GUTENBERG PROJECT REPORT

Made by Murched Kayed, Hallur vid Neyst & Zaeem Shafiq

MAY 30, 2019

COPENHAGEN BUSINESS ACADEMY

Table of Contents

Introduction

This report is about the solution we made for the Gutenberg project. We will, among other things, write about which databases and technologies we used to solve it. Furthermore, the report describes how we imported and structured the data in the databases.

Databases used

We have chosen to make use of two different kinds of databases: one relational database and one NoSQL database. We chose MySQL as the relational database and MongoDB as the NoSQL database.

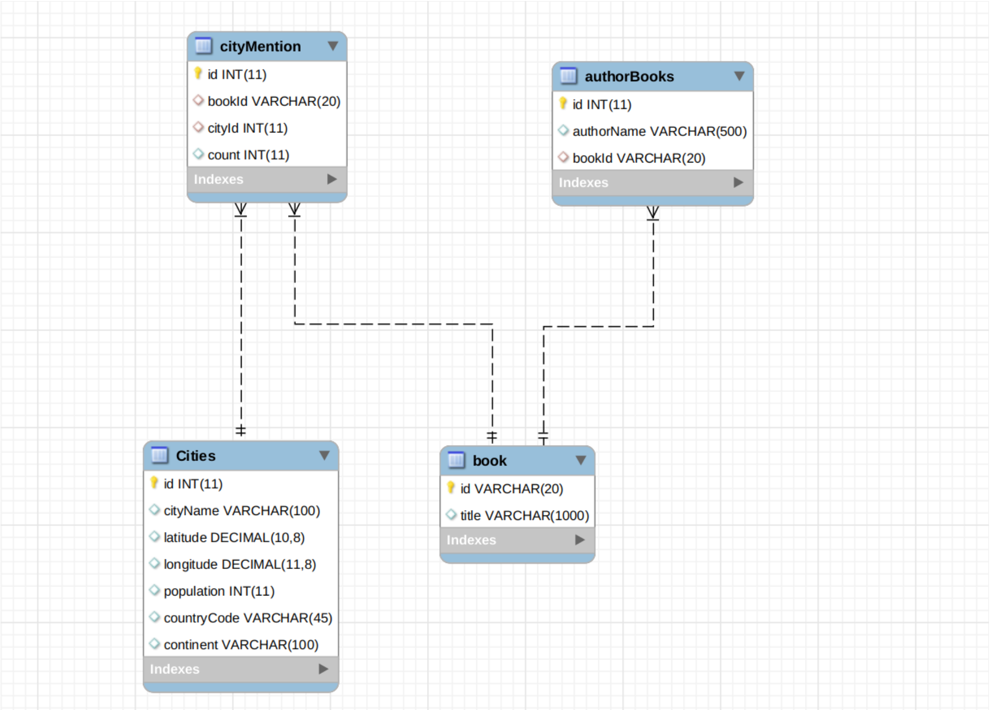
Programming language used

We agreed as a group that we will use Java to solve this project, since we all three had good skills in Java. To make it easier to handle dependencies, we have made our project using Maven. We have chosen to focus on the database part of the project and therefore just used a simple Command-line interface as our frontend.

Data modeling in the database

MySQL

We have chosen to structure our tables in MySQL as follows:

