Data Modeling and Migration

Created by
Jyoti Sharma

Nov 30, 2020

Advanced Database Management

Prof Jack Polnar

Table of Contents

GOAL

This project aims to explore the best methods and operational practices to import data from flat file or relation DB systems to NoSql MongoDB.

As a part of this project, I have worked on

- Exporting data from CSV to MongoDB/JSON format.
- Understanding best practices of data modeling/schema designing.
- Migrating data to MongoDB.
- Understand best operational practices in MongoDB.

How we are going to approach our goal

We will be using two datasets, data from different sources. The idea is first to understand the dataset and the differences in data modeling approaches in Relational databases and NoSql databases.

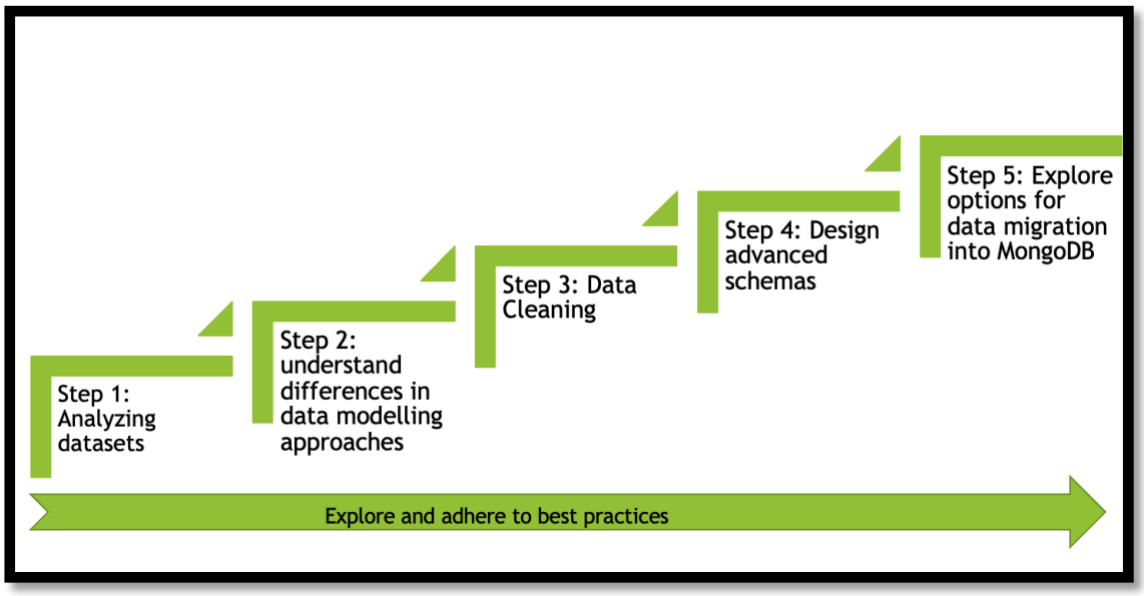


Fig: Path for achieving goals

DATASET INFORMATION

Dataset Description

I have used two different datasets downloaded from Kaggle.com. The datasets contain information about various job postings.

Data Source 1: moster_com-job_sample.csv
This data is an extract of job postings on monster.com- a job search website.

Data Source 2: data_scientist_united_states_job_postings_jobspikr.csv
This data contains information about multiple job postings for data scientist profiles across the United States and collections for different platforms like indeed, dice, monster, career builder, etc.

Dataset Attributes

Data Source 1: moster_com-job_sample.csv
The below table consists of a description of all columns:

Fields	**Description**	**Available in both datasets**
country	The country name for a job opening	N
country_code	Abbreviation of Country name for the job opening	Y
date_added	The date of job posting	Y
has_expired	Is the job still open	N
job_board	The platform posting the job opening	Y
job_description	Description of the job profile	Y
job_title	Job position title	Y
job_type	Type of Job	N
location	Location	N
organization	Job category, describing if it's in accounting or biotech or banking	Y
page_url	URL of job posting	Y
salary	Salary Range	N
sector	The experience type for the job.	N
uniq_id	Uniquely identifying the job posting	N

Data Source 2: data_scientist_united_states_job_postings_jobspikr.csv
The below table consists of a description of all columns:

Fields	**Description**	**Available in both datasets**
crawl_timestamp	The timestamp of extracting data from the job portal	N
URL	URL of job posting	Y
job_title	Job position title	Y
category	Job category, describing if it's in accounting or biotech or banking	Y
company_name	Name of the company	N
city	city area for the job position	N

state	State for the job position	N
country	Abbreviation of Country name for the job opening	Y
inferred_city	the city for the job position	N
inferred_state	State in which job is offered	N
inferred_country	The country in which job is open	Y
post_date	Date of job posting	Y
job_description	Description of the job profile	Y
job_type	Type of job contract Full/Part-time	N
salary_offered	Salary offered	Y
job_board	The site from which job post is fetched	Y
geo	Country name	N
cursor	Longitude and latitude of job location	N
contact_email	Contact person email	N
contact_phone_number	Contact person Phone	N
uniq_id	Unique id for the job posting	N
html_job_description	HTML code for Job description	N

Data Sources

https://www.kaggle.com/

Understanding the Datasets and differences

Both datasets contain information about different job postings. There are some differences in both datasets and the fields to be implemented in the final implementation. Also, few fields are missing, will be required to add more fields.

