Replica-2	172.16.4.104:27020	Ubuntu 18.04.2 LTS

Replica Configuration

This is our replica configuration performed on one of the three Servers

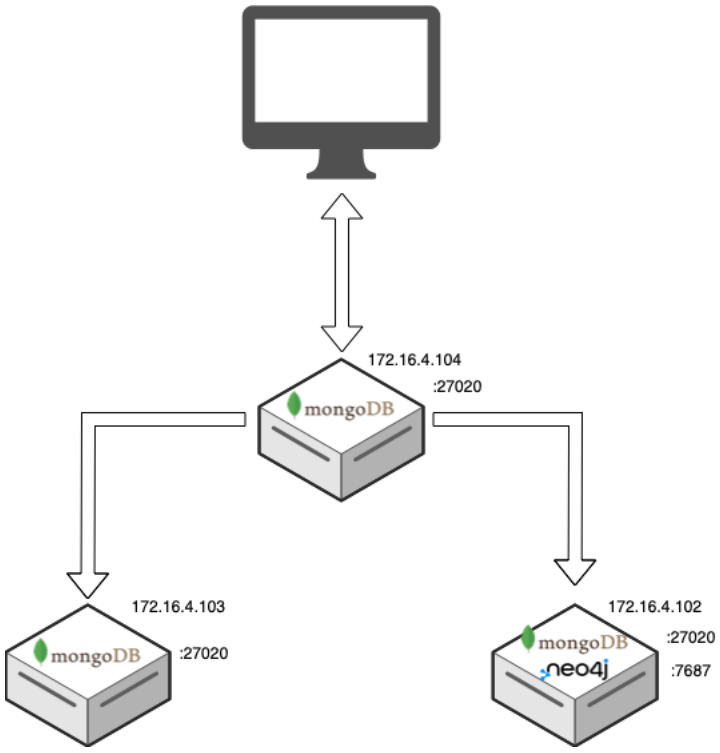
```
rsconf1 = {_id: "reviook",
        members: [
            {_id: 0, host: "172.16.4.102:27020", priority: 1
            },
            {_id: 1, host: "172.16.4.103:27020", priority: 2
            },
            {_id: 2, host: "172.16.4.104:27020", priority: 5
            }
        ]
};
```

The replica-2 has the highest priority, so it will be the primary. Our application is mainly read oriented as we discover in the query analysis section, so we decided to make some other configurations on the application sides to speed up writes and reads to the detriment of the consistency. For instance, we preferred to read from the nearest replica (the one which has the lowest latency) and wait for the writing acknowledgement from only one replica.

```
uri = new
ConnectionString("mongodb://172.16.4.102:27020,172.16.4.103:27020,172.16.4.104:2
7020/");
MongoClientSettings msc = MongoClientSettings.builder()
        .applyConnectionString(uri)
        .readPreference(ReadPreference.nearest())
```

```
        .retryWrites(true)
        .writeConcern(WriteConcern.W1).build();
```

Replica crash



If the primary crashes one of the secondaries will be elected as the new primary. We assigned priority on each replica to control the behavior of the election algorithm. For this reason, we decided to set up our Neo4J server on the Replica-0 that has the lowest priority to become the new primary, in this way we should have a better load balance on each machine.

At the start we have this configuration:

Primary	Secondary	Secondary
172.16.4.104:27020	172.16.4.103:27020	172.16.4.102:27020

If the Primary crashes, we will have the following new configuration:

-	Primary	Secondary
172.16.4.104:27020	172.16.4.103:27020	172.16.4.102:27020

Neo4J CRUD Operations

Create

- Add new user or author

```
public boolean addNewUsers(User user, String type) {
    boolean result;
    try (Session session = nd.getDriver().session()) {
        result = session.writeTransaction((TransactionWork<Boolean>) tx -> {
            tx.run("CREATE (ee:" + type + " { id: $id,  name: $name,
username: $username})", parameters("id", user.getId(), "name",
user.getName(), "username", user.getNickname()));
            return true;
        });
    }
    return result;
}
```

- Create follow relationship from username1 to username2

```
public boolean following(String username1, String type1, String username2,
String type2) {
    boolean result = false;
    if (incrementFollowerCount(username2)) {
        try (Session session = nd.getDriver().session()) {
            result = session.writeTransaction((TransactionWork<Boolean>) tx -
> {
   tx.run("MATCH (n:" + type1 + "),(nn:" + type2 + ") WHERE n.username ='"

+ username1 + "' AND nn.username='"

+ username2 + "'" + "CREATE (n)-[:FOLLOW]->(nn)");
                return true;
            });
        }
        if (result != true)
            decrementFollowerCount(username2);
    }
    return result;
}
```

- Create if not exists to_read relationship from username to book_id

```
public boolean toReadAdd(String type, String username, String book_id) {
    boolean result = false;
    try (Session session = nd.getDriver().session()) {
```

```
        result = session.writeTransaction((TransactionWork<Boolean>) tx -> {
            tx.run("MATCH (n:" + type + "),(nn:Book) WHERE n.username ='"
+ username + "' AND nn.id='" + book_id + "'" +
                    "MERGE (n)-[:TO_READ]->(nn)");
            return true;
        });
    }
    return result;
}
```

- Create if not exists read relationship from username to book_id

```
public boolean readAdd(String type, String username, String book_id) {
    boolean result = false;
    try (Session session = nd.getDriver().session()) {
        result = session.writeTransaction((TransactionWork<Boolean>) tx -> {
            tx.run("MATCH (n:" + type + "),(nn:Book) WHERE n.username ='"
+ username + "' AND nn.id='" + book_id + "'" +
                    "MERGE (n)-[:READ]->(nn)");
            return true;
        });
    }
    return result;
}
```

- Create Book

```
public boolean addBookN4J(Book newBook) {
    Boolean result;
    try (Session session = nd.getDriver().session()) {
        result = session.writeTransaction((TransactionWork<Boolean>) tx -> {
            tx.run("CREATE (ee: Book { id : $id, title: $title})",
parameters("id", newBook.getBook_id(), "title", newBook.getTitle()));
            for (int i = 0; i < newBook.getAuthors().size(); i++) {
                tx.run("MATCH (dd:Author),(ee: Book) WHERE dd.id = '" +
newBook.getAuthors().get(i).getId() + "' AND ee.id='" + newBook.getBook_id()
+ "'" + "CREATE (dd)-[:WROTE]->(ee)");
            }
            return true;
        });
    }
    return result;
}
```

Read

- Read read/to_read/wrote relationship from username to book_id

```
public ArrayList<Book> loadRelationsBook(String type, String username, String
read) {
    ArrayList<Book> readings = new ArrayList<Book>();
    ArrayList<Book> books = new ArrayList<>();
    try (Session session = nd.getDriver().session()) {
        readings = session.readTransaction((TransactionWork<ArrayList<Book>>)
tx -> {
            Result result = tx.run("MATCH (ee:" + type + ")-[:" + read + "]-
>(book) where ee.username = '" + username + "' " +
                    "return book.title, book.id");
            while (result.hasNext()) {
                Record r = result.next();
                books.add(new Book(((Record) r).get("book.title").asString(),
((Record) r).get("book.id").asString()));
            }
            return books;
        });
    }
    return readings;
}
```

- Read following relationship from username

```
public List<String> loadRelationsFollowing(String type, String username) {
    List<String> relationship = new ArrayList<>();
    try (Session session = nd.getDriver().session()) {
        relationship =
session.readTransaction((TransactionWork<List<String>>) tx -> {
            Result result = tx.run("MATCH (ee:" + type + ")-[:FOLLOW]-
>(friends) where ee.username = '" + username + "' " +
                    "return friends.username as Friends");
            ArrayList<String> following = new ArrayList<>();
            while (result.hasNext()) {
                Record r = result.next();
                following.add(((Record) r).get("Friends").asString());
            }
            return following;
        });
    }
    return relationship;
}
```

- Read follower relationship from username

```
public List<String> loadRelationsFollower(String type, String username) {
    List<String> relationship = new ArrayList<>();
    try (Session session = nd.getDriver().session()) {
```

```
            relationship =
    session.readTransaction((TransactionWork<List<String>>) tx -> {
                Result result = tx.run("MATCH (ee:" + type + ")<-[:FOLLOW]-
    (friends) where ee.username = '" + username + "' " +
                        "return friends.username as Friends");
                ArrayList<String> follower = new ArrayList<>();
                while (result.hasNext()) {
                    Record r = result.next();
                    follower.add(((Record) r).get("Friends").asString());
                }
                return follower;
            });
        }
        return relationship;
    }
```

Delete

- Delete user or author

```
public boolean deleteUserN4J(User user, String type) {
    boolean result;
    String t = type.equals("author") ? "Author" : "User";
    try (Session session = nd.getDriver().session()) {
        result = session.writeTransaction((TransactionWork<Boolean>) tx -> {
            tx.run("MATCH (n : " + t + " { username: '"

+ user.getNickname() + "'}) DETACH DELETE n");
            return true;
        });
    }
    return result;
}
```

- Delete follow relationship from username1 to username2

```
public boolean deleteFollowing(String username1, String type1, String
username2, String type2) {
    boolean result = false;
    if (decrementFollowerCount(username2)) {
        try (Session session = nd.getDriver().session()) {
            result = session.writeTransaction((TransactionWork<Boolean>)
tx -> {
           tx.run("MATCH (n:" + type1 + " { username: '"

+ username1 + "' })-[r:FOLLOW]-> " +
            "(c :" + type2 + " { username: '"

+ username2 + "' })" +
                        "DELETE r");
                return true;
```

```
            });
        }
        if (result != true)
            incrementFollowerCount(username2);
    }
    return result;
}
```

- Delete to_read/read relationship from user/author

```
public boolean removeBookFromList(String idBook, String Relation, String
username, String Type) {
    try (Session session = nd.getDriver().session()) {
        session.writeTransaction((TransactionWork<Void>) tx -> {
            tx.run("MATCH (n:" + Type + "{username: '"

+ username + "' })-[r:" + Relation + "]->" +
                    "(c : Book{id: '" + idBook + "'}) " +
                    "DELETE r");
            return null;
        });
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

- Delete book