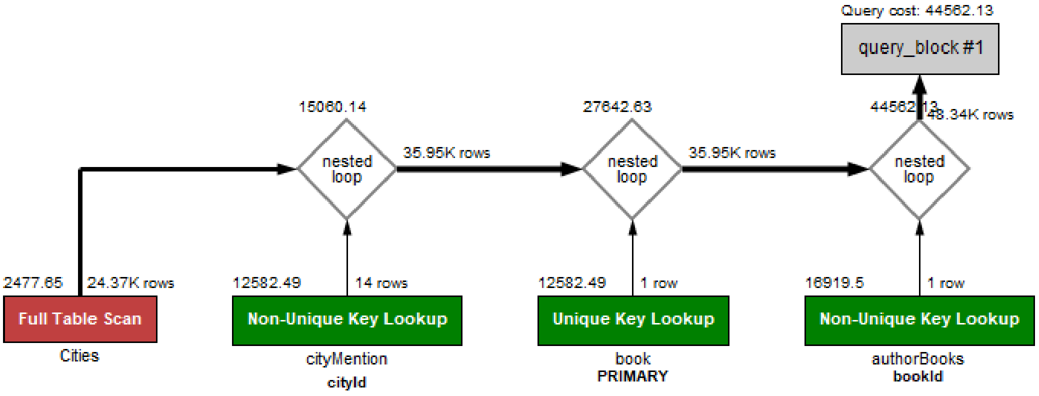


Indexes

To optimize performance, we have chosen to use indexes on some of our columns in the database. This has proved to be extremely useful and given us a much faster performance on our queries.

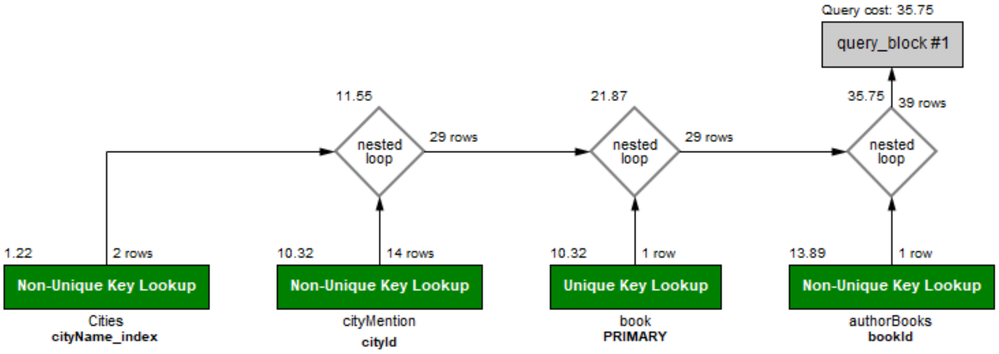
Below are a number of examples of the graphical execution plan where we make use of the same query both before and after we have used indexes.

Before creating the index *cityName_index*:

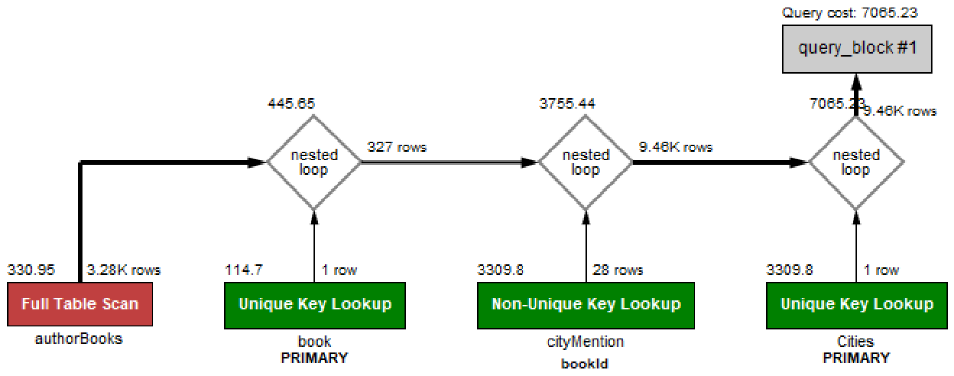


After creating the index:

```
create index cityName_index on Cities(cityName);
```