Fields to be Kept:

Fields	*Description*	*Available in both datasets*
job_title	Job position title	Y
category	Job category, describing if it's in accounting or biotech or banking	Y
company_name	Name of the company	N

inferred_city	the city for the job position	N
inferred_state	State for the job position	N
inferred_country	County for the job position	Y
job_description	Description of the job profile	Y
post_date	Date in which job is posted	N
job_type	Type of job contract Full/Part-time	N
contact_email	Contact person email	N
contact_phone_number	Contact person Phone	N
job_board	site name posting the job	Y
salary	Salary Range offered	Y
has_expired	Is the job position currently open?	N

Additional Fields:

Fields	*Description*
RequiredQual	What is the minimum required qualification?
top skills	what are the top skills needed by end-users

DATA MODELING PRACTICES

Differences in Relational and MongoDB (~NoSql)

MongoDB's design philosophy is to combine the critical capabilities of relational databases and innovative NoSql technologies. The vision is to leverage Oracle and others' work over the last 40 years to make relational databases what they are today. Rather than discard decades of proven database maturity, MongoDB picks up where they left off by combining key relational database capabilities with Internet pioneers' work to address modern applications' requirements [11].

Data Modeling Approach in MongoDB

One of the critical differences between relational and MongoDB is the idea based on which cluster/table structure is defined. Where relation DBMS focuses on how data is stored, NoSql focuses on data retrieval pattern by applications. Unlike in relational DBMS in MongoDB, the documents are designed keeping in mind how it is retrieved, paying a little or no attention towards normalization.

Advantages of NoSQL

- **Flexible Schema**
  Unlike relation SQL databases, where you must determine the and declare table schema before inserting data, MongoDB's collection by default doesn't require its document to have a schema. That is:
  - Documents of a single collection need not have the same set of fields. The data type for fields can differ across documents within the collection. [2]
  - To change the documents' structure in a collection like adding a new field, remove an existing field, or change field values to a new type, update the documents to a new structure. [2]

  This flexibility helps in mapping a document to an object or entity. Each document can match the represented entity's field, even if it has substantial document variation from other collection documents. [2]
  However, the latest version of MongoDB also gives the option of enforcing schema/document validation rules. The schema validation can be implemented using a JSON schema validator supported by MongoDB.

  For example, the following example specifies validator rules using the query expression:

```
db.createCollection( "contacts",
   { validator: { $or:
      [
         { phone: { $type: "string" } },
         { email: { $regex: /@mongodb\.com$/ } },
         { status: { $in: [ "Unknown", "Incomplete" ] } }
      ]
   }
} )
```

  Ref took from [3]
  It provides constraints on format of email field it should end with @mongodb.com, also the value of status field should be either 'Unknown' Or 'Incomplete'.

To control the scope of validation implementation, MongoDB also provides the following related options:

- **validationLevel option**: This determines how strictly MongoDB applies validation rules to existing documents during an update. There are two possible values strict or moderate. Strict is the default behavior.
    i. strict (the default) MongoDB applies validation rules to all inserts and updates.
    ii. moderate applies validation rules to inserts and to updates to existing documents that already fulfill the validation criteria
- **validationAction option**: which determines whether MongoDB should error and reject documents that violate the validation rules or warn about the log's violations but allow invalid documents. There are two possible values error (the default) or warn. If set to error it will reject any insert or updates that violates the validation criteria, whereas warn will log but allow insert of documents. [3]
    i. error (the default), MongoDB rejects any insert or update that violates the validation criteria.
    ii. warn, MongoDB logs any violations but allows the insertion or update to proceed.

- **Document structure**

The primary focus in data model designing for MongoDB application is focused on documents' structure and how the application represents relationships between data. MongoDB two ways to organize the data via **embedding** and **linking**.

- **Embedding:**

Embedding helps documents by capturing relationships between data in a single document store. This schema is also referred as a denormalized data model. It is used to depict

- One to one relationship
- One to many relationships

In general, embedding provided better performance for reading operations and the ability to request and retrieve related data in a single database operation. One restriction is the document size; there is a limit of 16MB on MongoDB documents.

- **Linking/References:**

Linking/References stores the relationship between data by including links and references to another document. Applications can use the references to access the related data. This is also referred to as the Normalized version.[2]

MongoDB applications use one of two methods for relating documents:

- Manual references
- DBRef's

Manual reference is the practice of including one document _id field into another. The application can take the _id and query the second document.

```
original_id = ObjectId()

db.places.insert({
    "_id": original_id,
    "name": "Broadway Center",
    "url": "bc.example.net"
})

db.people.insert({
    "name": "Erin",
    "places_id": original_id,
    "url":   "bc.example.net/Erin"
})
```

Example is taken from the MongoDB Manual

However, **DBRef's** maintains the reference by using three fields

- $ref
- $id
- $db

Where $db is usually optional.

- **Atomicity of write operations**
  - **Single Document Atomicity**

    In MongoDB, atomicity is at the document level, even if the operation modifies multiple documents embedded within the single document.[2]
  - **Multi-Document Transactions**

    From version 4.2 onwards, MongoDB introduces distributed transactions, which adds support for multi-document transactions on sharded clusters and incorporates the existing support for multi-document transactions on replica sets. The atomicity of multi-document transactions is maintained by below
    - all-or-nothing: In case of multiple document transactions, it will either commit/rollback all transactions or none.
    - In a situation where a transaction is aborted, all the data changes made by the transaction are discarded. [14]

- **Data Use and Performance**

  In MongoDB, one of the primary focuses is on how your application is going to behave. If the applications use only recently introduced document, then **Capped Collections** usage can be considered.[2]