```
public boolean deleteBookN4J(Book book) {
    boolean result;
    try (Session session = nd.getDriver().session()) {
        result = session.writeTransaction((TransactionWork<Boolean>) tx -> {
            tx.run("MATCH (n : Book { id: '" + book.getBook_id() + "'})
DETACH DELETE n");
            return true;
        });
    }
    return result;
}
```

Crud operations MongoDB

Here we can find listed the crud operations performed by the application in the MongoDB.

- Create operations that allow us to manage books, author, users, reviews, logs and reports insertions:

```
public boolean addBookMongo(Book newBook) {
    InsertOneResult result = null;
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(new Date());

    ArrayList<DBObject> authorsObj = new ArrayList<>();
    for (Author a : newBook.getAuthors()) {
        DBObject author = new BasicDBObject();
        author.put("author_name", (String) a.getName());
        author.put("author_role", ""); // TODO da togliere
        author.put("author_id", (String) a.getId());
        authorsObj.add(author);
    }

    ArrayList<Review> reviews = new ArrayList<Review>();
    Document doc = new Document("language_code", newBook.getLanguage_code())
            .append("isbn", newBook.getIsbn())
            .append("description", newBook.getDescription())
            .append("num_pages", newBook.getNum_pages())
            .append("publication_day", newBook.getPublication_day())
            .append("publication_month", newBook.getPublication_month())
            .append("publication_year", newBook.getPublication_year())
            .append("image_url", newBook.getImage_url())
            .append("book_id", newBook.getBook_id())
            .append("title", newBook.getTitle())
            .append("average_rating", 0.0)
            .append("ratings_count", 0)
            .append("genres", newBook.getGenres())
            .append("authors", authorsObj)
            .append("reviews", newBook.getReviews());
    try {
        result = md.getCollection(bookCollection).insertOne(doc);
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (result != null)
        return result.wasAcknowledged();
    return false;
}

public boolean addReviewToBook(String reviewText, Integer ratingBook, String
book_id) {
    MongoCollection<Document> book = md.getCollection(bookCollection);
    Document newReview = new Document();
    String reviewID = UUID.randomUUID().toString();
    UpdateResult addReview, rateUpdated;
    LocalDateTime now = LocalDateTime.now();
    Date date = Date.from(now.atZone(ZoneId.systemDefault()).toInstant());
    newReview.append("date_updated", date);
    newReview.append("review_id", reviewID);
    newReview.append("likes", 0);
    newReview.append("rating", ratingBook);
    newReview.append("review_text", reviewText);
```

```
        if (session.getLoggedUser() != null) {
            newReview.append("user_id", session.getLoggedUser().getId());
            newReview.append("username", session.getLoggedUser().getNickname());
        } else {
            newReview.append("user_id", session.getLoggedAuthor().getId());
            newReview.append("username",
    session.getLoggedAuthor().getNickname());
        }
        Bson getBook = eq("book_id", book_id);
        DBObject elem = new BasicDBObject("reviews", new
    BasicDBObject(newReview));
        DBObject insertReview = new BasicDBObject("$push", elem);

        try {
            addReview = book.updateOne(getBook, (Bson) insertReview);

            if (addReview.getModifiedCount() == 1) {
                Book bookToUpdate = getBookByID(book_id);
                Double newRating = updateRating(bookToUpdate.getReviews());
                rateUpdated = book.updateOne(getBook,
    Updates.set("average_rating", newRating));
                if (rateUpdated.getModifiedCount() == 1) {
                    if (session.getLoggedUser() != null)
                        userManager.readAdd("User",
    session.getLoggedUser().getNickname(), book_id);
                    else
                        userManager.readAdd("Author",
    session.getLoggedAuthor().getNickname(), book_id);
                    return true;
                }
            }
        }catch (Exception e){
            e.printStackTrace();
        }
        return false;
    }

public boolean register(User user, String type) {
    ArrayList<String> liked_review = new ArrayList<>();
    InsertOneResult result = null;
    try {
        Document doc = new Document("name", user.getName())
                .append("password", user.getPassword())
                .append("follower_count", 0)
                .append("liked_review", liked_review)
                .append("email", user.getEmail())
                .append("username", user.getNickname());
        if (type.equals("Author")) {
            doc.append("author_id", user.getId());
            result = md.getCollection(authorCollection).insertOne(doc);
        } else {
            doc.append("user_id", user.getId());
            result = md.getCollection(usersCollection).insertOne(doc);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (result != null)
        return result.wasAcknowledged();
    return false;
}
```

```
public boolean addLog(Report report, String operation) {
    MongoCollection<Document> logs = md.getCollection("logs");
    LocalDateTime now = LocalDateTime.now();
    Date date = Date.from(now.atZone(ZoneId.systemDefault()).toInstant());
    if (report.getType().equals("book")) {
        ArrayList<DBObject> authorsList = new ArrayList<DBObject>();
        for (Author a : report.getAuthors()) {
            DBObject author = new BasicDBObject();
            author.put("author_name", (String) a.getName());
            author.put("author_username", (String) a.getNickname());
            author.put("author_id", (String) a.getId());
            authorsList.add(author);
        }
        Document newLog = new Document("id", UUID.randomUUID().toString())
                .append("report_id", report.getReport_id())
                .append("date", date)
                .append("operation", operation)
                .append("admin", session.getAdmin())
                .append("type", report.getType())
                .append("isbn", report.getIsbn() != null ? report.getIsbn() :
"")
                .append("asin", report.getAsin() != null ? report.getAsin() :
"")
                .append("num_pages", report.getNum_pages())
                .append("publication_day", report.getPublication_day())
                .append("publication_month", report.getPublication_month())
                .append("publication_year", report.getPublication_year())
                .append("image_url", report.getImage_url())
                .append("book_id", report.getBook_id())
                .append("title", report.getTitle())
                .append("description", report.getDescription())
                .append("authors", authorsList)
                .append("genres", report.getGenres());
        InsertOneResult res = logs.insertOne(newLog);
        if (!res.wasAcknowledged()) {
            return false;
        }
    } else if (report.getType().equals("review")) {
        Document newLog = new Document("id", UUID.randomUUID().toString())
                .append("report_id", report.getReport_id())
                .append("date", date)
                .append("operation", operation)
                .append("admin", session.getAdmin())
                .append("type", report.getType())
                .append("review_id", report.getReview_id())
                .append("review_text", report.getReview_text())
                .append("rating", report.getRating())
                .append("user_id", report.getUser_id())
                .append("username", report.getUsername())
                .append("book_id", report.getBook_id());
        InsertOneResult res = logs.insertOne(newLog);
        if (!res.wasAcknowledged()) {
            return false;
        }
    }
    return true;
}

public boolean addNewAdmin(String username, String password) {
    InsertOneResult result = null;
```

```
    try {
        Document doc = new Document("username", username).append("password",
password);
        result = md.getCollection(adminCollection).insertOne(doc);
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (result != null)
        return result.wasAcknowledged();
    return false;
}

public boolean reportBook(Book book) {
    MongoCollection<Document> reports = md.getCollection("reports");
    InsertOneResult result = null;
    try (MongoCursor<Document> cursor = reports.find(and(eq("book_id",
book.getBook_id()), eq("type", "book"))).iterator()) {
        if (!cursor.hasNext()) {
            ArrayList<DBObject> authorsList = new ArrayList<DBObject>();
            for (Author a : book.getAuthors()) {
                DBObject author = new BasicDBObject();
                author.put("author_name", (String) a.getName());
                author.put("author_username", (String) a.getNickname());
                author.put("author_id", (String) a.getId());
                authorsList.add(author);
            }
            Document newBook = new Document();
            newBook.append("report_id", UUID.randomUUID().toString());
            newBook.append("type", "book");
            newBook.append("isbn", book.getIsbn() != null ? book.getIsbn() :
"");
            newBook.append("asin", book.getAsin() != null ? book.getAsin() :
"");
            newBook.append("book_id", book.getBook_id());
            newBook.append("title", book.getTitle());
            newBook.append("description", book.getDescription());
            newBook.append("num_pages", book.getNum_pages());
            newBook.append("publication_day", book.getPublication_day());
            newBook.append("publication_month", book.getPublication_month());
            newBook.append("publication_year", book.getPublication_year());
            newBook.append("image_url", book.getImage_url());
            newBook.append("genres", book.getGenres());
            newBook.append("authors", authorsList);
            result = reports.insertOne(newBook);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (result != null)
        return result.wasAcknowledged();
    return false;
}

public boolean reportReview(Review review, String book_id) {
    MongoCollection<Document> reports = md.getCollection("reports");
    InsertOneResult result = null;
    try (MongoCursor<Document> cursor = reports.find(and(eq("review_id",
review.getReview_id()), eq("type", "review"))).iterator()) {
        if (!cursor.hasNext()) {
            Document newReview = new Document();
            newReview.append("report_id", UUID.randomUUID().toString());
```

```
            newReview.append("type", "review");
            newReview.append("review_id", review.getReview_id());
            newReview.append("review_text", review.getReview_text());
            newReview.append("rating", review.getRating());
            newReview.append("user_id", review.getUser_id());
            newReview.append("username", review.getUsername());
            newReview.append("book_id", book_id);
            result = reports.insertOne(newReview);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (result != null)
        return result.wasAcknowledged();
    return false;
}
```

- Read operations that allow us to get books, author, users, reviews, logs and view them:

```
public Book getBookByID(String book_id) {
    MongoCollection<Document> books = md.getCollection(bookCollection);
    Document book = new Document();
    try (MongoCursor<Document> cursor = books.find(eq("book_id",
book_id)).iterator()) {
        if (!cursor.hasNext()) {
            return null;
        }
        book = cursor.next();
    } catch (Exception e) {
        e.printStackTrace();
    }
    ArrayList<Author> authorsLis = new ArrayList<>();
    ArrayList<Review> reviewsList = new ArrayList<>();
    ArrayList<Document> authors = (ArrayList<Document>) book.get("authors");
    ArrayList<Document> reviews = (ArrayList<Document>) book.get("reviews");
    ArrayList<String> genres = (ArrayList<String>) book.get("genres");

    for (Document r : reviews) {
        String date;
        if(r.get("date_updated") == null) {
            if (r.get("date_added") == null) {
                date = null;
            }else{
                date = r.get("date_added").toString();
            }
        }else {
            date = r.get("date_updated").toString();
        }
        reviewsList.add(new Review(
                r.getString("username"),
                r.getString("review_id"),
                date,
                r.get("likes") == null ? r.getInteger("helpful") :
r.getInteger("likes"),
                r.getString("user_id"),
                r.get("rating").toString(),
                r.getString("review_text")
        ));
    }
```