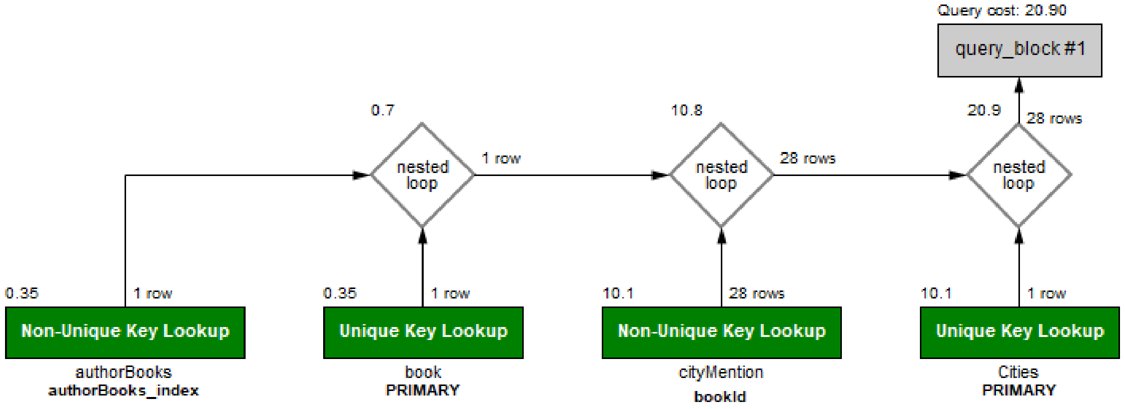


Before creating the index *authorName_index*:



After creating the index:

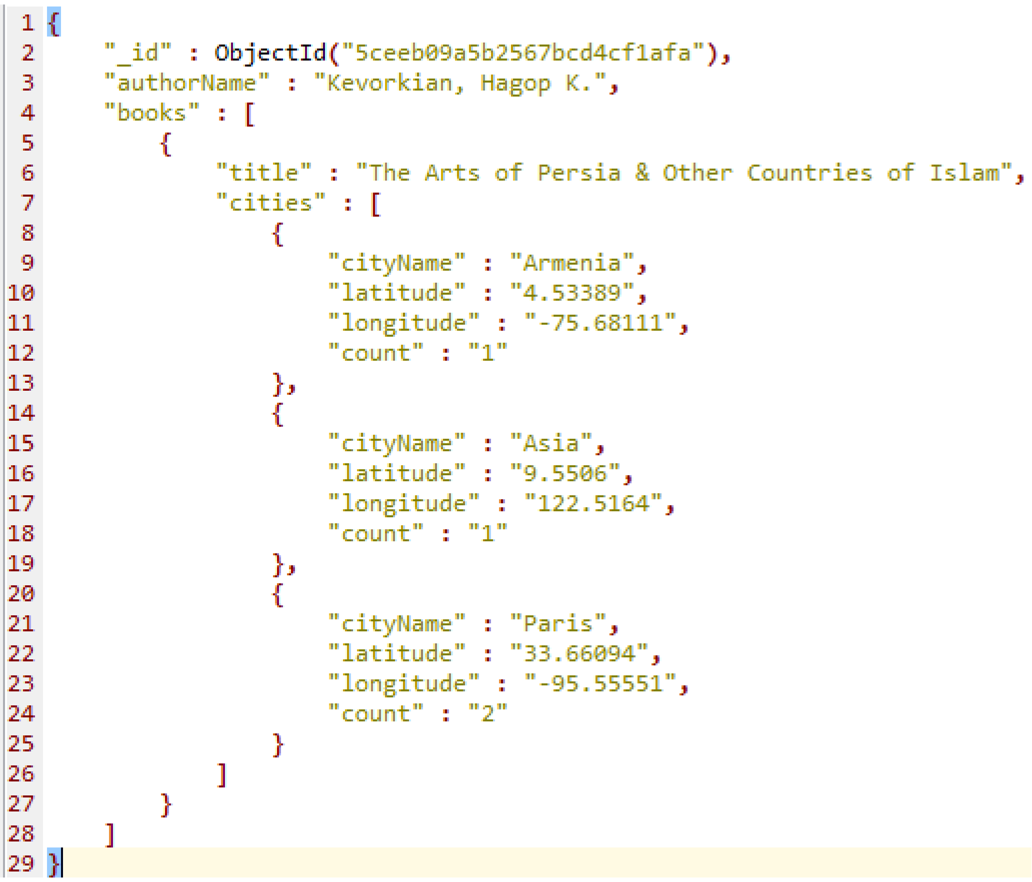
```
create index authorBooks_index on authorBooks(authorName);
```



As it can be seen above, our performance has been drastically improved after we implemented indexes in our MySQL database. We avoid making a Full Table Scan, which turns out to save us a lot of time. E.g. in one of the examples above the query cost goes from 44562.13 to only 35.75. This is a great improvement.

