BOOKBASE

Alexander Chea

Daniel Colson

Karishma Jain

Vishrutha Konappa Reddy

INTRODUCTION AND PROJECT GOAL.

GoodReads.com is a popular social media website centered around users' engagement with books, with users being able to access GoodRead's databases of books, book recommendations, and connect with other users of the website. The goal for our project, BookBase, was to rebuild some of the functionality of GoodReads using several very large (25GB+) GoodReads academic datasets we found. Given the size of the datasets, we decided to try using NoSQL with MongoDB for our backend due to the possibility of better performance through sharding and taking advantage of map reduce. To augment these datasets and to get experience with data cleaning and entity resolution, we also decided to scrape Reddit's /r/books community to get a second set of recommendations for books.

ARCHITECTURE

BookBase is built using the MEAN stack. The MEAN stack is composed of MongoDB for the database, Express for the server framework, Angular for the front-end, and Node.js for providing a javascript runtime environment. Our data is stored on MongoDB's Atlas Cloud. Atlas Cloud has a 512MB maximum limit on data hosted in its free version so the current data sets hosted there are a subset of the full original datasets.

Ultimately, we had some trouble implementing the full suite of features we hoped to achieve. Nonetheless, we did succeed at giving users flexible access to the book databases. On Bookbase, users can:

- search for the title of a book which will bring up a list of all books with that title and their authors. The user can then navigate to the intended book.
- visit the page of a book which includes the book title, author(s), the other books in the book's series (if there are others), reviews of the book, number of user ratings, genres of the book, and the book cover image.
- visit the page of an author which includes their name, books by the author, the average rating of the author, and the number of people that have rated the author.
- navigate to a list of genres. Selecting a genre will reveal the top fifteen books for that genre according to users' ratings.
- search the text of comments from Reddit's /r/books weekly book recommendations thread and view matching comments ordered by the number of upvotes that comment received. Users can then click buttons to view the parent of a comment or the children of a comment. This way, users can find people discussing recommendations related to some topic the user is interested in, for example an author or title of a book.

DATA

I. Our first set of datasets come from the UCSD Book Graph, an academic database scraped from GoodReads. This database contains the meta-data for 2.36M books and over 229M user-book interactions like adding books to lists. There are also 15M text reviews of books. The original datasets can be found here: https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home?authuser=0. Subsets of these datasets are hosted on our Cloud Atlas server in six different collections: authors, book_series, books, genres, reviews, and user_interactions.

- **authors collection**: This collection contains information of an author. It has fields such as author_id(which was used in several of our aggregate-lookup queries), name, average_rating, ratings_count(total ratings given to the works of the author) amongst others. There are 10,655 documents in the collection.
- **books collection**: Out main dataset which contains the bulk of the information regarding a book. It has a total of 29 fields, some of which have embedded documents or arrays as value. Some of the important fields(apart from the usual book_id, title etc) that were consequently used to join with other datasets are series_id(id of a series if a book is part of a series) and author_id. There are 21,908 book documents in the collection.
- **book_series collection**: This collection contains information about books series. It has fields such as series_id, title, description, series_works_count(total books in that series). The books collection contains a field series, which is used to join on books_series's series_id to find other books in a particular series. There are 1,831 series documents in the collection.
- **genres collection**: This collection has only two fields, book_id and genres. The genres field has a value which is a JSON array with genres a book can have as the key and value as the number of votes(from users) that particular genre gets for that particular book. This was a particularly tricky collection to work with because the keys are all unique to each book and hence could not be referenced or called the way we usually would. There are 674 documents listing all genres of a book.
- **reviews collection**: The reviews collection holds data regarding a review given by a particular user to a particular book. It has fields like book_id, user_id, review_id, all of which are unique and fields like review_text(text review), rating, n_votes, n_comments(number of votes and comments on that review) amongst others.

There are 62,267 review documents in the collection.

- **user_interactions collection**: What could be said as the second most important dataset after books. This collection holds information of a particular user's interaction with one or more books, such as if the user has read a book(is_read), the rating(if given) to the book, the date the book was read apart from the usual book_id, user_id and review_id. There are 41,872 user interaction documents in the collection.

These GoodReads datasets serve as the backbone of our project, as they provide the information we use to populate our basic author and book pages. We combine the genres and books databases to show the user a book's genre on the book page, and likewise for reviews and user_interactions to collect text reviews and ratings of the book. On the authors page, we combine the authors with lists of their books and also with author ratings from user_interactions.