```
    for (Document a : authors) {
        Author author = new Author(
                a.getString("author_id"),
                a.getString("author_name"),
                "",
                "",
                "",
                null,
                0
        );
        authorsLis.add(author);
    }

    Book outputBook = new Book(
            book.get("isbn") == null ? null : book.getString("isbn"),
            book.get("language_code") == null ? null :
book.getString("language_code"),
            book.get("asin") == null ? null : book.getString("asin"),
            book.get("average_rating").toString().equals("") ? Double.valueOf(0)
: Double.valueOf(book.get("average_rating").toString()),
            book.get("description") == null ? null :
book.getString("description"),
            book.get("num_pages") == null ? null :
book.getInteger("num_pages"),
            book.get("publication_day") == null ? null :
book.getInteger("publication_day"),
            book.get("publication_month") == null ? null :
book.getInteger("publication_month"),
            book.get("publication_year") == null ? null :
book.getInteger("publication_year"),
            book.get("image_url") == null ? null : book.getString("image_url"),
            book.getString("book_id"),
            book.getInteger("ratings_count"),
            book.getString("title"),
            authorsLis,
            genres,
            reviewsList
    );
    return outputBook;
}

public ArrayList<Book> searchBooks(String searchField, String genre) {
    MongoCollection<Document> books = md.getCollection(bookCollection);
    MongoCursor<Document> cursor = null;
    ArrayList<Book> result = new ArrayList<>();

    boolean titleSearch = true;
    boolean genresSearch = true;

    if (searchField == null || searchField.equals(""))
        titleSearch = false;
    if (genre == null || genre.equals(""))
        genresSearch = false;

    Bson titleFilter;
    Bson genreFilter;

    try {
        //global research
        if (!titleSearch && !genresSearch)
            cursor = books.find().iterator();
```

```
            //search by title
        else if (titleSearch && !genresSearch) {
            titleFilter = text("\"" + searchField + "\"", new
TextSearchOptions().caseSensitive(false));
            cursor = books.find(titleFilter).iterator();
        }
        //search by genre
        else if (!titleSearch && genresSearch) {
            genreFilter = in("genres", genre);
            cursor = books.find(genreFilter).iterator();
        }
        //search by title & genre
        else {
            titleFilter = match(text(searchField, new
TextSearchOptions().caseSensitive(false)));
            genreFilter = match(in("genres", genre));
            cursor = books.aggregate(Arrays.asList(titleFilter,
genreFilter)).iterator();
        }

        while (cursor.hasNext()) {
            Document book = cursor.next();
            //System.out.println("documento->" + document);

            ArrayList<Author> authorsLis = new ArrayList<>();
            ArrayList<Review> reviewsList = new ArrayList<>();
            ArrayList<Document> authors = (ArrayList<Document>)
book.get("authors");
            ArrayList<Document> reviews = (ArrayList<Document>)
book.get("reviews");
            ArrayList<String> genres = (ArrayList<String>) book.get("genres");

            for (Document r : reviews) {
                String date;
                if(r.get("date_updated") == null) {
                    if (r.get("date_added") == null) {
                        date = null;
                    }else{
                        date = r.get("date_added").toString();
                    }
                }else {
                    date = r.get("date_updated").toString();
                }
                reviewsList.add(new Review(
                        r.getString("username"),
                        r.getString("review_id"),
                        date,
                        r.get("likes") == null ? r.getInteger("helpful") :
r.getInteger("likes"),
                        r.getString("user_id"),
                        r.get("rating").toString(),
                        r.getString("review_text")
                ));
            }
            for (Document a : authors) {
                Author author = new Author(
                        a.getString("author_id"),
                        a.getString("author_name"),
                        "",
                        "",
                        "",
```

```
                        null,
                        0
                );
                authorsLis.add(author);
                //authorsLis.add(a.getString("author_name"));
            }

            result.add(new Book(
                    book.get("isbn") == null ? null : book.getString("isbn"),
                    book.get("language_code") == null ? null :
book.getString("language_code"),
                    book.get("asin") == null ? null : book.getString("asin"),
                    book.get("average_rating").toString().equals("") ?
Double.valueOf(0) : Double.valueOf(book.get("average_rating").toString()),
                    book.get("description") == null ? null :
book.getString("description"),
                    book.get("num_pages") == null ? null :
book.getInteger("num_pages"),
                    book.get("publication_day") == null ? null :
book.getInteger("publication_day"),
                    book.get("publication_month") == null ? null :
book.getInteger("publication_month"),
                    book.get("publication_year") == null ? null :
book.getInteger("publication_year"),
                    book.get("image_url") == null ? null :
book.getString("image_url"),
                    book.getString("book_id"),
                    book.getInteger("ratings_count"),
                    book.getString("title"),
                    authorsLis,
                    genres,
                    reviewsList
            ));
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        if (cursor != null)
            cursor.close();
    }
    return result;
}

public ArrayList<User> searchUser(String Username) {
    MongoCollection<Document> user = md.getCollection(usersCollection);
    List<Document> queryResults;
    ArrayList<User> result = new ArrayList<>();
    //search on exact username
    try {
        if (Username.equals("")) {
            queryResults = user.find().into(new ArrayList());
        } else {
            queryResults = user.find(eq("username", Username)).into(new
ArrayList());
        }

        for (Document r : queryResults) {
            ArrayList<String> listReviewID = (ArrayList<String>)
r.get("liked_review");
            result.add(new User(r.getString("user_id"),
r.get("name").toString(), r.get("username").toString(),
```

```
r.get("email").toString(), r.get("password").toString(), listReviewID, (Integer)
r.get("follower_count")));
        }

        //search on name or surname
        if (!Username.equals("")) {
            queryResults = user.find(text("\"" + Username + "\"", new
TextSearchOptions().caseSensitive(false))).into(new ArrayList());
            User us;
            for (Document r : queryResults) {
                ArrayList<String> listReviewID = (ArrayList<String>)
r.get("liked_review");
                us = new User(r.getString("user_id"), r.get("name").toString(),
r.get("username").toString(), r.get("email").toString(),
r.get("password").toString(), listReviewID, (Integer) r.get("follower_count"));
                if (!result.contains(us))
                    result.add(us);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}

public ArrayList<Author> searchAuthor(String Username) {
    MongoCollection<Document> author = md.getCollection(authorCollection);
    List<Document> queryResults;
    ArrayList<Author> result = new ArrayList<>();

    //search on exact username
    try {
        if (Username.equals("")) {
            queryResults = author.find().into(new ArrayList());
        } else {
            queryResults = author.find(eq("username", Username)).into(new
ArrayList());
        }

        for (Document r : queryResults) {
            ArrayList<String> listReviewID = (ArrayList<String>)
r.get("liked_review");
            result.add(new Author(r.getString("author_id"),
r.get("name").toString(), r.get("username").toString(),
r.get("email").toString(), r.get("password").toString(), listReviewID, (Integer)
r.get("follower_count")));
        }

        if (!Username.equals("")) {
            //search on name or surname
            queryResults = author.find(text("\"" + Username + "\"", new
TextSearchOptions().caseSensitive(false))).into(new ArrayList());
            Author auth;
            for (Document r : queryResults) {
                ArrayList<String> listReviewID = (ArrayList<String>)
r.get("liked_review");
                auth = new Author(r.getString("author_id"),
r.get("name").toString(), r.get("username").toString(),
r.get("email").toString(), r.get("password").toString(), listReviewID, (Integer)
r.get("follower_count"));
                if (!result.contains(auth))
```

```
                    result.add(auth);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return result;
    }

    public ArrayList<Report> loadReviewReported() {
        ArrayList<Report> reportedReview = new ArrayList<>();
        try {
            MongoCollection<Document> reports =
    md.getCollection(reportsCollection);
            List<Document> queryResults;
            queryResults = reports.find().into(new ArrayList<>());
            for (Document r : queryResults) {
                if (r.getString("type").equals("review")) {
                    reportedReview.add(new Report(
                            r.getString("report_id"),
                            r.getString("type"),
                            "",
                            "",
                            r.getString("book_id"),
                            "",
                            "",
                            0,
                            0,
                            0,
                            0,
                            "",
                            r.getString("review_id"),
                            r.getString("review_text"),
                            r.get("rating").toString(),
                            r.getString("user_id"),
                            r.getString("username"),
                            null,
                            null
                    ));
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return reportedReview;
    }

    public ArrayList<Report> loadBookReported() {
        ArrayList<Report> reportedBook = new ArrayList<>();
        try {
            MongoCollection<Document> reports =
    md.getCollection(reportsCollection);
            List<Document> queryResults;
            queryResults = reports.find().into(new ArrayList<>());
            ArrayList<Author> authorsLis = new ArrayList<>();
            for (Document r : queryResults) {
                if (r.getString("type").equals("book")) {
                    ArrayList<Document> authors = (ArrayList<Document>)
    r.get("authors");
                    ArrayList<String> genres = (ArrayList<String>) r.get("genres");
                    for (Document a : authors) {
```

```
                    Author author = new Author(
                            a.getString("author_id"),
                            a.getString("author_name"),
                            a.getString("author_username"),
                            "",
                            "",
                            null,
                            0
                    );
                    authorsLis.add(author);
                }
                reportedBook.add(new Report(
                        r.getString("report_id"),
                        r.getString("type"),
                        r.getString("isbn"),
                        r.getString("asin"),
                        r.getString("book_id"),
                        r.getString("title"),
                        r.getString("description"),
                        (Integer) r.get("num_pages"),
                        (Integer) r.get("publication_day"),
                        (Integer) r.get("publication_month"),
                        (Integer) r.get("publication_year"),
                        r.getString("image_url"),
                        "",
                        "",
                        "",
                        "",
                        "",
                        authorsLis,
                        genres
                ));
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return reportedBook;
}

public ArrayList<Log> loadLogs() {
    ArrayList<Log> logsList = new ArrayList<>();
    try {
        MongoCollection<Document> logs = md.getCollection(logsCollection);
        List<Document> results = logs.find().sort(descending("date")).into(new
ArrayList<>());
        for (Document l : results) {
            if (l.getString("type").equals("review")) {
                logsList.add(
                        new Log(
                                l.getString("id"),
                                l.getDate("date"),
                                l.getString("operation"),
                                l.getString("admin"),
                                l.getString("report_id"),
                                l.getString("type"),
                                "",
                                "",
                                l.getString("book_id"),
                                "",
                                "",
```

