

Ann-Marie Czuboka, 40209452
Isabelle Czuboka, 40209525
Andy Chhuon, 40199798
Camila-Paz Vejar-Rojas, 40208489
Due Date: April 15th, 2024

Overview Report

Overview of system

Our system integrates data from two distinct sources: one comprising data gathered through an API, and the other comprising direct CSV imports. The system is organized into five collections. It is an overview of language supply and demand within repositories and job postings. We are using the GitHub API to collect the information on repositories and a CSV from Kaggle that contains thousands of GlassDoor postings.

At the core of our system is the 'language' collection, serving as the foundation with attributes 'lid' and 'language'. Furthermore, there are two specialized collections: 'supplylanguage' and 'demandlanguage'. These collections establish a hierarchical relationship with the 'language' collection, representing the supply and demand aspects.

The 'supplylanguage' collection is associated with repositories, featuring an attribute, 'lid', referencing the 'language' collection, and an attribute, 'rid', used for linking to repositories. On the other hand, the 'demandlanguage' collection is linked to job postings, containing an attribute, 'lid', to connect to the 'language' collection, and an attribute, 'jid', to relate to jobs.

For the last two collections, there are the 'repository' and 'job' collections. The 'repository' collection encompasses attributes such as name, number of watchers, and number of issues, providing insights into the repositories hosting languages. Meanwhile, the 'job' collection, includes attributes for job title and description, offering visibility into the language skills sought in various job opportunities.

As opposed to the SQL implementation, there are no classes, tables, primary keys or reference keys. MongoDB uses collections to organize data, documents [and stores them in a format called BSON \(Binary JSON\)](#), which allows for more flexible and schema-less data structures.

Approaches in populating data

In order to work with NoSQL, we decided to work with the MongoDB platform. The approach to populating the data is then pretty simple. Since we had the CSVs from the last phase, all we had to do was import them to their respective collection, see Image 1. Of course, there is the task of downloading and setting up a project in MongoDB first. After initial setup, the process of querying the data and getting execution times is relatively simple in MongoDB. MongoDB Compass is a GUI that has a user-friendly interface. Therefore, after watching one or two YouTube videos about the application, running queries is very fast and simple.

Queries must be run in stages, so the different parts of a query like find, sort, and limit are all separated in stages. After all the stages are complete, you can save this pipeline in the aggregation section in MongoDB Compass and run it. After this, the output of the query will be given to you. Another great point about MongoDB Compass is that receiving specs about a query only takes clicking one button: “Explain”.

Another huge part of this project was adding indexes to maximize query optimization. Adding indexes is also another star point about Compass. Adding indexes is as easy as navigating toward the Index section on the GUI and selecting which attribute to index and how: in ascending and descending order. As well, you can add properties, like adding uniqueness to an index.