II. Our second dataset comes from comments that we scraped from https://www.reddit.com/r/books/. Every Friday, /r/books has a book recommendations thread (for example, the last one was on December 6th: https://www.reddit.com/r/books/comments/e3cdvi/weekly_recommendation_thread_november_29_2019/). We wrote a scraper using praw (a Python wrapper for the Reddit API) and successfully collected 63,608 comments from weekly threads since 2015 (before which, book recommendations were scattered in top-level posts). For the code we wrote for the scraper, see Appendix A. For each comment, we kept:

- the comment id,
- the id of the comment's parent,
- the title of the top-post for the week it was submitted,
- the comment author's name,
- the time the comment was created,
- the number of upvotes the comment received,
- and the body text of the comment.

These are the fields in a single collection hosted on our Cloud Atlas server. Keeping the ids of comments is important, having the parent id stored allows us to rebuild the original comment trees. The comment ids and author names also have the option of being used to make further calls to the Reddit API real-time, if desired.

DATABASE

I. For the GoodReads data, we found that the datasets were already relatively clean (they had clearly been cleaned by their publishers). As such, we did not attempt any entity resolution for the GoodReads datasets. We pruned each of the very large GoodReads datasets to be in the tens-of-thousands-of-rows range in order to be able to host the files on Cloud Atlas. The files were originally formatted as JSON arrays, so we were able to take advantage of the MongoDB's mongoimport tool to bulk import the documents.

As we decided to use a NoSQL database for our project, there aren't applicable ER diagrams, nor were the collections normalized into any particular normal form.

II. For the scraped Reddit data, we wrote our scraper such that it formatted the scrapped information into a JSON array such that we could use mongoimport. This led to several issues due to the body text of comments containing special characters that would break the correct formatting of JSON arrays (for example, new line characters). Some of these issues were handled during the scraping process, others were handled using regex searches after the fact to find problematic entries.

One issue we encountered was that we originally captured the upvotes for each comment as a string as opposed to an int, which caused it to not be possible to sort comments by their number of upvotes. We corrected this by converting the string to an int in javascript and re-saving the document in the Mongo shell.

PERFORMANCE EVALUATION

The MySQL query to fetch the top books and corresponding authors for a particular genre is being used to compare and evaluate the performance of both MySQL and NoSQL. The same query was reproduced using MongoDB and the time taken to fetch results were compared.

The query is designed to first join the books and authors dataset, in order to get the author's name. Then a join with genres dataset is done to filter out only the required genre. The result obtained is then ordered in decreasing order of average rating given to the book.

In the first go of the query a full run of the above tables were done, i.e a cartesian product was taken and then, only tuples matching the criteria was selected for the next join. Since the tables used have at least 450k rows each, taking product and then selecting the appropriate rows was an expensive task and took about 9.68s to fetch results.In order to optimize, the query was restructured to have left deep joins and only the needed fields were projected at each stage. An index on book_id in books table and author_id in authors table was also created which brought down the execution time to 2.38s.

The same query was written in NoSQL. Without optimization and the time taken to fetch results was observed to be around 10.7s. Creating index on book_id in both books and genre collections and projecting only required fields reduced the time to 1.68s. It was observed that although without optimization MySQL performed better, after creating of indexes in Mongodb collections significantly reduced the execution time to be lower than that of MySQL

TECHNICAL CHALLENGES

The dataset that was used to populate our database is in JSON format, some of which have complicated document structures. Storing JSON data in MySQL is not easy nor is it efficient. For example, the books dataset's "author" field is a JSON array. In order to save the data, we'd have to unwind and save the data, which leads to a lot of rows with duplicate data or we'd have to have a lot of attributes. This was the main motivation behind having all our data in NoSQL over MySQL.

Secondly, joining in Mongodb is not as easy or organic as in MySQL. Queries involving joins were challenging and complicated to write. Lookup was used to perform joins. Things were made more complicated due to the structure of the data. For example, the different genres a book might have was stored as a JSON Array, with the structure unlike anyone would expect. The genres(like fiction, mystery) were the keys as opposed to what one would usually expect it to be the values of a field. Extraction of such data was a huge challenge because there was no fixed key through which we could access it.

Third, the front-end development process was often challenging for us due to no one having a strong grasp of using Angular. While we were able to take advantage of the starter code base from Homework 2 and the code patterns established there, there were a number of issues that we encountered with Angular that we couldn't figure out. For example, much of our current code exists in a singular Angular controller. While this is far from ideal in terms of performance, moving the HTTP calls to different controllers and changing the relevant controller in the HTML file resulted in the HTTP calls failing to run on the relevant page.