MongoDB

Since MongoDB is a NoSQL database and hence doesn't contain relations, we have chosen a completely different structure for our MongoDB database. We have chosen to simply store all the data in one collection. Our collection is structured as follows:



```
{
    "_id" : ObjectId("5ceeb09a5b2567bcd4cf1afa"),
    "authorName" : "Kevorkian, Hagop K.",
    "books" : [
        {
            "title" : "The Arts of Persia & Other Countries of Islam",
            "cities" : [
                {
                    "cityName" : "Armenia",
                    "latitude" : "4.53389",
                    "longitude" : "-75.68111",
                    "count" : "1"
                },
                {
                    "cityName" : "Asia",
                    "latitude" : "9.5506",
                    "longitude" : "122.5164",
                    "count" : "1"
                },
                {
                    "cityName" : "Paris",
                    "latitude" : "33.66094",
                    "longitude" : "-95.55551",
                    "count" : "2"
                }
            ]
        }
    ]
}
```

As seen in the picture above, we only have one collection in our database. This collection follows the following structure: In the object is an author. An author contains an ID, a name, and a list of the books he/she has written. The list of books then contains a title and a list of the cities mentioned in that book. The list of cities contains different attributes about a city: the city's name, location and how many times the city is mentioned in the book (count). This simple structure has been followed throughout the whole collection.

Data modeling in the application

To have a better overview of our project, we have structured our classes into different packages as follows:

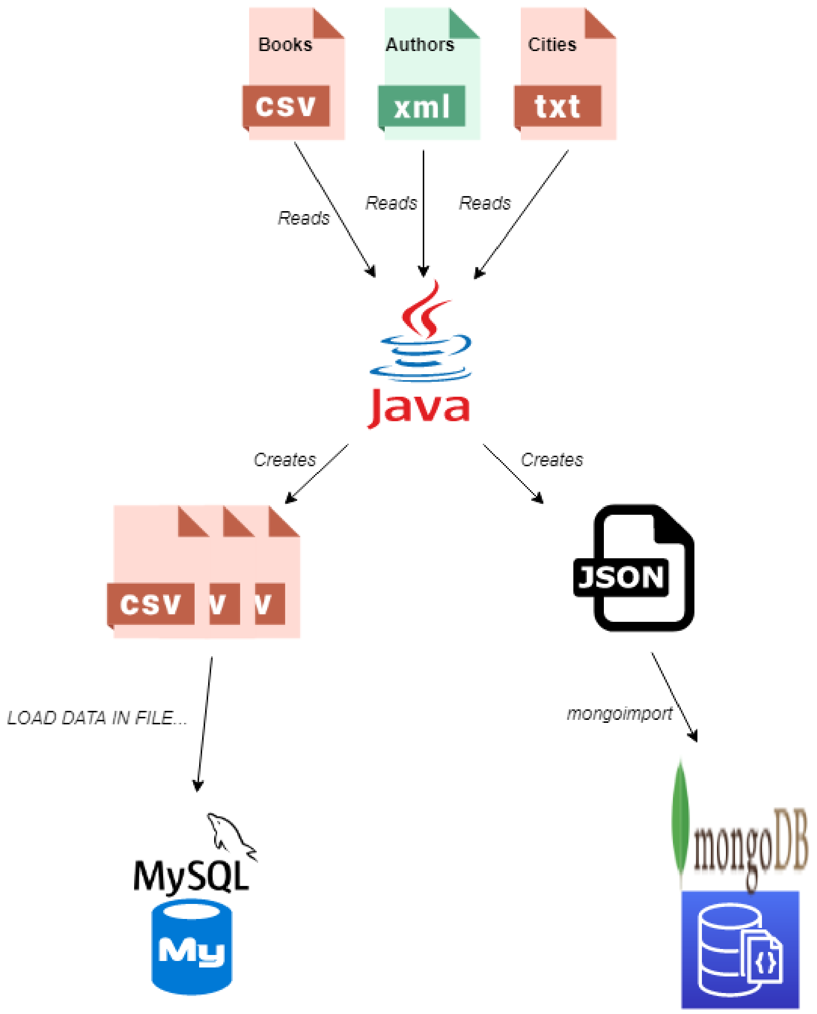
- ***entity***
    - This package simply contains all our entity classes with corresponding attributes.
- ***entitymanager***

- This package contains classes with methods that help to read books/cities from the txt/RDF files. It also contains methods that generate csv files with a header and all necessary attributes. These csv files can later be used to import data into our databases.