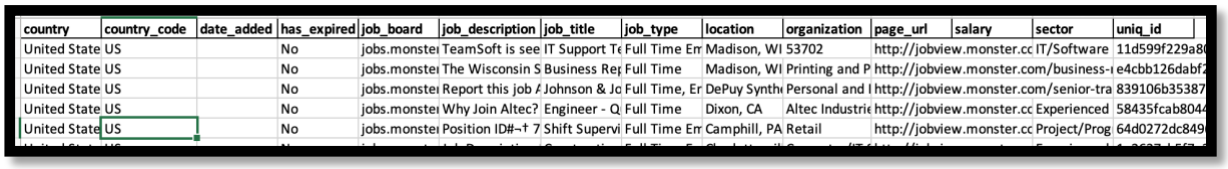
  Few operational considerations that might affect the data designs are

  - Sharding
  - Indexes
  - A large number of collections or Collection containing a large number of documents
  - Storage optimization for large documents
  - Data Life Cycle Management

Best design practices in MongoDB considers the above-mentioned facts before deciding on the data schema design.

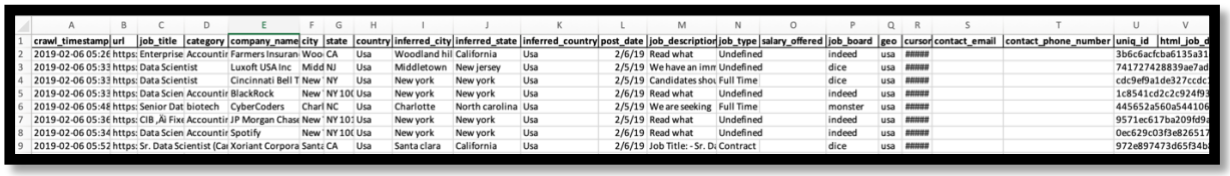
DATA PREPROCESSING

For any data migration activity preprocessing the data plays a critical role. Below are the snapshots of both datasets



country	country_code	date_added	has_expired	job_board	job_description	job_title	job_type	location	organization	page_url	salary	sector	uniq_id
United State	US		No	jobs.monster	TeamSoft is see	IT Support Te	Full Time Em	Madison, WI	53702	http://jobview.monster.cc		IT/Software	11d599f229a8
United State	US		No	jobs.monster	The Wisconsin S	Business Rep	Full Time	Madison, WI	Printing and P	http://jobview.monster.com/business-i			e4cbb126dabf
United State	US		No	jobs.monster	Report this job A	Johnson & Jo	Full Time, Er	DePuy Synthe	Personal and I	http://jobview.monster.com/senior-tra			839106b35387
United State	US		No	jobs.monster	Why Join Altec?	Engineer - Q	Full Time	Dixon, CA	Altec Industrie	http://jobview.monster.cc		Experienced	58435fcab8044
United State	US		No	jobs.monster	Position ID#¬† 7	Shift Supervi	Full Time Em	Camphill, PA	Retail	http://jobview.monster.cc		Project/Prog	64d0272dc849

Snapshot of the **monster_com-job_sample** dataset



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	crawl_timestamp	url	job_title	category	company_name	city	state	country	inferred_city	inferred_state	inferred_country	post_date	job_description	job_type	salary_offered	job_board	geo	cursor	contact_email	contact_phone_number	uniq_id	html_job_d
2	2019-02-06 05:2	https:	Enterprise	Accountir	Farmers Insuran	Woo	CA	Usa	Woodland hil	California	Usa	2/6/19	Read what	Undefined		indeed	usa	#####			3b6c6acfcba6135a31	
3	2019-02-06 05:33	https:	Data Scientist		Luxoft USA Inc	Midd	NJ	Usa	Middletown	New jersey	Usa	2/5/19	We have an imn	Undefined		dice	usa	#####			741727428839ae7ad	
4	2019-02-06 05:33	https:	Data Scientist		Cincinnati Bell T	New	NY	Usa	New york	New york	Usa	2/5/19	Candidates sho	Full Time		dice	usa	#####			cdc9ef9a1de327ccdc	
5	2019-02-06 05:33	https:	Data Scien	Accountir	BlackRock	New	NY 10(	Usa	New york	New york	Usa	2/6/19	Read what	Undefined		indeed	usa	#####			1c8541cd2c2c924f93	
6	2019-02-06 05:48	https:	Senior Dat	biotech	CyberCoders	Charl	NC	Usa	Charlotte	North carolina	Usa	2/5/19	We are seeking	Full Time		monster	usa	#####			445652a560a544106	
7	2019-02-06 05:36	https:	CIB ,Ãi Fixe	Accountir	JP Morgan Chase	New	NY 10:	Usa	New york	New york	Usa	2/5/19	Read what	Undefined		indeed	usa	#####			9571ec617ba209fd9a	
8	2019-02-06 05:34	https:	Data Scien	Accountir	Spotify	New	NY 10(	Usa	New york	New york	Usa	2/6/19	Read what	Undefined		indeed	usa	#####			0ec629c03f3e826517	
9	2019-02-06 05:52	https:	Sr. Data Scientist (Ca		Xoriant Corpora	Santa	CA	Usa	Santa clara	California	Usa	2/6/19	Job Title: - Sr. Da	Contract		dice	usa	#####			972e897473d65f34b	

Snapshot of **data_scientist_united_states_job_postings_jobspikr** dataset

As a part of preprocessing, I have performed the below steps

- **Deciding on the final fields**

  Both data files contain some additional fields that might not be required for the target application I have performed data modeling. So, we have removed a few columns and added some to bring both the datafiles in sync. The final data frame will consist of the below fields