```
                                0,
                                0,
                                0,
                                0,
                                "",
                                l.getString("review_id"),
                                l.getString("review_text"),
                                l.getString("rating"),
                                l.getString("user_id"),
                                l.getString("username"),
                                null,
                                null
                        )
                );
            } else if (l.getString("type").equals("book")) {
                ArrayList<Author> authorsLis = new ArrayList<>();
                ArrayList<Document> authors = (ArrayList<Document>)
l.get("authors");
                ArrayList<String> genres = (ArrayList<String>) l.get("genres");
                for (Document a : authors) {
                    Author author = new Author(
                            a.getString("author_id"),
                            a.getString("author_name"),
                            a.getString("author_username"),
                            "",
                            "",
                            null,
                            0
                    );
                    authorsLis.add(author);
                }
                logsList.add(
                        new Log(
                                l.getString("id"),
                                l.getDate("date"),
                                l.getString("operation"),
                                l.getString("admin"),
                                l.getString("report_id"),
                                l.getString("type"),
                                l.getString("isbn"),
                                l.getString("asin"),
                                l.getString("book_id"),
                                l.getString("title"),
                                l.getString("description"),
                                (Integer) l.get("num_pages"),
                                (Integer) l.get("publication_day"),
                                (Integer) l.get("publication_month"),
                                (Integer) l.get("publication_year"),
                                l.getString("image_url"),
                                "",
                                "",
                                "",
                                "",
                                "",
                                authorsLis,
                                genres
                        )
                );
            }
        }
    } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        return logsList;
    }
```

- Update operations that allow us to keep updated books, author, users, reviews, logs and reports collections:

```
    public boolean updatePassword(String newPassword) {
        MongoCollection<Document> user = md.getCollection(session.getIsAuthor() ?
    authorCollection : usersCollection);
        UpdateResult updateResult = null;
        String username;
        try {
            if (session.getIsAuthor())
                username = session.getLoggedAuthor().getNickname();
            else
                username = session.getLoggedUser().getNickname();
            updateResult = user.updateOne(eq("username", username),
    Updates.set("password", newPassword));
        } catch (Exception e) {
            e.printStackTrace();
        }
        if (updateResult != null & updateResult.getModifiedCount() == 1)
            return true;
        return false;
    }

    public Double updateRating(ArrayList<Review> reviews) {
        Double ratingSum = 0.0;
        if (reviews.size() > 0) {
            for (Review r : reviews) {
                ratingSum += Double.parseDouble(r.getRating());
            }
            return ratingSum / reviews.size();
        } else {
            return ratingSum;
        }
    }
```

- Delete operations that allow us to delete books, author, users, reviews, logs and reports:

```
    public boolean deleteBookMongo(Book book) {
        MongoCollection<Document> books = md.getCollection(bookCollection);
        DeleteResult deleteResult = null;
        Book backup = getBookByID(book.getBook_id());
        try {
            deleteResult = books.deleteOne(eq("book_id", book.getBook_id()));
        }catch (Exception e){
            e.printStackTrace();
        }
        if(deleteResult != null && deleteResult.getDeletedCount() == 1)
            return true;
        return false;
    }

    public boolean deleteReview(String review_id, String book_id) {
        MongoCollection<Document> books = md.getCollection(bookCollection);
        Bson getBook = eq("book_id", book_id);
```

```
    try {
        UpdateResult removeReview = books.updateOne(getBook,
Updates.pull("reviews", new Document("review_id", review_id)));
        Book bookToUpdate = getBookByID(book_id);
        if (bookToUpdate == null) {
            return true;
        }
        if (removeReview.getModifiedCount() == 1) {
            Double newRating = updateRating(bookToUpdate.getReviews());
            UpdateResult updateAvgRating = books.updateOne(getBook,
Updates.set("average_rating", newRating));
            removeLikeReview(review_id, book_id);
            if (updateAvgRating.getModifiedCount() == 1) {
                return true;
            }
        }
    }catch (Exception e){
        e.printStackTrace();
    }
    return false;
}

public boolean deleteUserMongo(User userDel, String type) {
    MongoCollection<Document> user = md.getCollection(type.equals("author") ?
authorCollection : usersCollection);
    DeleteResult deleteResult = null;
    try {
        deleteResult = user.deleteOne(eq("username", userDel.getNickname()));
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (deleteResult != null && deleteResult.getDeletedCount() == 1)
        return true;
    return false;
}

public boolean deleteReport(Report report, Boolean unreport) {
    MongoCollection<Document> reports = md.getCollection(reportsCollection);
    DeleteResult result = null;
    try {
        result = reports.deleteOne(eq("report_id", report.getReport_id()));
        if (unreport) {
            if (addLog(report, "unreport")) {
                return true;
            } else {
                return false;
            }
        } else {
            if (addLog(report, "delete")) {
                return true;
            } else {
                return false;
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (result != null) {
        return result.wasAcknowledged();
```

```
    }
    return false;
}

public boolean deleteLog(Log log) {
    MongoCollection<Document> reports = md.getCollection(logsCollection);
    DeleteResult result = null;
    try {
        result = reports.deleteOne(eq("id", log.getId()));
    } catch (Exception e) {
        e.printStackTrace();
    }
    if (result != null) {
        return result.wasAcknowledged();
    }
    return false;
}
```

Neo4J Analytics & Suggestions

- Users that are connected to the visualized user by a read/to_read relationship through a book will be suggested to me If they are not already in my follow list

```
public ArrayList<User> similarUsers(String username, String type, String
myUsername, String myType) {
    ArrayList<User> suggestion = new ArrayList<>();
    ArrayList<User> queryResult = new ArrayList<>();

    try (Session session = nd.getDriver().session()) {
        suggestion = session.readTransaction((TransactionWork<ArrayList<User>>)
tx -> {
            Result result = tx.run("MATCH (u:" + myType + "{username:'" +
myUsername + "'}) " +
                    "OPTIONAL MATCH (u)-[:FOLLOW]->(f:User) " +
                    "WITH u, collect(f) as followed " +
                    "MATCH (u1:" + type + "{username:'"

   + username + "'})-[:READ|:TO_READ]->(b:Book)<-[]-(u2:User) " +
                        "WHERE u1<>u2 AND u<>u2 AND NOT u2 IN followed " +
                        "RETURN DISTINCT u2.id,u2.name,u2.username");
                while (result.hasNext()) {
                    Record r = result.next();
                    queryResult.add(new User(r.get("u2.id").asString(),
   r.get("u2.name").asString(), r.get("u2.username").asString(), "", "", new
   ArrayList<>(), 0));
                }
                return queryResult;
            });
        }
        return suggestion;
   }
```

- Authors that are connected to the visualized user by a read/to_read relationship through a book will be suggested to me If they are not already in my follow list

```
public ArrayList<Author> similarAuthors(String username, String type, String
myUsername, String myType) {
    ArrayList<Author> suggestion;
    ArrayList<Author> queryResult = new ArrayList<>();
    try (Session session = nd.getDriver().session()) {
        suggestion = (ArrayList<Author>)
session.readTransaction((TransactionWork<ArrayList<Author>>) tx -> {
            Result result = tx.run("MATCH (u:" + myType + "{username:'"
```

```
+ myUsername + "'}) " +
                    "OPTIONAL MATCH (u)-[:FOLLOW]->(f:Author) " +
                    "WITH u, collect(f) as followed " +
                    "MATCH (u1:" + type + "{username:'"

      + username + "'})-[:READ|:TO_READ]->(b:Book)<-[:READ|:TO_READ]-(a:Author)
      " +
                         "WHERE u1<>a AND u<>a AND NOT a IN followed " +
                         "RETURN DISTINCT a.id,a.name,a.username");
                while (result.hasNext()) {
                    Record r = result.next();
                    queryResult.add(new Author(r.get("a.id").asString(),
      r.get("a.name").asString(), r.get("a.username").asString(), "", "", new
      ArrayList<>(), 0));
                }
                return queryResult;
            });
        }
        return suggestion;
      }
```

- Books that are connected to the visualized book by a wrote relationship through the authors (of the visualized book) will be suggested to me If they are not already in my read/to_read list

```
public ArrayList<Book> similarBooks(String book_id,String myUsername, String
myType) {
    ArrayList<Book> suggestion = new ArrayList<>();
    ArrayList<Book> queryResult = new ArrayList<>();

    try (Session session = nd.getDriver().session()) {
        suggestion = session.readTransaction((TransactionWork<ArrayList<Book>>)
tx -> {
            Result result = tx.run("OPTIONAL MATCH (u1:" + myType +
"{username:'" + myUsername + "'})-[:READ|:TO_READ]->(b:Book) " +
                    "WITH collect(b) as readings " +
                    "MATCH (b1:Book)<-[:WROTE]-(a:Author)-[:WROTE]->(b2:Book) "
+
                    "WHERE b1.id = '" + book_id + "' AND b1<>b2 " +
                    "AND NOT b2 IN readings " +
                    "RETURN DISTINCT b2.id,b2.title");
            while (result.hasNext()) {
                Record r = result.next();
                queryResult.add(new Book(r.get("b2.title").asString(),
r.get("b2.id").asString()));
            }
            return queryResult;
        });
    }
    return suggestion;
```

```
}
```

- Authors that worked together with the authors, of the visualized book, on another paper will be suggested to me If they are not already in follow list

```
public ArrayList<Author> suggestedAuthors(String book_id, String myUsername,
String myType) {
    ArrayList<Author> suggestion;
    ArrayList<Author> queryResult = new ArrayList<>();

    try (Session session = nd.getDriver().session()) {
        suggestion = (ArrayList<Author>)
session.readTransaction((TransactionWork<ArrayList<Author>>) tx -> {
            Result result = tx.run("MATCH (u1:" + myType + "{username:'"
   + myUsername + "'}) " +
                    "OPTIONAL MATCH (u1)-[:FOLLOW]->(f:Author) " +
                    "WITH u1,collect(f) as followed " +
                    "MATCH (b1:Book)<-[:WROTE]-(a1:Author)-[:WROTE]-
   >(b2:Book)<-[:WROTE]-(a2:Author) " +
                    "WHERE b1.id = '" + book_id + "' AND b1<>b2 AND a1<>a2 "
   +
                    "AND NOT a2 IN followed AND u1<>a2 " +
                    "RETURN DISTINCT a2.id,a2.name,a2.username");
            while (result.hasNext()) {
                Record r = result.next();
                queryResult.add(new Author(r.get("a2.id").asString(),
   r.get("a2.name").asString(), r.get("a2.username").asString(), "", "", new
   ArrayList<>(), 0));
            }
            return queryResult;
        });
    }
    return suggestion;
}
```

- Get the Top desired(to_read)/popular(read) books