QUERIES

1. Given a given title of a book, we find the information for the author of that book. We use this to, for example, get the name of the author for a book page.

```
db.books.aggregate(
  {
      $match: {
          title: ”<book title>”
          }
  }, {
      $lookup: {
          from: "authors",
          localField: "authors.author_id",
          foreignField: "author_id",
          as: "author_id"
      }
  }, {
      $project: {
          _id: 0,
          author_id: 1
      }
  }
)
```

2. We use map-reduce to get the distribution of ratings and count of it for a particular book. This is to provide summary statistics for books on book pages.

```
var mapFunction1 = function(){
      var ckey = this.rating;
      var cvalue = 1;
      emit(ckey, cvalue);
};

var reduceFunction1 = function(keyRating, countRating){
      count_r = Array.sum(countRating);
      return count_r
};

db.user_interactions.mapReduce(mapFunction1, reduceFunction1,
            {out: {inline: 1},
            query:{book_id:"1384"}}
);
```

3. The authors page displays the series that an author has written. This query is used to collect all books in a series to be displayed together:

```
db.books.aggregate(
        {
            $match: {
```

```
                isbn: "<book_isbn>"
                }
        }, {
            $lookup: {
                from: "books",
                localField: "authors.author_id",
                foreignField: "authors.author_id",
                as: "series"
            }
        }, {
            $unwind: "$series"
        }, {
            $project: {
                _id: 0,
                book: "$series.series"
            }
        }
)
```

4. Get all books by a particular author. Used for populating the list of books by an author on the author page.

```
titles = []
db.books.aggregate(
        {
        $match: {
            authors.author_id:"18540"
            }
        }, {
            $project: {
                _id:0,
                title: "$title"
            }
        }, {
            $unwind: "$title"
        }
).toArray().forEach(function(values){titles.push(values.title)});
```

5. To get top 15 book titles based on the genre that the user has clicked on:

```
db.books.aggregate([
  {
    $lookup: {
      from: "genres",
```

```
        localField: "book_id",
        foreignField: "book_id",
        as: "book_id"
      },
    },
    {
       $project: {
          _id: 0,
          genres_obj: { $arrayElemAt: ["$book_id", 0] },
          average_rating: 1,
          title: 1,
          ratings_count: 1,
          text_reviews_count: 1
       }
    },
    {
       $project: {
          _id: 0,
          genres_arr: { $objectToArray: "$genres_obj.genres" },
          average_rating: 1,
          title: 1,
          ratings_count: 1,
          text_reviews_count: 1
       }
    },
    {
      $match: {
        genres_arr: {
          $elemMatch: { k: "fiction" },
        },
      }
    },
    {
      $sort: { average_rating: -1 }
    },
    {
      $limit: 15
    }
])
```

6. Given the title of a book, get the titles of the other books in the series. This is used for displaying series on author and book pages.

```
db.books.aggregate(
        {
            $match: {
```

```
                title: "Desiderata"
            }
        }, {
            $lookup: {
                from: "books",
                localField: "series",
                foreignField: "series",
                as: "series"
            }
        }, {
            $project: {
                _id: 0,
                title: "$series.title"
            }
        }
)
```

7. Get all the books a user has read, rated, or reviewed.

```
var b_id = []
db.user_interactions.aggregate(
    {$match:
        {
            user_id:"8842281e1d1347389f2ab93d60773d4d"
        }
    }, {
        $project: {
            _id:0,
            book_id: "$book_id"
        }
    }
).toArray().forEach(function(key){book_id.push(key.book_id)}
```

8. Query to get all the genres of a book known title

```
db.genres.aggregate(
    {$match: {
        book_id: {$in: ["7327624"]}
        }
    }, {
```

```
            $project: {
                _id:0,
                genres:1
            }
        }
).toArray().forEach(function(key){arr.push(key.genres)})
```

9. Find all distinct genres that exist. This was necessary for creating the pages for the top-15 books in each genre.

```
db.genres.aggregate([
  {
     $project: {
        _id: 0,
        genres_arr: { $objectToArray: "$genres" }
     }
  },
  {
    $unwind: "$genres_arr"
  },
  {
    $group: {
      _id: "$genres_arr.k", count: { $sum: 1 }
    }
  },
  {
    $sort: {count: -1}
  }
])
```

10. FInd the the top 15 authors overall. This is used for the “Top Authors” page.

```
db.authors.aggregate([
    {$group: {_id:0, names: {$push: {name: "$name" }}}},
      {
            $sort: {average_rating: -1}
```

```
        },
        {
                $limit: 15
        },
])
```

11. This query allows us to search Reddit comment body texts for a particular search phrase. Unfortunately, we could not get this query to work in the web app.

```
db.scraped_reddit_data.find(
    {
        $text: {$search: "Harry Potter"}
    }
)
```

12. This query returns the comments from a given week's recommendation thread sorted by number of upvotes. As mentioned above, we were unfortunately not able to use this query due to what seems to be a limit set by the node mongodriver on the number of documents that can be returned by a query.

```
db.scraped_reddit_data.aggregate([
        {
          $match: {submission_title: "Weekly Recommendation Thread: <MONTH>
<DD>, <YYYY>"}
        }
        {
          $sort: {score: -1}
        }
])
```