Fields	**Description**
country_code	Abbreviation of Country name for the job opening
Date_added	Date in which job is posted
has_expired	Is the job still open

job_board	The site on which job is posted
job_description_Abstract	Trimmed description of the job profile
job_description	Description of the job profile
job_title	Job position title
job_type	Type of job contract Full/Part-time/contract
City	City in which job is offered
State	State in which job is offered
sector	Which field the job belongs to Medical or IT Or Marketing
JobPostingLink	Link of job posted
salary	Salary range offered
*TopSkills**	The top skill required for the job
*MinRequiredQual**	Minimum qualification required to apply for the job
CompanyName	Company name

***Newly added fields**

- **Cleaning of data**
  Since the dataset are obtained from the web scraping process, the data consist of many junk characters like "¬†>,Ä¢". So as a part of the cleaning process, I have applied the cleaning function to remove the junk characters from job_description, job_type, organization, country,inferred_city, inferred_state, category, company_name fields of original datasets before merging them into the clean dataset. I have written a python script 'CleaningScript.py' to perform cleaning and merging of the datasets. Below is the snippet of the string clean function.

```
#create a clean function
def function_clean(stringToBeCleaned):
new_string=""
new_string=re.sub("[¬†>,Ä¢]", " ",
stringToBeCleaned)
new_string=re.sub("/", ",", new_string)
new_string=new_string.strip()
```

Code snippet of cleaning function

- **Extracting data fields from a single string**
  In the monster job data set, the details of city and state are merged in one single column. As a part of the cleaning process and bringing both datasets in

sync, I have written a function for parsing the string and extract the data needed.

```
#Clean the job type field
def function_City(LocationString):
    str_ret=""
    str_ret= LocationString.split(',')
    #check if the string is in corect format
    if(len(str_ret)==2):
        return str_ret[0]
    else:
        return 'nan'
```

Function snippet for extracting city Name

```
def function_State(LocationString):
    str_ret=""
    str_ret_1=""
    str_ret= LocationString.split(',')
    #check if the string is in corect format
    if(len(str_ret)==2):
        #check the length of string
        if (len(str_ret[1])>=1):
            str_ret[1]=str_ret[1].replace(u'\xa0', u' ')
            str_ret[1]= str_ret[1].strip()
            #check if the regular expression is matched
            matched = re.match("[A-Z]{2}[0-9]{0,}" , str(str_ret[1]))
            boolVal=bool(matched)
            if(boolVal):
                str_ret_1 = str_ret[1].split(' ')
                return str_ret_1[0]
            else:
                return 'nan'
        else:
            return 'nan'
    else:
        return 'nan'
```

Function snippet for extracting state Name

- **Merging the datasets**

After cleaning, both datasets are merged into a single data frame, ‘CleanDataSet.csv’. The data modeling process will be carried out on this dataset.

	country_code	Date_added	has_expired	job_board	job_description_Abstract	job_description	job_title	job_type	City	State	JobArea	JobPostingLink	salary	sector	TopSkills	MinRequiredQual	CompanyName
0	US		No	monster	TeamSoft is seeing an IT Sup	TeamSoft is seeing	IT Support Te	Full Time Em	Madisc	WI	nan	http://jobview.monster.c		IT/Software Development			Farmers Insurance Gro
1	US		No	monster	The Wisconsin State Journa	The Wisconsin Sta	Business Rep	Full Time	Madisc	WI	Printing a	http://jobview.monster.com/business-reporter-editor-job-madison-					Luxoft USA Inc
2	US		No	monster	Report this job About the Jc	Report this job Ab	Johnson & Jc	Full Time, En	nan	nan	Personal a	http://jobview.monster.com/senior-training-leader-job-raynham-m					Cincinnati Bell Techno
3	US		No	monster	Why Join Altec? If you,Âóre	Why Join Altec? I	Engineer - Qu	Full Time	Dixon	CA	Altec Indu	http://jobview.monster.c		Experienced (Non-Manager)			BlackRock
4	US		No	monster	Position ID#-† 76162 # Pos	Position ID#-† 76	Shift Supervi	Full Time Em	Camph	PA	Retail	http://jobview.monster.c		Project/Program Management			CyberCoders
5	US		No	monster	Job Description Job #: -†72	Job Description J	Constructio	Full Time Em	Charlo	VA	Compute	http://jobview.monster.c		Experienced (Non-Manager)			JP Morgan Chase