```
public ArrayList<RankingObject> topBooks(String type, Integer limit) {
    ArrayList<RankingObject> books;
    ArrayList<RankingObject> queryResult = new ArrayList<>();

    try (Session session = nd.getDriver().session()) {
        books = (ArrayList<RankingObject>)
session.readTransaction((TransactionWork<ArrayList<RankingObject>>) tx -> {
            Result result = tx.run("MATCH (b:Book)<-[r:" + type + "]-() " +
                    "RETURN b.id,b.title,count(r) as count " +
                    "ORDER BY count DESC " +
                    "LIMIT " + limit + "");
```

```
            while (result.hasNext()) {
                Record r = result.next();
                queryResult.add(new RankingObject(r.get("b.title").asString(),
Integer.valueOf(r.get("count").toString())));
            }
            return queryResult;
        });
    }
    return books;
}
```

MongoDB Analitycs Implementations

- Average rating per category about author's books

This analytics gives us informations about the average rating about each category in which an author has written books.

```
[
      { $match: { 'authors.author_id': '4748056' }},
      { $unwind: { path: '$genres', preserveNullAndEmptyArrays: false }},
      { $group: { _id: '$genres', average_rating: { $avg: '$average_rating' }}},
      { $sort: { average_rating: -1 }}
]
```

The Mongo Java Driver code:

```
public ArrayList<Genre> averageRatingCategoryAuthor(String username) {
    MongoCollection<Document> author = md.getCollection(authorCollection);
    MongoCollection<Document> books = md.getCollection(bookCollection);
    String author_id = null;
    ArrayList<Genre> topRated = new ArrayList<>();
    Bson getAuthor;
    Bson unwindGenres;
    Bson groupGenres;
    Bson sortAvg;
    try (MongoCursor<Document> cursor = author.find(eq("username",
username)).iterator()) {
        while (cursor.hasNext()) {
            author_id = cursor.next().getString("author_id");
        }
    } catch (Exception e) {
        e.printStackTrace();
        return topRated;
    }
    if (author_id == null)
        return null;
    getAuthor = match(eq("authors.author_id", author_id));
    unwindGenres = unwind("$genres", new
UnwindOptions().preserveNullAndEmptyArrays(false));
    groupGenres = group("$genres", avg("average_rating", "$average_rating"));
    sortAvg = sort(orderBy(descending("average_rating")));
    try (MongoCursor<Document> cursor = books.aggregate(Arrays.asList(getAuthor,
unwindGenres, groupGenres, sortAvg)).iterator()) {
        while (cursor.hasNext()) {
            Document stat = cursor.next();
            Double avg = Math.round((stat.getDouble("average_rating")) * 100) /
100.0;
            Genre genre = new Genre(stat.getString("_id"), Double.valueOf(avg));
            topRated.add(genre);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return topRated;
}
```

- Average review rating grouped by books categories

This analytics gives us informations about the average rating about each category in which an user has left a review.

```
[
        {$match: { 'reviews.user_id': 'f30a3aeb4761541cfc1907c83353a563' }},
        {$unwind: { path: '$reviews', preserveNullAndEmptyArrays: false }},
        {$unwind: { path: '$genres', preserveNullAndEmptyArrays: false }},
        {$group: { _id: '$genres', average_rating: { $avg: '$reviews.rating' }}},
        {$sort: {average_rating: -1}}
]
```

The Mongo Java Driver code:

```
public ArrayList<Genre> averageRatingCategoryUser(String username) {
    MongoCollection<Document> books = md.getCollection(bookCollection);
    ArrayList<Genre> topRated = new ArrayList<>();
    Bson getUser;
    Bson unwindReviews;
    Bson unwindGenres;
    Bson groupGenres;
    Bson sortAvg;
    getUser = match(eq("reviews.username", username));
    unwindReviews = unwind("$reviews", new
UnwindOptions().preserveNullAndEmptyArrays(false));
    unwindGenres = unwind("$genres", new
UnwindOptions().preserveNullAndEmptyArrays(false));
    groupGenres = group("$genres", avg("average_rating", "$reviews.rating"));
    sortAvg = sort(orderBy(descending("average_rating")));
    try (MongoCursor<Document> cursor =
books.aggregate(Arrays.asList(unwindReviews, getUser, unwindGenres, groupGenres,
sortAvg)).iterator()) {
        while (cursor.hasNext()) {
            Document stat = cursor.next();
            Double avg = Math.round((stat.getDouble("average_rating")) * 100) /
100.0;
            Genre genre = new Genre(stat.getString("_id"), Double.valueOf(avg));
            topRated.add(genre);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return topRated;

}
```

- Books published

This analytics gives us informations about the number of book published for each category filtered by year

```
[
      {$match: { publication_year: 2013 }},
      {$unwind: { path: '$genres', preserveNullAndEmptyArrays: false }},
      {$group: { _id: '$genres', publications: { $sum: 1 }}}
]
```

The Mongo Java Driver code:

```
public ArrayList<Genre> searchRankBook(Integer year) {
    MongoCollection<Document> bookGenres = md.getCollection(bookCollection);

    ArrayList<Genre> genres = new ArrayList<>();
    Bson match = match(in("publication_year", year));
    Bson unwind = unwind("$genres");
    Bson group = group("$genres", sum("counter", 1));

    try (MongoCursor<Document> result =
bookGenres.aggregate(Arrays.asList(match, unwind, group)).iterator()) {

        while (result.hasNext()) {
            Document y = result.next();
            genres.add(new Genre(y.getString("_id"),
Double.valueOf(y.get("counter").toString())));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return genres;
}
```

- Users rank

This analytics gives us the users rank based on the average review rating received

```
[
      {$unwind: {path: '$reviews', preserveNullAndEmptyArrays: false }},
      {$project: {'reviews.username': 1 ,
      'reviews.likes': {$ifNull: [ '$reviews.likes', '$reviews.helpful' ]}}},
      {$group: { _id: '$reviews.username' , reviews_number: {$sum: 1 },
            average_likes: {$avg: '$reviews.likes' }}},
      {$match: { reviews_number: {$gte: 200 }}},
      {$sort: {average_likes: -1 }},
      {$limit: 100}
]
```

The Mongo Java Driver code:

```
public ArrayList<RankingObject> rankReview() {
    MongoCollection<Document> book = md.getCollection(bookCollection);
    ArrayList<RankingObject> users = new ArrayList<>();

    Bson unwindReviews = unwind("$reviews");
    //project likes : ($likes != null) $likes : $helpful
    Bson projectLikes = new Document("$project",
            new Document("reviews.username", 1L)
                    .append("reviews.likes",
                            new Document("$ifNull",
Arrays.asList("$reviews.likes", "$reviews.helpful"))));
    Bson groupUsername = group("$reviews.username", sum("reviews_number", 1),
avg("average_likes", "$reviews.likes"));
    Bson matchGreaterThan200 = match(gte("reviews_number", 200));
    Bson sort = sort(orderBy(descending("average_likes")));
    Bson limit = limit(100);

    try (MongoCursor<Document> result =
book.aggregate(Arrays.asList(unwindReviews, projectLikes, groupUsername,
matchGreaterThan200, sort, limit)).iterator()) {

        while (result.hasNext()) {
            Document document = result.next();
            users.add(new RankingObject(document.getString("_id"),
                    document.getInteger("reviews_number"),
                    document.getDouble("average_likes")));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return users;

}
```

Neo4J Index Analysis

Constraint Name	Constraint Type	Constraint Lable	Constraint Properties
user_username	BTREE [UNIQUE]	:User	username
author_username	BTREE [UNIQUE]	:Author	username
book_id	BTREE [UNIQUE]	:Book	id

author_username

Match (a:Author{username:"elmo.kling"})-[f:FOLLOW]->(p) return p;

Without constrain

```
neo4j@neo4j> Profile Match (a:Author{username:"elmo.kling"})-[f:FOLLOW]->(p) return p;
+-------------------------------------------------------------------------------------+
| p                                                                                   |
+-------------------------------------------------------------------------------------+
| (:Author {name: "Alan Steinberg", id: "56752", username: "aleisha.considine1"}) |
| (:Author {name: "Bill Russell", id: "75414", username: "delmar.miller"})        |
| (:Author {name: "r`maephng", id: "3400307", username: "von.sawayn"})            |
+-------------------------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------------------------+
| Plan      | Statement    | Version      | Planner | Runtime       | Time | DbHits | Rows | Memory (Bytes) |
+-----------------------------------------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.4" | "COST"  | "INTERPRETED" | 75   | 119223 | 3    | 64             |
+-----------------------------------------------------------------------------------------------------------------------+


+-----------------------+---------------------------+----------------+-------+---------+-------------------------+
| Operator              | Details                   | Estimated Rows | Rows  | DB Hits | Page Cache Hits/Misses |
+-----------------------+---------------------------+----------------+-------+---------+-------------------------+
| +ProduceResults@neo4j | p                         |            649 |     3 |      12 |                     0/0 |
| |                     +---------------------------+----------------+-------+---------+-------------------------+
| +Expand(All)@neo4j    | (a)-[f:FOLLOW]->(p)       |            649 |     3 |       4 |                     0/0 |
| |                     +---------------------------+----------------+-------+---------+-------------------------+
| +Filter@neo4j         | a.username = $autostring_0 |           2980 |     1 |   59603 |                     0/0 |
| |                     +---------------------------+----------------+-------+---------+-------------------------+
| +NodeByLabelScan@neo4j | a:Author                  |          59603 | 59603 |   59604 |                     0/0 |
+-----------------------+---------------------------+----------------+-------+---------+-------------------------+