- ***files***
  - This package contains all the files we have used to create our databases and to insert data into our databases.
- ***main***
  - This package just contains our main method that runs our program.
- ***mongodb***
  - This package contains all communication between our program and the MongoDB database. That is, it contains both a connector that connects to the database and a number of methods that send the various queries to the MongoDB database.
- ***mysql***
  - This package contains all communication between our program and the MySQL database. That is, it contains both a connector that connects to the database and a number of methods that send the various queries to the MySQL database.
- ***threads***
  - This package contains eight threads classes. These have been made to optimize the process when looking for city names in our books (txt files). Since we worked with a lot of books/cities, the program was very slow as it had to read the files. We therefore chose to run eight threads at the same time, which proved to improve performance a lot.

Data import

Below is a picture of how we have imported the data in our program.



As it can be seen on the diagram above, we created a Java program to import the data into the databases. Our program works as follows:

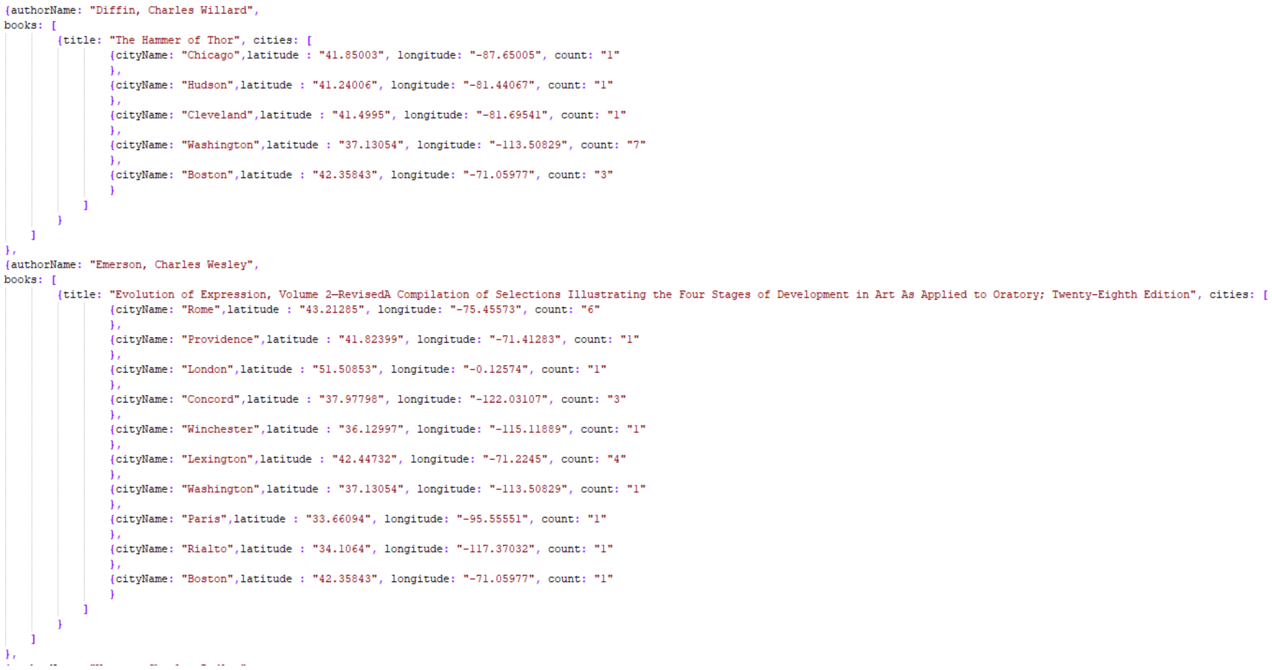
1. Our program starts by reading the books/cities from different files. This is done as follows:
   - The cities and their geolocations are read from the csv file found here: http://download.geonames.org/export/dump/cities15000.zip. For this we used a Java library called opencsv (http://opencsv.sourceforge.net/).
   - The author names and book titles are read from the XML/RDF files from the offline catalogue available here: https://www.gutenberg.org/wiki/Gutenberg:Feeds. For this we used a Java library called Apache Jena (https://jena.apache.org/documentation/io/rdf-input.html).