Snapshot of Merged Dataset

Cleaning script execution

Below are the specifics of the script used for data cleaning purpose

Tool Used: Spyder

Programming Language: Python

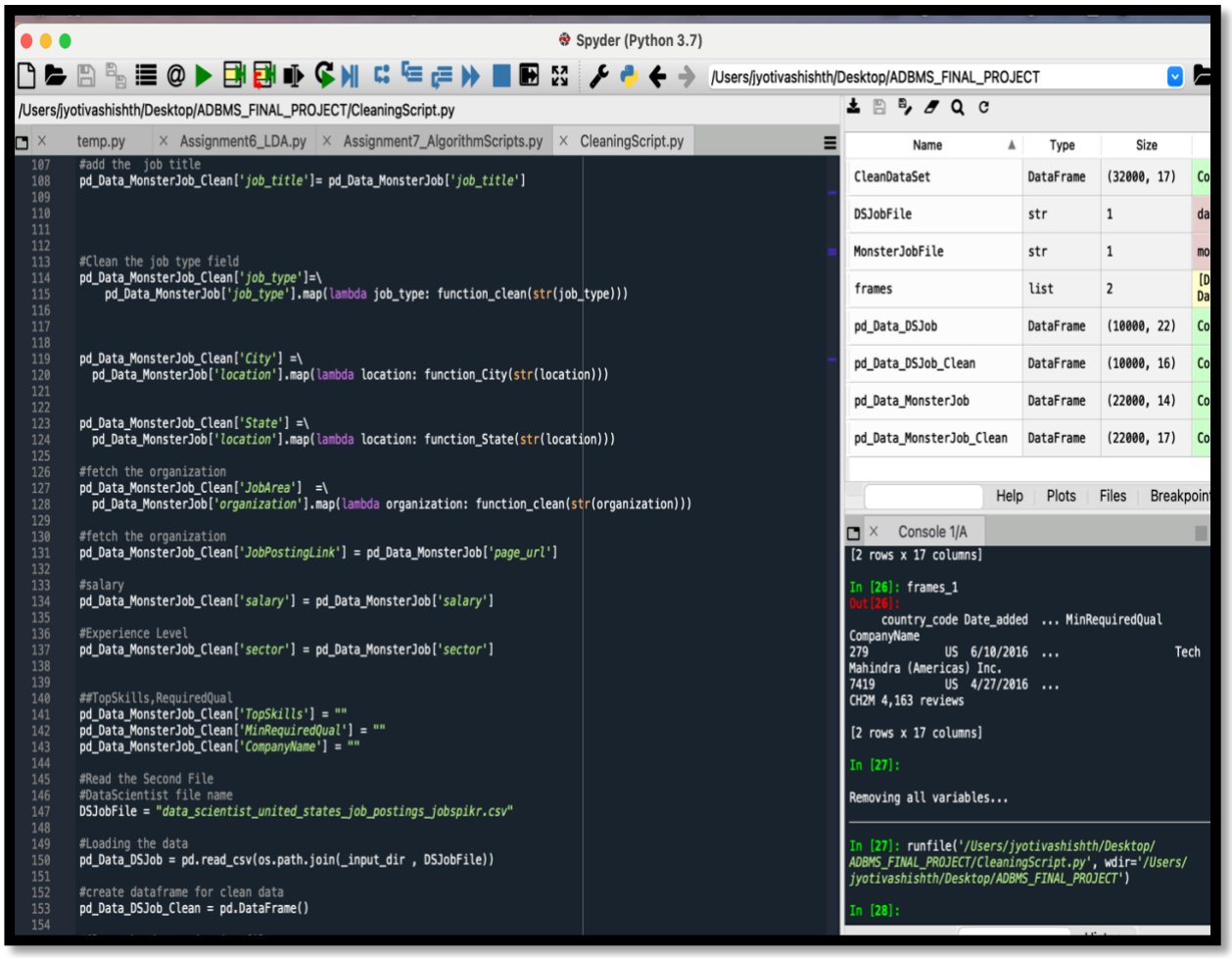


Fig: Screenshot of Python cleaning script execution

APPLICATION ANALYSIS

One of the primaries focuses on data modeling in MongoDB is to understand the application queries.

	Tabular	MongoDB
Steps to create the model	1 – define schema 2 – develop app and queries	1 – identifying the queries 2 – define schema
Initial schema	• 3rd normal form • one possible solution	• many possible solutions
Final schema	• likely denormalized	• few changes
Schema evolution	• difficult and not optimal • likely downtime	• easy • no downtime

Summary of differences between Tabular and MongoDB [17]

The MongoDB while implementing an ideal methodology, one should follow the below steps
Step 1: Describing the workload
Step 2: Identify the entities and relationship
Step 3: Design model and apply patterns

Step 1: Describing the workload

I have aimed to design the MongoDB schema for an entry-level job portal, which collects data from multiple job search platforms and shows the job openings inside the USA at one go. This is a one-stop solution for job hunting needs.
Name of System: One-Click Job Search
Benefits: The main goal of developing this system is to have a centralized job search platform. They are helping end-users to save time in the tedious job hunt process. The schema designing process will begin with describing the workload, look for the most frequent operations which will be performed by the application.

<u>List of operations:</u>
Below is the list of operations to be implemented in the application

Query	Operation	Description
1. Search the job postings based on job title or a substring of job title.	Read	Lookup for jobs with keywords or full job title.
2. Search the job posting based on the company name.	Read	Look for jobs based on the name of the company in which the position is open.

3. Search the job posting based on State name.	Read	Look for jobs based on the state name in which the job position is opened.
4. Read the full job description.	Read	Look up the full job description for the job post of interest.
5. Updating the jobs	Write	Update a job posting in the system, adding a new job posting, or marking a job posting a closed.
6. Search the job posting based on skills required	Read	Lookup for a job-based on the skills job requires.

Workload quantify/qualify operation:
Below is the summary of the quantification and qualification of the most frequent operations in the initial implementation.

Query	Quantification	Qualification
1. Search the job postings based on job title or a substring of job title.	Retrieval time < 1 s	No stale data <Critical Operation>
2. Search the job posting based on the company name.	Retrieval time < 1 s	No stale data <Critical Operation>
3. Search the job posting based on State name.	Retrieval time < 1 s	No stale data <Critical Operation>
4. Read the full job description.	Retrieval time < 3 s	No stale data
5. Updating the jobs	Write time < 0.05s per transaction	critical write
6. Search the job posting based on skills required	Retrieval time < 3 s	No stale data

Step 2: Identify the entities and relationship

In the process of identifying models and relationships, the first step is to identify the entities.