3 rows
ready to start consuming query after 17 ms, results consumed after another 58 ms
```

With constrain

```
neo4j@neo4j> Profile Match (a:Author{username:"elmo.kling"})-[f:FOLLOW]->(p) return p;
+-------------------------------------------------------------------------------+
| p                                                                             |
+-------------------------------------------------------------------------------+
| (:Author {name: "Alan Steinberg", id: "56752", username: "aleisha.considine1"}) |
| (:Author {name: "Bill Russell", id: "75414", username: "delmar.miller"})      |
| (:Author {name: "r`maephng", id: "3400307", username: "von.sawayn"})          |
+-------------------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------------+
| Plan      | Statement   | Version      | Planner | Runtime       | Time | DbHits | Rows | Memory (Bytes) |
+-----------------------------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.4" | "COST"  | "INTERPRETED" | 2    | 18     | 3    | 64             |
+-----------------------------------------------------------------------------------------------------------+

+----------------------+-------------------------------------------------------------+----------------+------+---------+-------------------------+
| Operator             | Details                                                     | Estimated Rows | Rows | DB Hits | Page Cache Hits/Misses |
+----------------------+-------------------------------------------------------------+----------------+------+---------+-------------------------+
| +ProduceResults@neo4j | p                                                          |              0 |    3 |      12 |                     0/0 |
| |                    +-------------------------------------------------------------+----------------+------+---------+-------------------------+
| +Expand(All)@neo4j   | (a)-[f:FOLLOW]->(p)                                         |              0 |    3 |       4 |                     0/0 |
| |                    +-------------------------------------------------------------+----------------+------+---------+-------------------------+
| +NodeIndexSeek@neo4j | BTREE INDEX a:Author(username) WHERE username = $autostring_0 |              1 |    1 |       2 |                     0/0 |
+----------------------+-------------------------------------------------------------+----------------+------+---------+-------------------------+

3 rows
ready to start consuming query after 1 ms, results consumed after another 1 ms
```

user_username

Match (a:User{username:"rodrigo.sporer"})-[f:FOLLOW]->(p) return p;

Without constrain

```
neo4j@neo4j> Profile Match (a:User{username:"rodrigo.sporer"})-[f:FOLLOW]->(p) return p;
+-----------------------------------------------------------------------------------------+
| p                                                                                       |
+-----------------------------------------------------------------------------------------+
| (:Author {name: "Nimue Brown", id: "5391189", username: "cherelle.wilkinson1"})         |
| (:Author {name: "Robert  Wolfe", id: "6950916", username: "racheal.schmeler"})          |
| (:Author {name: "Nicole Sager", id: "6538148", username: "leena.harvey"})               |
| (:User {name: "Cordie Harvey", id: "A2CUNJVX5PM1O6", username: "natividad.hintz"})      |
| (:User {name: "Kristeen Schneider", id: "A1Q5M90ZZEEAL1", username: "jesenia.bailey"}) |
| (:User {name: "Ira Kemmer", id: "A1PICGZ33Z0JVE", username: "monet.carter"})           |
+-----------------------------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------------+
| Plan      | Statement   | Version      | Planner | Runtime       | Time | DbHits | Rows | Memory (Bytes) |
+-----------------------------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.4" | "COST"  | "INTERPRETED" | 127  | 302002 | 6    | 64             |
+-----------------------------------------------------------------------------------------------------------+

+-----------------------+-----------------------------+----------------+--------+---------+-------------------------+
| Operator              | Details                     | Estimated Rows | Rows   | DB Hits | Page Cache Hits/Misses |
+-----------------------+-----------------------------+----------------+--------+---------+-------------------------+
| +ProduceResults@neo4j | p                           |           1854 |      6 |      24 |                     0/0 |
| |                     +-----------------------------+----------------+--------+---------+-------------------------+
| +Expand(All)@neo4j    | (a)-[f:FOLLOW]->(p)         |           1854 |      6 |       7 |                     0/0 |
| |                     +-----------------------------+----------------+--------+---------+-------------------------+
| +Filter@neo4j         | a.username = $autostring_0  |           7549 |      1 |  150985 |                     0/0 |
| |                     +-----------------------------+----------------+--------+---------+-------------------------+
| +NodeByLabelScan@neo4j | a:User                     |         150985 | 150985 |  150986 |                     0/0 |
+-----------------------+-----------------------------+----------------+--------+---------+-------------------------+

6 rows
ready to start consuming query after 20 ms, results consumed after another 107 ms
```

With constrain

```
neo4j@neo4j> Profile Match (a:User{username:"rodrigo.sporer"})-[f:FOLLOW]->(p) return p;
+-----------------------------------------------------------------------------------------------------------+
| p                                                                                                         |
+-----------------------------------------------------------------------------------------------------------+
| (:Author {name: "Nimue Brown", id: "5391189", username: "cherelle.wilkinson1"})                           |
| (:Author {name: "Robert  Wolfe", id: "6950916", username: "racheal.schmeler"})                            |
| (:Author {name: "Nicole Sager", id: "6538148", username: "leena.harvey"})                                 |
| (:User {name: "Cordie Harvey", id: "A2CUNJVX5PM1O6", username: "natividad.hintz"})                        |
| (:User {name: "Kristeen Schneider", id: "A1Q5M90ZZEEAL1", username: "jesenia.bailey"})                    |
| (:User {name: "Ira Kemmer", id: "A1PICGZ33Z0JVE", username: "monet.carter"})                              |
+-----------------------------------------------------------------------------------------------------------+

+-----------------------------------------------------------------------------------------------------------------------+
| Plan      | Statement   | Version      | Planner | Runtime       | Time | DbHits | Rows | Memory (Bytes) |
+-----------------------------------------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.4" | "COST"  | "INTERPRETED" | 2    | 33     | 6    | 64             |
+-----------------------------------------------------------------------------------------------------------------------+

+----------------------+-----------------------------------------------------------------+----------------+------+---------+-------------------------+
| Operator             | Details                                                         | Estimated Rows | Rows | DB Hits | Page Cache Hits/Misses |
+----------------------+-----------------------------------------------------------------+----------------+------+---------+-------------------------+
| +ProduceResults@neo4j | p                                                               |              0 |    6 |      24 |                     0/0 |
| |                    +-----------------------------------------------------------------+----------------+------+---------+-------------------------+
| +Expand(All)@neo4j   | (a)-[f:FOLLOW]->(p)                                             |              0 |    6 |       7 |                     0/0 |
| |                    +-----------------------------------------------------------------+----------------+------+---------+-------------------------+
| +NodeIndexSeek@neo4j | BTREE INDEX a:User(username) WHERE username = $autostring_0     |              1 |    1 |       2 |                     0/0 |
+----------------------+-----------------------------------------------------------------+----------------+------+---------+-------------------------+

6 rows
ready to start consuming query after 1 ms, results consumed after another 1 ms
```

```
MATCH (u:  User  {username: 'rodrigo.sporer'})
    OPTIONAL MATCH (u)-[:FOLLOW]->(f:User)
    WITH u, collect(f) as followed
    MATCH (u1:  User  {username:'rodrigo.sporer'})-[:READ|:TO_READ]->(b:Book)<-[]-(u2:User)
    WHERE u1<>u2 AND u<>u2 AND NOT u2 IN followed
    RETURN DISTINCT u2.id,u2.name,u2.username
```

Without constrain

```
+-----------------------------------------------------------------------------------------------------------+
| Plan      | Statement   | Version      | Planner | Runtime      | Time | DbHits | Rows | Memory (Bytes) |
+-----------------------------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.4" | "COST"  | "INTERPRETED" | 511  | 839795 | 1    | 704            |
+-----------------------------------------------------------------------------------------------------------+

+----------------------+-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| Operator             | Details                                                                                             | Estimated Rows | Rows   | DB Hits | Memory (Bytes)  | Page Cache Hits/Misses  | Ordered by |
+----------------------+-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +ProduceResults@neo4j | `u2.id`, `u2.name`, `u2.username`                                                                   |          14788 |      1 |       0 |                 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Distinct@neo4j      | u2.id AS `u2.id`, u2.name AS `u2.name`, u2.username AS `u2.username`                                 |          14788 |      1 |       6 |             472 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Filter@neo4j        | not u1 = u2 AND not anon_1 = anon_2 AND u1:User AND u1.username = $autostring_1                     |          15566 |      2 |   18624 |                 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Expand(All)@neo4j   | (b)<-[anon_1:READ|TO_READ]-(u1)                                                                     |         396678 |  43202 |   76496 |                 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Filter@neo4j        | b:Book                                                                                              |        1375714 |  33294 |   70359 |                 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Expand(All)@neo4j   | (u2)-[anon_2]->(b)                                                                                  |        1375714 |  70359 |  221340 |                 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Filter@neo4j        | not u2 IN followed AND not u = u2                                                                   |        2951673 | 150981 |       0 |             120 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Apply@neo4j         |                                                                                                     |       13118546 | 150985 |       0 |                 |                     0/0 | u ASC      |
| |\                   +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| | +NodeByLabelScan@neo4j | u2:User                                                                                       |       13118546 | 150985 |  150986 |                 |                     0/0 |            |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +OrderedAggregation@neo4j | u, collect(f) AS followed                                                                      |           87 |      1 |       0 |             280 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +OptionalExpand(All)@neo4j | (u)-[anon_0:FOLLOW]->(f) WHERE f:User                                                          |           7549 |      3 |      13 |                 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +Filter@neo4j        | u.username = $autostring_0                                                                          |           7549 |      1 |  150985 |                 |                     0/0 | u ASC      |
| |                    +-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+
| +NodeByLabelScan@neo4j | u:User                                                                                        |         150985 | 150985 |  150986 |                 |                     0/0 | u ASC      |
+----------------------+-----------------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-------------------------+------------+

1 row
ready to start consuming query after 68 ms, results consumed after another 443 ms
```

With constrain

```
+-----------------------------------------------------------------------------------------+
| Plan      | Statement   | Version      | Planner | Runtime       | Time | DbHits | Rows | Memory (Bytes) |
+-----------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.4" | "COST"  | "INTERPRETED" | 64   | 74     | 1    | 992            |
+-----------------------------------------------------------------------------------------+
```