- The cities mentioned in a book are found in the txt files we downloaded from our droplet on Digitalocean which downloaded them from the Gutenberg webpage. To recognize the cities mentioned in a book, we used the Stanford Named Entity Recognizer (NER) in Java (https://nlp.stanford.edu/software/CRF-NER.html).

2. Then our program generates a CSV/JSON file that can be used to import the data into the databases. The files only contain the attributes we agreed were needed to solve the queries. This is done as follows:
    - We created csv files (separated by tab characters) that can be easily imported in MySQL. An example of how a part of a csv file looks like is (the first row is the header):

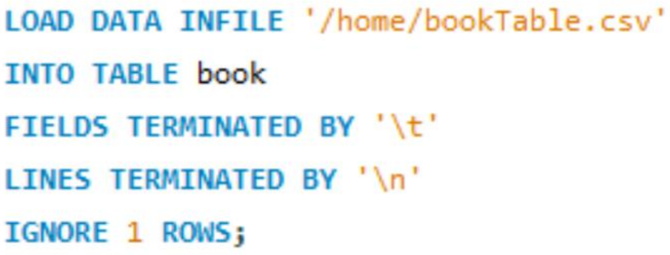
```
cityId  cityName        latitude        longitude       population      countryCode     continent
3040051 les Escaldes    42.50729        1.53414 15853   AD      Europe/Andorra
3041563 Andorra la Vella        42.50779        1.52109 20430   AD      Europe/Andorra
290594  Umm al Qaywayn  25.56473        55.55517        44411   AE      Asia/Dubai
291074  Ras al-Khaimah  25.78953        55.9432 115949  AE      Asia/Dubai
291696  Khawr Fakkān    25.33132        56.34199        33575   AE      Asia/Dubai
292223  Dubai   25.0657 55.17128        1137347 AE      Asia/Dubai
292231  Dibba Al-Fujairah       25.59246        56.26176        30000   AE      Asia/Dubai
292239  Dibba Al-Hisn   25.61955        56.27291        26395   AE      Asia/Dubai
292672  Sharjah 25.33737        55.41206        1324473 AE      Asia/Dubai
292688  Ar Ruways       24.11028        52.73056        16000   AE      Asia/Dubai
292878  Al Fujayrah     25.11641        56.34141        62415   AE      Asia/Dubai
292913  Al Ain  24.19167        55.76056        408733  AE      Asia/Dubai
292932  Ajman   25.41111        55.43504        226172  AE      Asia/Dubai
292953  Adh Dhayd       25.28812        55.88157        24716   AE      Asia/Dubai
292968  Abu Dhabi       24.46667        54.36667        603492  AE      Asia/Dubai
12042053        Musaffah        24.35893        54.48267        243341  AE      Asia/Dubai
1120985 Zaranj  30.95962        61.86037        49851   AF      Asia/Kabul
1123004 Taloqan 36.73605        69.53451        64256   AF      Asia/Kabul
1125155 Shīndand        33.30294        62.1474 29264   AF      Asia/Kabul
1125444 Shibirghān      36.66757        65.7529 55641   AF      Asia/Kabul
1125896 Shahrak 34.10737        64.3052 15967   AF      Asia/Kabul
1127110 Sar-e Pul       36.21544        65.93249        52121   AF      Asia/Kabul
1127628 Sang-e Chārak   35.84972        66.43694        15377   AF      Asia/Kabul
1127768 Aībak   36.26468        68.01551        47823   AF      Asia/Kabul
1128265 Rustāq  37.12604        69.83045        25636   AF      Asia/Kabul
1129516 Qarqīn  37.41853        66.04358        15018   AF      Asia/Kabul
1129648 Qarāwul 37.21959        68.7802 24544   AF      Asia/Kabul
1130490 Pul-e Khumrī    35.94458        68.71512        56369   AF      Asia/Kabul
1131316 Paghmān 34.58787        68.95091        49157   AF      Asia/Kabul
1132495 Nahrīn  36.0649 69.13343        22363   AF      Asia/Kabul
```