13. Given a book_id, this finds all reviews for that book. We use this to display the reviews of a book on a book's page.

```
db.books.aggregate([
          {
            $match: {
              title:"The True Blue Scouts of Sugar Man Swamp"
              }
```

```
    }, {
      $lookup: {
        from: "reviews",
        localField: "book_id",
        foreignField: "book_id",
        as: "comments"
      }
    }
  ]);
```

14. This query suggests other books read by users having read a particular book. The main dataset is the user_interactions collection. Here, all users who have read book X are considered and then their respective lists of books read is obtained. Then the count of books from all users are taken and sorted in decreasing order of occurrence of a book. The result returned is a list of book titles.

```
db.user_interactions.aggregate(
    {
        $match:
        {
            book_id: "1384"
        }
    }, {
        $project: {
            _id: 0,
            user_id: "$user_id",
            book_ids: "$book_id",
            book_title: "title"
        }
    }, {
        $lookup: {
          from: "user_interactions",
          localField: "user_id",
          foreignField: "user_id",
          as: "other_users_books"
        }
    }, {
        $unwind: "$other_users_books"
    }, {
        $group: {
            _id: "$other_users_books.book_id",
            count: {$sum: 1}
        }
```

```
    }, {
        $project: {
            _id: 0,
            book_id: "$_id",
            count: "$count"
        }
    }, {
        $sort: {
            count: -1
        }
    }, {
        $limit: 20
    }
);
```

15. Below is a query used to recommend books based on a users past reading activity. First, all the books a particular user has read is taken and genres of each book is obtained. Next, the count of occurrence of each genre is found and then ordered in decreasing order. Query result is an ordered list of genres. For each genre, top books are fetched.

```
db.user_interactions.aggregate(
    {
        $match: {
            user_id: "8842281e1d1347389f2ab93d60773d4d"
        }
    }, {
        $project: {
            _id:0,
            book_id: 1
        }
    }, {
        $lookup: {
          from: "genres",
          localField: "book_id",
          foreignField: "book_id",
          as: "books_genres"
        }
    }, {
        $project: {
            _id:0,
            genres_obj: {$arrayElemAt: ["$books_genres", 0]},
        }
    }, {
        $project: {
            _id:0,
            genres_arr: {$objectToArray: "$genres_obj.genres"}
        }
    }, {
```

```
            $unwind: "$genres_arr"
        }, {
            $project: {
                _id:0,
                genres: "$genres_arr.k",
                count: "$genres_arr.v"
            }
        }, {
            $group: {
                _id: "$genres",
                count: {$sum: {$toInt: "$count"}}
            }
        }, {
            $project: {
                _id:0,
                genre: "$_id",
                total_count: "$count"
            }
        }, {
            $sort: {total_count: -1}
        }, {
            $group: {
                _id: "$genres",
                genre_array: {$push: "$genre"}
            }
        }, {
            $project: {
                _id: 0,
                genre_array: 1
            }
        }
    );
```

Appendix A: Reddit Scraper Code

```
# PRAW tutorial link: https://www.youtube.com/watch?v=NRgfgtzIhBQ

# PRAW documentation: https://praw.readthedocs.io/en/latest/index.html

# Weekly recommendation threads on Reddit:

https://www.reddit.com/r/books/search?q=weekly+recommendation&restrict_sr=
on&sort=new&t=all


import praw

import inspect


reddit = praw.Reddit(client_id = 'j_3XCr61F-oIUg',

                     client_secret = '0z902fQEULquYtC11SqdxGb6VWI',

                     username = 'cis450project',

                     password =
'gxChby9xwsFhp6CBjjkgrnCLKhqVkY6ThSMiWWgY',

                     user_agent = 'cochrane')


subreddit = reddit.subreddit('books')


weekly_thread = subreddit.search('title:Weekly Recommendation Thread,
selftext:Welcome to our weekly recommendation '

                                 'thread!', sort='new', limit=1000)


file = open("scraped_reddit_data.txt", "w+")
```

```
for submission in weekly_thread:

    # replace the "MoreComments" objects with the comments they represent
    submission.comments.replace_more(limit=0)

    comments = submission.comments.list()
    for comment in comments:

        # need to escape any special characters in the body of the message
        body = str(comment.body)
        metaCharacters = ["\\", "$", "'", '"', "\n"];
        for i in range(len(metaCharacters)):
            body = body.replace(metaCharacters[i], ' ')

        string = "{ \"id\": \"" + str(comment.id) + "\", \"parent_id\":
\"" + str(comment.parent_id) + \
                "\", \"submission_title\": \"" + str(submission.title) +
"\", \"author\": \"" + str(comment.author) + \
                "\", \"created_utc\": \"" + str(comment.created_utc) +
"\", \"score\": \"" + str(comment.score) + \
                "\", \"body\": \"" + body + "\"},\n"
        file.write(string)

file.close()
```