```
+-----------------------------+-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| Operator                    | Details                                                                                                               | Estimated Rows | Rows | DB Hits | Memory (Bytes)  | Page Cache Hits/Misses |
+-----------------------------+-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +ProduceResults@neo4j       | `u2.id`, `u2.name`, `u2.username`                                                                                     |              0 |    1 |       0 |                 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +Distinct@neo4j             | u2.id AS `u2.id`, u2.name AS `u2.name`, u2.username AS `u2.username`                                                   |              0 |    1 |       6 |             472 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +Filter@neo4j               | not u2 IN followed AND not u = u2 AND not u1 = u2 AND not anon_1 = anon_2 AND u2:User                                  |              0 |    2 |      13 |             120 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +Expand(All)@neo4j          | (b)<-[anon_2]-(u2)                                                                                                    |              0 |   19 |      25 |                 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +Filter@neo4j               | b:Book                                                                                                                |              0 |    6 |       6 |                 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +Expand(All)@neo4j          | (u1)-[anon_1:READ|TO_READ]->(b)                                                                                       |              0 |    6 |       7 |                 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +Apply@neo4j                |                                                                                                                       |              1 |    1 |       0 |                 |                   0/0 |
| |\                          +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| | +NodeUniqueIndexSeek@neo4j | UNIQUE u1:User(username) WHERE username = $autostring_1                                                               |              1 |    1 |       2 |                 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +EagerAggregation@neo4j     | u, collect(f) AS followed                                                                                             |              1 |    1 |       0 |             760 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +OptionalExpand(All)@neo4j  | (u)-[anon_0:FOLLOW]->(f) WHERE f:User                                                                                 |              1 |    3 |      13 |                 |                   0/0 |
| |                           +-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
| +NodeUniqueIndexSeek@neo4j  | UNIQUE u:User(username) WHERE username = $autostring_0                                                                |              1 |    1 |       2 |                 |                   0/0 |
+-----------------------------+-----------------------------------------------------------------------------------------------------------------------+----------------+------+---------+-----------------+-----------------------+
```

```
1 row
ready to start consuming query after 63 ms, results consumed after another 1 ms
```

user_username + book_id

```
OPTIONAL MATCH (u1:User {username:'rodrigo.sporer'})-[:READ|:TO_READ]->(b:Book)
    WITH collect(b) as readings
    MATCH (b1:Book)<-[:WROTE]-(a:Author)-[:WROTE]->(b2:Book)
    WHERE b1.id = '1819013' AND b1<>b2
    AND NOT b2 IN readings
    RETURN DISTINCT b2.id,b2.title
```

Without constrains

```
+-----------------------------------------------------------------------------------------+
| Plan      | Statement   | Version      | Planner | Runtime       | Time | DbHits | Rows | Memory (Bytes) |
+-----------------------------------------------------------------------------------------+
| "PROFILE" | "READ_ONLY" | "CYPHER 4.4" | "COST"  | "INTERPRETED" | 217  | 337956 | 26   | 6272           |
+-----------------------------------------------------------------------------------------+
```

```
+-----------------------------------------+-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| Operator                                | Details                                                                                       | Estimated Rows | Rows   | DB Hits | Memory (Bytes)  | Page Cache Hits/Misses | Ordered by |
+-----------------------------------------+-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +ProduceResults@neo4j                   | `b2.id`, `b2.title`                                                                           |           4908 |     26 |       0 |                 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Distinct@neo4j                         | b2.id AS `b2.id`, b2.title AS `b2.title`                                                      |           4908 |     26 |      52 |            5760 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Filter@neo4j                           | not b2 IN readings AND not b1 = b2 AND not anon_1 = anon_2 AND b2:Book                        |           5166 |     26 |      26 |             120 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Expand(All)@neo4j                      | (a)-[anon_2:WROTE]->(b2)                                                                      |          23192 |     27 |      28 |                 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Filter@neo4j                           | a:Author                                                                                      |           8314 |      1 |       1 |                 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Expand(All)@neo4j                      | (b1)<-[anon_1:WROTE]-(a)                                                                      |           8314 |      1 |       2 |                 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Filter@neo4j                           | b1.id = $autostring_1                                                                         |           6556 |      1 |  131111 |                 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Apply@neo4j                            |                                                                                               |         131111 | 131111 |       0 |                 |                   0/0 |            |
| |\                                      +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| | +NodeByLabelScan@neo4j                | b1:Book                                                                                       |         131111 | 131111 |  131112 |                 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +EagerAggregation@neo4j                 | collect(b) AS readings                                                                        |              1 |      1 |       0 |             352 |                   0/0 |            |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Optional@neo4j                         |                                                                                               |           1665 |      6 |       0 |                 |                   0/0 | anon_0 ASC |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +Filter@neo4j                           | u1.username = $autostring_0 AND u1:User AND b:Book                                            |           1665 |      6 |   37817 |                 |                   0/0 | anon_0 ASC |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +OrderedDistinct@neo4j                  | anon_0, u1, b                                                                                 |          37805 |  37805 |       0 |             296 |                   0/0 | anon_0 ASC |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +OrderedUnion@neo4j                     |                                                                                               |          37805 |  37805 |       0 |                 |                   0/0 | anon_0 ASC |
| |\                                      +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| | +DirectedRelationshipTypeScan@neo4j   | (u1)-[anon_0:TO_READ]->(b)                                                                    |          18902 |  18902 |   18903 |                 |                   0/0 | anon_0 ASC |
| |                                       +-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
| +DirectedRelationshipTypeScan@neo4j     | (u1)-[anon_0:READ]->(b)                                                                       |          18903 |  18903 |   18904 |                 |                   0/0 | anon_0 ASC |
+-----------------------------------------+-----------------------------------------------------------------------------------------------+----------------+--------+---------+-----------------+-----------------------+------------+
```

```
26 rows
ready to start consuming query after 55 ms, results consumed after another 162 ms
```

With user_username constrain

Plan	Statement	Version	Planner	Runtime	Time	DbHits	Rows	Memory (Bytes)
"PROFILE"	"READ_ONLY"	"CYPHER 4.4"	"COST"	"INTERPRETED"	95	262347	26	5968

Operator	Details	Estimated Rows	Rows	DB Hits	Memory (Bytes)	Page Cache Hits/Misses
+ProduceResults@neo4j	`b2.id`, `b2.title`	4908	26	0		0/0
+Distinct@neo4j	b2.id AS `b2.id`, b2.title AS `b2.title`	4908	26	52	5760	0/0
+Filter@neo4j	not b2 IN readings AND not b1 = b2 AND not anon_1 = anon_2 AND b2:Book	5166	26	26	120	0/0
+Expand(All)@neo4j	(a)-[anon_2:WROTE]->(b2)	23192	27	28		0/0
+Filter@neo4j	a:Author	8314	1	1		0/0
+Expand(All)@neo4j	(b1)<-[anon_1:WROTE]-(a)	8314	1	2		0/0
+Filter@neo4j	b1.id = $autostring_1	6556	1	131111		0/0
+Apply@neo4j		131111	131111	0		0/0
| +NodeByLabelScan@neo4j	b1:Book	131111	131111	131112		0/0
+EagerAggregation@neo4j	collect(b) AS readings	1	1	0	352	0/0
+Optional@neo4j		1	6	0		0/0
+Filter@neo4j	b:Book	0	6	6		0/0
+Expand(All)@neo4j	(u1)-[anon_0:READ|TO_READ]->(b)	0	6	7		0/0
+NodeUniqueIndexSeek@neo4j	UNIQUE u1:User(username) WHERE username = $autostring_0	1	1	2		0/0

26 rows
ready to start consuming query after 1 ms, results consumed after another 94 ms

With user_username and book_id constrain

Plan	Statement	Version	Planner	Runtime	Time	DbHits	Rows	Memory (Bytes)
"PROFILE"	"READ_ONLY"	"CYPHER 4.4"	"COST"	"INTERPRETED"	2	126	26	5960

Operator	Details	Estimated Rows	Rows	DB Hits	Memory (Bytes)	Page Cache Hits/Misses
+ProduceResults@neo4j	`b2.id`, `b2.title`	1	26	0		0/0
+Distinct@neo4j	b2.id AS `b2.id`, b2.title AS `b2.title`	1	26	52	5760	0/0
+Filter@neo4j	not b2 IN readings AND not b1 = b2 AND not anon_1 = anon_2 AND b2:Book	1	26	26	120	0/0
+Expand(All)@neo4j	(a)-[anon_2:WROTE]->(b2)	4	27	28		0/0
+Filter@neo4j	a:Author	1	1	1		0/0
+Expand(All)@neo4j	(b1)<-[anon_1:WROTE]-(a)	1	1	2		0/0
+Apply@neo4j		1	1	0		0/0
| +NodeUniqueIndexSeek@neo4j	UNIQUE b1:Book(id) WHERE id = $autostring_1	1	1	2		0/0
+EagerAggregation@neo4j	collect(b) AS readings	1	1	0	352	0/0
+Optional@neo4j		1	6	0		0/0
+Filter@neo4j	b:Book	0	6	6		0/0
+Expand(All)@neo4j	(u1)-[anon_0:READ|TO_READ]->(b)	0	6	7		0/0
+NodeUniqueIndexSeek@neo4j	UNIQUE u1:User(username) WHERE username = $autostring_0	1	1	2		0/0

26 rows
ready to start consuming query after 1 ms, results consumed after another 1 ms