Entities:

After analyzing the data and the goals, I have come with the below entities

- **JobInfo**: This entity will consist of information that helps describe the job or give details about the job.
- **JobPostingInfo:** This entity will consist of information that helps describe which website the job posting came from and what date it will be available.
- **LocationInfo:** This entity will consist of information that helps describe the location of job openings like the city, state, and country.

- **JobCategory:** This entity will consist of information that helps describe the job posting's domain type.
- **JobRequirements:** This entity will consist of information that helps describe the job posting's skill and minimum education requirements.

All the fields of these entities are not normalized; for example, TopSkills field in entity **JobRequirements** can be further denormalized. TopSkills field will contain multiple skills required for the job like c programming, HTML, .Net, Java, python etc.

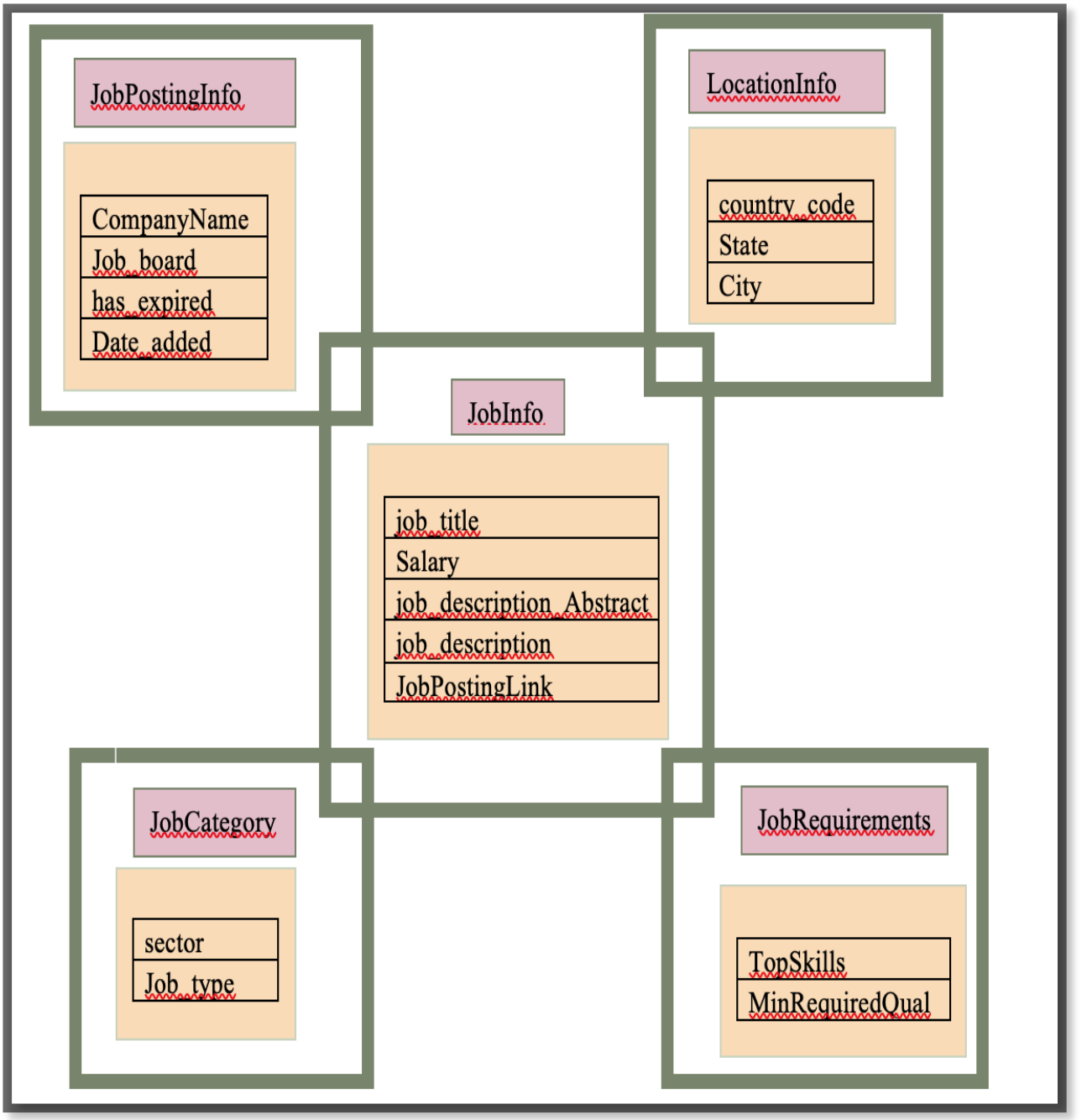


Fig: Multiple entities

The entities derived are not in normalized form, but that's the beauty of the NoSQL system. We can create documents as per our needs.