- We created a JSON file that can be easily imported in MongoDB. An example of how a part of the JSON file looks like is:



```
{authorName: "Diffin, Charles Willard",
books: [
        {title: "The Hammer of Thor", cities: [
                {cityName: "Chicago",latitude : "41.85003", longitude: "-87.65005", count: "1"
                },
                {cityName: "Hudson",latitude : "41.24006", longitude: "-81.44067", count: "1"
                },
                {cityName: "Cleveland",latitude : "41.4995", longitude: "-81.69541", count: "1"
                },
                {cityName: "Washington",latitude : "37.13054", longitude: "-113.50829", count: "7"
                },
                {cityName: "Boston",latitude : "42.35843", longitude: "-71.05977", count: "3"
                }
            ]
        }
    ]
},
{authorName: "Emerson, Charles Wesley",
books: [
        {title: "Evolution of Expression, Volume 2—RevisedA Compilation of Selections Illustrating the Four Stages of Development in Art As Applied to Oratory; Twenty-Eighth Edition", cities: [
                {cityName: "Rome",latitude : "43.21285", longitude: "-75.45573", count: "6"
                },
                {cityName: "Providence",latitude : "41.82399", longitude: "-71.41283", count: "1"
                },
                {cityName: "London",latitude : "51.50853", longitude: "-0.12574", count: "1"
                },
                {cityName: "Concord",latitude : "37.97798", longitude: "-122.03107", count: "3"
                },
                {cityName: "Winchester",latitude : "36.12997", longitude: "-115.11889", count: "1"
                },
                {cityName: "Lexington",latitude : "42.44732", longitude: "-71.2245", count: "4"
                },
                {cityName: "Washington",latitude : "37.13054", longitude: "-113.50829", count: "1"
                },
                {cityName: "Paris",latitude : "33.66094", longitude: "-95.55551", count: "1"
                },
                {cityName: "Rialto",latitude : "34.1064", longitude: "-117.37032", count: "1"
                },
                {cityName: "Boston",latitude : "42.35843", longitude: "-71.05977", count: "1"
                }
            ]
        }
    ]
},
```

3. Then, via our program, we can easily insert the data into our databases by using the generated files. This is done as follows:
    - In MySQL the data can be imported using the following query:



```
LOAD DATA INFILE '/home/bookTable.csv'
INTO TABLE book
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

    - In MongoDB the data can be imported using the following command:

```
mongoimport --db your db Name --collection authors --file authorsJson.json --jsonArray
```

Behavior of query test set

Query runtime influenced by the database vs. the application frontend

It turns out that there is a difference in how long it takes to execute a query if you choose to run it directly in MySQL Workbench instead of running it through the program. This is because, among other things, Java must first connect to the database using a JDBC driver, whereas MySQL Workbench already has a connection to the database and thus can execute the query faster. The same goes for our MongoDB queries. For example, Robo 3T executes the queries faster than Java. In addition, we also found out that MySQL Workbench uses a default limit of 1000 rows, whereas Java doesn't have any limit. This of course also made our queries run much faster in MySQL Workbench.

Conclusion

All in all, the whole development process has been extremely instructive for all of us. We experienced a lot of exciting challenges along the way and it has also given us the opportunity to try a lot of things on our own. We have become much wiser on the use of indexes and seen how much importance they can have for the query performance when working with a lot of data. In addition, we have strengthened our competences in reading from and writing to files of various formats.

Which database should be used in such a project for production?

Since we did not use the same tables in both our databases, we actually had a good performance in both databases. If we had used the same structure in MongoDB, which we used in MySQL, our queries in MongoDB would have become a lot slower. This is due to the fact that we would have had four tables in MongoDB, which means that we would have to use joins even though MongoDB was not originally designed for joining collections. There is no doubt that both databases have their pros and cons, and which one is the best, greatly depends on what you want to store in the database.

If we were to recommend a database for a project like Gutenberg, we would recommend MongoDB. This is partly due to the fact that we could make much simpler queries in MongoDB, as we did not need to join with other collections. In addition, this project contained a lot of work with cities and their locations and here MongoDB also has the great advantage that you can make use of the Geospatial Index to store city locations. Furthermore, JSON is used in MongoDB, which is also a great advantage.