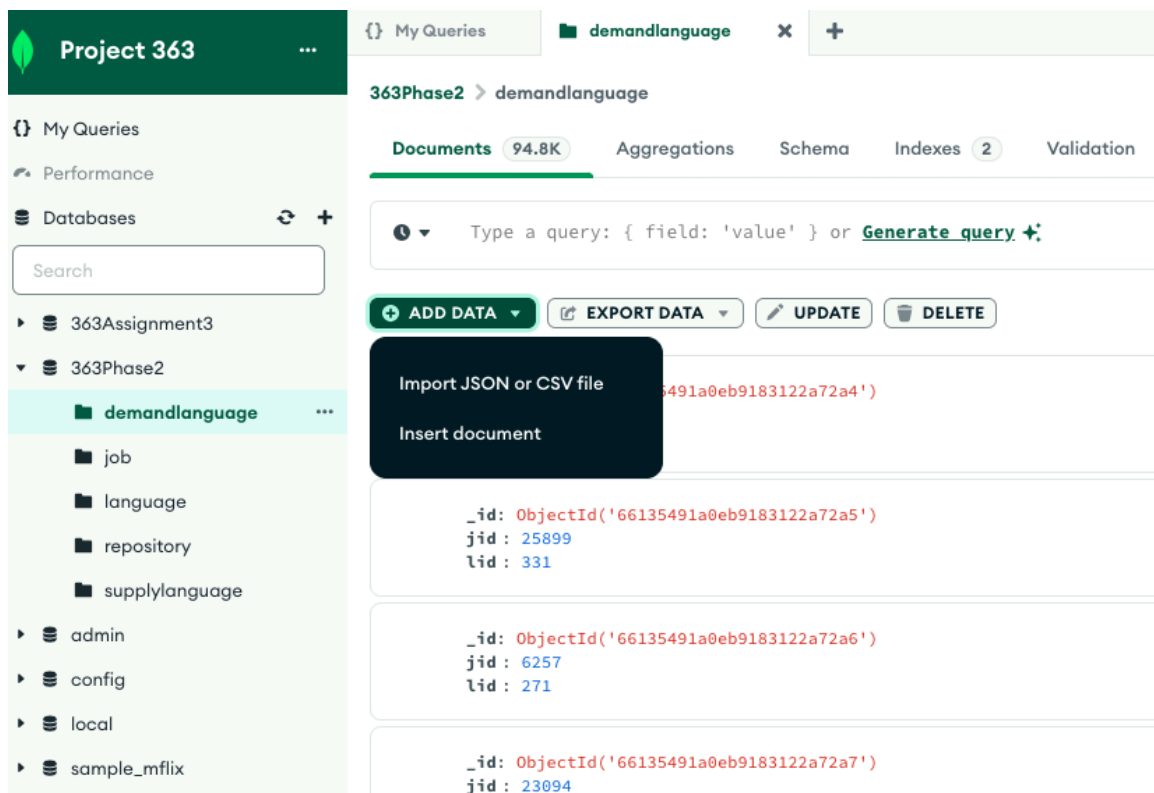


Image 1:Example of Importing a CSV to demandlanguage Collection

Data Model

Collection language
_id: objectid
lid: int32
language: string

Collection repository
<code>_id: objectid</code>
<code>rid: int32</code>
<code>name: string</code>
<code>number_of_watchers: int32</code>
<code>number_of_issues: int32</code>

Collection supply;language
_id: objectid
rid: int32
lid: int32

Collection demandlanguage
_id: objectid
jid: int32
lid: int32

Collection job
_id: objectId
jid: int32
job_title: string
job_description: string


Project collections overview:

The screenshot shows the MongoDB Compass interface. On the left, a sidebar displays the project hierarchy: Project 363 > My Queries > Performance > Databases > Search > 363Assignment3 > 363Phase2 > demandlanguage, job, language, repository, supplylanguage, admin, config, local, sample_mflix. The main area shows five database cards for 'demandlanguage', 'job', 'language', 'repository', and 'supplylanguage'. Each card displays storage size, document count, average document size, index count, and total index size.

Database	Storage size	Documents	Avg. document size	Indexes	Total index size
demandlanguage	2.27 MB	95 K	40.00 B	2	4.35 MB
job	72.91 MB	37 K	3.21 KB	2	121.34 MB
language	69.63 kB	784	51.00 B	2	122.88 kB
repository	15.41 MB	360 K	93.00 B	1	10.15 MB
supplylanguage	15.84 MB	752 K	40.00 B	1	22.41 MB

These documents are compressed. Therefore, the size in MB would be well over 200MB.

Here is the real size of the job collection shown below.

job

Uncompressed data size: 118.13 MB

Storage size: 72.91 MB

Documents: 37 K

Avg. document size: 3.21 kB

Indexes: 2

Total index size: 121.34 MB

Challenges in populating data

There are some challenges that we faced while working with MongoDB. Since we were working on MongoDB Compass, the members of the team who weren't owners of the project had to create users inside the MongoDB Atlas after being invited by the owner. In order to connect this user, we needed to get the connection URI through MongoDB Atlas with the new user credentials to access the cluster within Compass. On top of this, MongoDB Compass couldn't connect with the project unless the IP address of the computer of each member was specified within MongoDB Atlas. Only after these two steps could the member have access to the project. Therefore, there could easily be a delay before working on queries.

Furthermore, MongoDB Compass is a local version of the remote version Atlas. Meaning, that queries (aggregations) created on Compass weren't synced with the remote version. Only indexes or new collections created locally were synced to remote, not any new pipelines (queries). This is why we created a 'Query Implementation' document, added screenshots of the pipeline and exported the aggregation in JSON. If we wanted to run other teammates' queries, we could use the exported JSON of the aggregation and run it with the command line using Atlas to see the output or create a new pipeline on Atlas GUI with the same workflow shown in the screenshots.