Data example

Below is data for one row for all entities

JobInfo

ROW ID	1
job_title	IT Support Technician Job in Madison
salary	IT/Software Development
job_description_Abstract	TeamSoft is seeing an IT Support Specialist to join our client in Madison, WI. The ideal candidate must have at least 6 years of experience in the fie...
job_description	b"TeamSoft is seeing an IT Support Specialist to join our client in Madison, WI. The ideal candidate must have at least 6 years of experience in the field. They need to be familiar with a variety of the field's concepts, practices, and procedures as this position relies on extensive experience and judgment to plan and accomplish goals. Required Skills: Call tracking software Phone based technical support Problem documentation and communication Remote Desktop Management Tools Respond to customer requests General understanding of LANDesk Microsoft Office 2007 SuiteFind out why TeamSoft is the Madison area's technology leader with over 200 IT consultants. Owned, operated, and managed by IT consultants, TeamSoft is the clear choice for your career. Join the TeamSoft family today!Did you know? TeamSoft offers salaried options for many positions that include 5 weeks PTO and great benefits! TeamSoft has been in business since 1996 TeamSoft's owners are local IT professionals who possess a wealth of experience in application development, business analysis, and project management TeamSoft's Recruiters are knowledgeable, friendly, and ready to help you make your next great career move TeamSoft offers a full benefits suite to all of our W2 consultants, including a 401k plan with a 100% vested 4% dollar for dollar match Our targeted selection process is designed to get to know your strengths and career interests and provide you with the best chance for success in your new position Our longest running consultant has been with us for 16 years Consultants can participate in our charitable giving committee. To date, TeamSoft's charitable giving committee has donated over $100,000!TeamSoft is owned by local IT professionals. Our team also has deep experience with IT staffing. Our clients understand this and rely on our expertise. That is why TeamSoft is the Madison Area's largest privately owned IT consulting

	firm, with more than 200 current consultants! With over 50 active clients, TeamSoft will give you the greatest exposure to the widest range of IT careers.Our owners possess a combined 60+ years of IT consulting experience. Our recruiting and Account Management staff are very well-versed in our client markets. We get you. We know you. We know what you do every day. And we know what you deal with on the job.You take your career seriously and so do we. At TeamSoft you are not just a resume or number. Our goal is to create long-term partnerships with each of our consultants, and according to feedback from many of them, we are very good at this. Click HERE to see what our consultants have to say about working with TeamSoft.TeamSoft -- Promote yourself!TeamSoft is an equal employment opportunity employer functioning under an Affirmative Action Plan."
JobPostingLink	http://jobview.monster.com/it-support-technician-job-madison-wi-us-167855963.aspx?mescoid=1500134001001&jobPosition=20

JobPostingInfo

ROW ID	**1**
CompanyName	
job_board	monster
has_expired	No
Date_added	

LocationInfo

ROW ID	**1**
country_code	US
City	Madison
State	WI

JobCategory

ROW ID	**1**
sector	IT/Software Development
job_type	b'Full Time Employee'

JobReqirements

ROW ID	**1**
TopSkills	
MinRequiredQual	

In this scenario

- **JobInfo** entity will have **one-to-one relationships** to **LocationInfo**, as each job posting will have one city, one state, and one job associated with it.
- **JobInfo** entity will have **one-to-one relationships with JobPostingInfo,** as each job posting will have one company name, one job board, date, and an expired flag associated with it.

In summary the way schema is designed every entity will have one-to-one relationships with the main entity JobInfo.

Step 3: Design Model and Apply patterns

For designing I Based on the frequent operations, entities and relationships embed all documents inside the JobInfo document can be a design choice to cater our needs.