```
MATCH (u1:  User  {username: 'rodrigo.sporer'})
    OPTIONAL MATCH (u1)-[:FOLLOW]->(f:Author)
    WITH u1,collect(f) as followed
    MATCH (b1:Book)<-[:WROTE]-(a1:Author)-[:WROTE]->(b2:Book)<-[:WROTE]-(a2:Author)
    WHERE b1.id = '1819013' AND b1<>b2 AND a1<>a2
    AND NOT a2 IN followed AND u1<>a2
    RETURN DISTINCT a2.id,a2.name,a2.username
```

Without constrains

Plan	Statement	Version	Planner	Runtime	Time	DbHits	Rows	Memory (Bytes)
"PROFILE"	"READ_ONLY"	"CYPHER 4.4"	"COST"	"INTERPRETED"	303	564616	42	11080

Operator	Details	Estimated Rows	Rows	DB Hits	Memory (Bytes)	Page Cache Hits/Misses	Ordered by
+ProduceResults@neo4j	`a2.id`, `a2.name`, `a2.username`	429308	42	0		0/0	u1 ASC
+Distinct@neo4j	a2.id AS `a2.id`, a2.name AS `a2.name`, a2.username AS `a2.username`	429308	42	180	10840	0/0	u1 ASC
+Filter@neo4j	not a2 IN followed AND not a1 = a2 AND not u1 = a2 AND not anon_1 = anon_3 AND not anon_2 = anon_3 AND a2:Author	451903	60	60	120	0/0	u1 ASC
+Expand(All)@neo4j	(b2)<-[anon_3:WROTE]-(a2)	2276930	86	112		0/0	u1 ASC
+Filter@neo4j	not b1 = b2 AND not anon_1 = anon_2 AND b2:Book	1795435	26	26		0/0	u1 ASC
+Expand(All)@neo4j	(a1)-[anon_2:WROTE]->(b2)	2015079	27	28		0/0	u1 ASC
+Filter@neo4j	a1:Author	722339	1	1		0/0	u1 ASC
+Expand(All)@neo4j	(b1)<-[anon_1:WROTE]-(a1)	722339	1	2		0/0	u1 ASC
+Filter@neo4j	b1.id = $autostring_1	569588	1	131111		0/0	u1 ASC
+Apply@neo4j		11391765	131111	0		0/0	u1 ASC
+NodeByLabelScan@neo4j	b1:Book	11391765	131111	131112		0/0	
+OrderedAggregation@neo4j	u1, collect(f) AS followed	87	1	0	280	0/0	u1 ASC
+OptionalExpand(All)@neo4j	(u1)-[anon_0:FOLLOW]->(f) WHERE f:Author	7549	3	13		0/0	u1 ASC
+Filter@neo4j	u1.username = $autostring_0	7549	1	150985		0/0	u1 ASC
+NodeByLabelScan@neo4j	u1:User	150985	150985	150986		0/0	u1 ASC

42 rows
ready to start consuming query after 108 ms, results consumed after another 195 ms

With user_username constrain

Plan	Statement	Version	Planner	Runtime	Time	DbHits	Rows	Memory (Bytes)
"PROFILE"	"READ_ONLY"	"CYPHER 4.4"	"COST"	"INTERPRETED"	100	262647	42	11376

Operator	Details	Estimated Rows	Rows	DB Hits	Memory (Bytes)	Page Cache Hits/Misses
+ProduceResults@neo4j	`a2.id`, `a2.name`, `a2.username`	4941	42	0		0/0
+Distinct@neo4j	a2.id AS `a2.id`, a2.name AS `a2.name`, a2.username AS `a2.username`	4941	42	180	10840	0/0
+Filter@neo4j	not a2 IN followed AND not a1 = a2 AND not u1 = a2 AND not anon_1 = anon_3 AND not anon_2 = anon_3 AND a2:Author	5201	60	60	120	0/0
+Expand(All)@neo4j	(b2)<-[anon_3:WROTE]-(a2)	26206	86	112		0/0
+Filter@neo4j	not b1 = b2 AND not anon_1 = anon_2 AND b2:Book	20664	26	26		0/0
+Expand(All)@neo4j	(a1)-[anon_2:WROTE]->(b2)	23192	27	28		0/0
+Filter@neo4j	a1:Author	8314	1	1		0/0
+Expand(All)@neo4j	(b1)<-[anon_1:WROTE]-(a1)	8314	1	2		0/0
+Filter@neo4j	b1.id = $autostring_1	6556	1	131111		0/0
+Apply@neo4j		131111	131111	0		0/0
+NodeByLabelScan@neo4j	b1:Book	131111	131111	131112		0/0
+EagerAggregation@neo4j	u1, collect(f) AS followed	1	1	0	760	0/0
+OptionalExpand(All)@neo4j	(u1)-[anon_0:FOLLOW]->(f) WHERE f:Author	1	3	13		0/0
+NodeUniqueIndexSeek@neo4j	UNIQUE u1:User(username) WHERE username = $autostring_0	1	1	2		0/0

42 rows
ready to start consuming query after 1 ms, results consumed after another 99 ms

With user_username and book_id constrain

Plan	Statement	Version	Planner	Runtime	Time	DbHits	Rows	Memory (Bytes)
"PROFILE"	"READ_ONLY"	"CYPHER 4.4"	"COST"	"INTERPRETED"	3	426	42	11368

Operator	Details	Estimated Rows	Rows	DB Hits	Memory (Bytes)	Page Cache Hits/Misses
+ProduceResults@neo4j	`a2.id`, `a2.name`, `a2.username`	1	42	0		0/0
+Distinct@neo4j	a2.id AS `a2.id`, a2.name AS `a2.name`, a2.username AS `a2.username`	1	42	180	10840	0/0
+Filter@neo4j	not a2 IN followed AND not a1 = a2 AND not u1 = a2 AND not anon_1 = anon_3 AND not anon_2 = anon_3 AND a2:Author	1	60	60	120	0/0
+Expand(All)@neo4j	(b2)<-[anon_3:WROTE]-(a2)	4	86	112		0/0
+Filter@neo4j	not b1 = b2 AND not anon_1 = anon_2 AND b2:Book	3	26	26		0/0
+Expand(All)@neo4j	(a1)-[anon_2:WROTE]->(b2)	4	27	28		0/0
+Filter@neo4j	a1:Author	1	1	1		0/0
+Expand(All)@neo4j	(b1)<-[anon_1:WROTE]-(a1)	1	1	2		0/0
+Apply@neo4j		1	1	0		0/0
+NodeUniqueIndexSeek@neo4j	UNIQUE b1:Book(id) WHERE id = $autostring_1	1	1	2		0/0
+EagerAggregation@neo4j	u1, collect(f) AS followed	1	1	0	760	0/0
+OptionalExpand(All)@neo4j	(u1)-[anon_0:FOLLOW]->(f) WHERE f:Author	1	3	13		0/0
+NodeUniqueIndexSeek@neo4j	UNIQUE u1:User(username) WHERE username = $autostring_0	1	1	2		0/0

42 rows
ready to start consuming query after 2 ms, results consumed after another 1 ms

Mongo Index Analysis

We decided to use indexes in order to speed up the application. We performed some test to measure the speed improvement obtained by using indexes. This test is carried out using the *explain()* function offered by MongoDB.
All operations have been tested on virtual machines.

Index Name	Index Type	Collection	Attributes
Username_author	Unique	Author	username
Username_user	Unique	Users	username
Book_ID	Unique	Books	Book_id
Book_genres		Books	genres
Book_title	Compound	Books	title
Find_review		Books	reviews.review_id

Authors and Users Collections test

We consider introducing the index on the attributes “username” in the authors and users collections to improve operations that require it.

For example we have the login operations:

- Login as author

```
db.authors.find(
                {
                    username: "Mattiax"
                }
)
```

Index	Documents returned	Index keys examined	Documents examined	Execution Time (ms)
False	1	0	59605	218
True	1	1	1	0

- Login as user

```
db.users.find(
        {
            username: "Mattiax"
        }
)
```

Index	Documents returned	Index keys examined	Documents examined	Execution Time (ms)
False	1	0	150985	694
True	1	1	1	0

Books Collection test

We consider introducing the index on the attributes “genres” in the books collection to improve operations that filter book by genres.

For example, we have the search by genre operation:

- Find book by genre

```
db.books.find(
        {
            genres: "History"
        }
)
```

Index	Documents returned	Index keys examined	Documents examined	Execution Time (ms)
False	29948	0	131111	226
True	29948	29948	29948	85

We consider to introduce also the index on the attributes “reviews.review_id” in the books collection to improve operations that return review document.

For example, we have the research operation of review by id:

- Find review by review_id

```
db.books.find(
                    {
                     "reviews.review_id": "45a0abe1dd897691f7af896cb22d1556"
                    }
)
```

Index	Documents returned	Index keys examined	Documents examined	Execution Time (ms)
False	1	0	131110	523
True	1	1	1	0

We consider introducing also the index on the attribute “title” in the books collection to improve operations that filter books by title.

For example, we have the research operation by title:

- Find book by title

```
/* -- with title index -- */
db.books.find(
        {$text:
                {
                $search: "harry",
                $caseSensitive: false
                }
        }
)
```

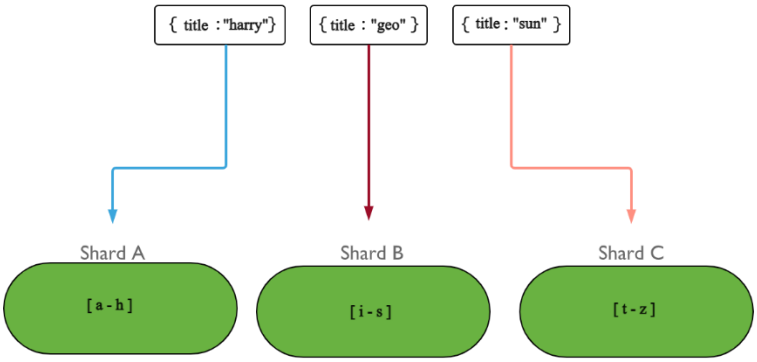
```
/* -- without title index -- */
db.books.find(
        {
         title: {
                $regex: /.*harry.*/i
                }
        }
)
```

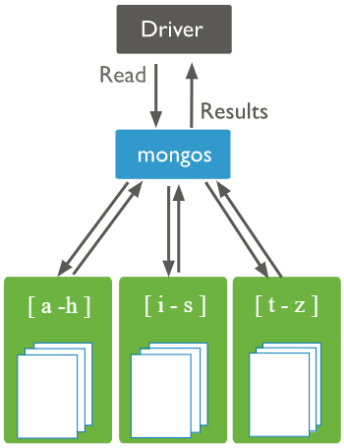
Index	Documents returned	Index keys examined	Documents examined	Execution Time (ms)
False	331	0	131112	285
True	331	331	331	2

Sharding proposal

We propose to implement a *Range Based Strategy* about the dataset portion formed by the largest collections Users, Authors and Books to speed up reading operations.
This idea is based on the objective of speeding up document search operations; these operations are performed on username field for Users and Authors collections and on title field for Books collection.

We decided to split the dataset portion in 3 chunks, each chunk contains a dataset portion defined by a range. The ranges are *[a-h], [i-s],[t-z]*.





We also think to use 3 replicas set for each shard in order to avoid failures and maintain high availability.