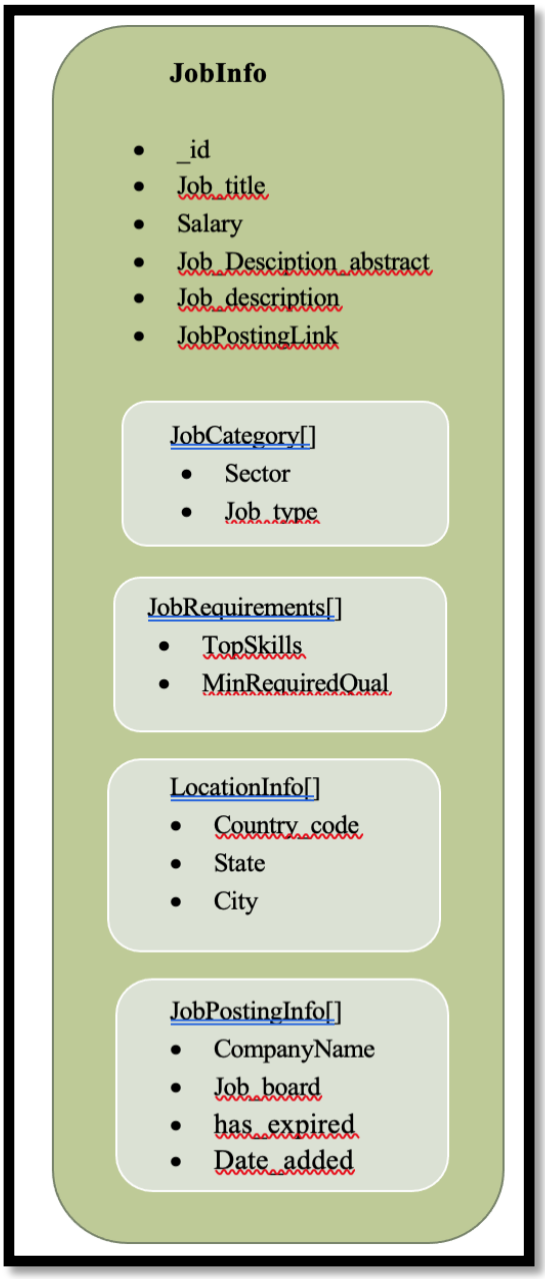


Fig: Displaying the embed all design – DesignVersion 1.0

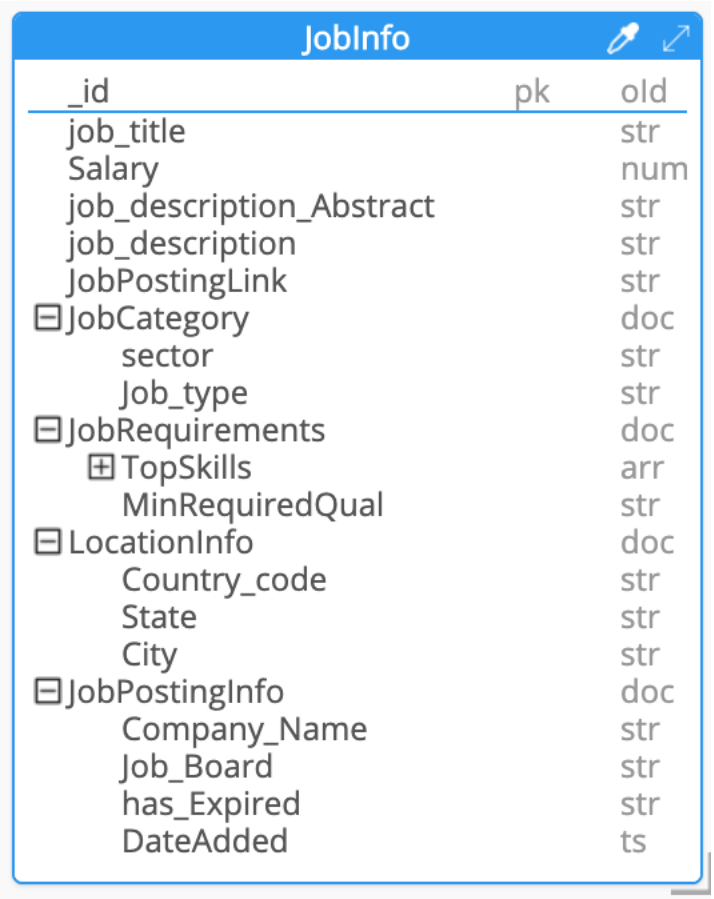


Fig: ER design for Version 1_0

MongoDB Script generated by Hackolade tool

JobInfo JSON Data syntax generated by Hackolade tool

Based on the current list of operations this design should serve the purpose, however there are few constraints and possible challenges that might occur. Below is the list of challenges that might occur

i. Job description field: This field contains vary long strings and one of the constraints in MongoDB is its document limit. So, it will make sense to move the job_description field into another cluster and link it to the existing one. In this design the job_description_abstract(~only contains 153 char) will give an overview of job.

Possible future requirements:

The model can be further enhanced by adding some schema design patterns in it.

i. Schema Versioning pattern:

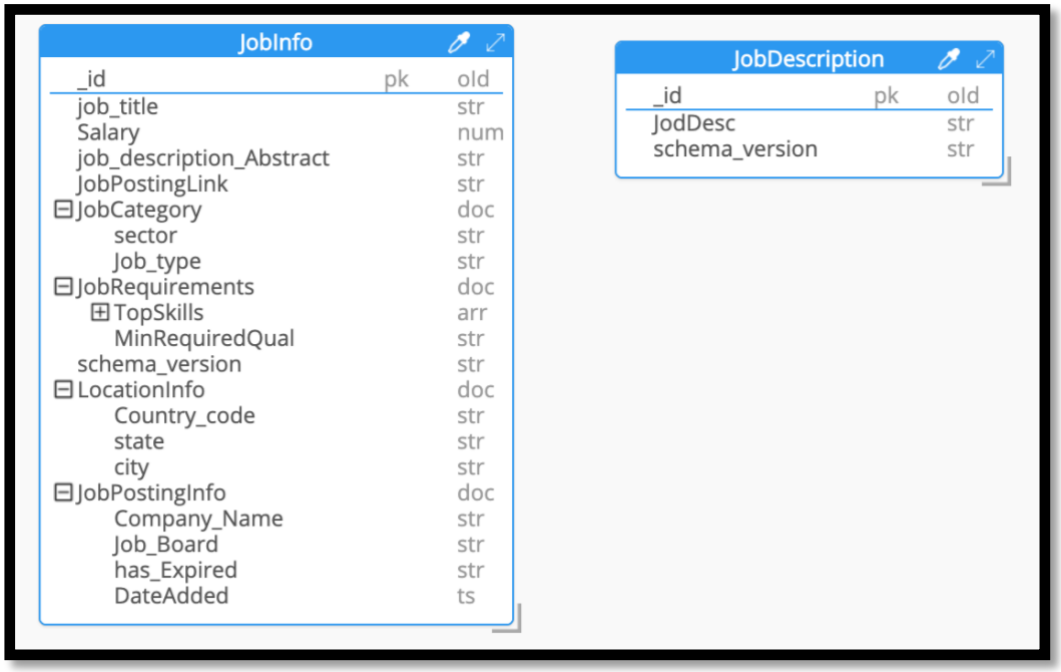
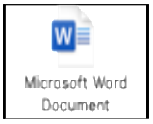


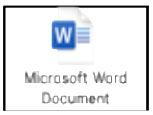
Fig: ER design for Version 1_1

JobInfo MongoDB Script generated by Hackolade tool

JobDescription MongoDB Script generated by Hackolade tool



JobInfo JSON Data syntax generated by Hackolade tool



JobDescription JSON Data syntax generated by Hackolade tool

DATA LOADING

Step 1: Generating documents using script

NoSQL schema requires a specific format for data. For creating the insert file, I have created a python script. The python script takes the clean data from "CleanDataSet.csv" file and generates two json files

i. JobDescription_new_insert.json file containing document for JobDescription collection.

ii. JobinfoList_new_insert.json file containing document for JobInfo collection.

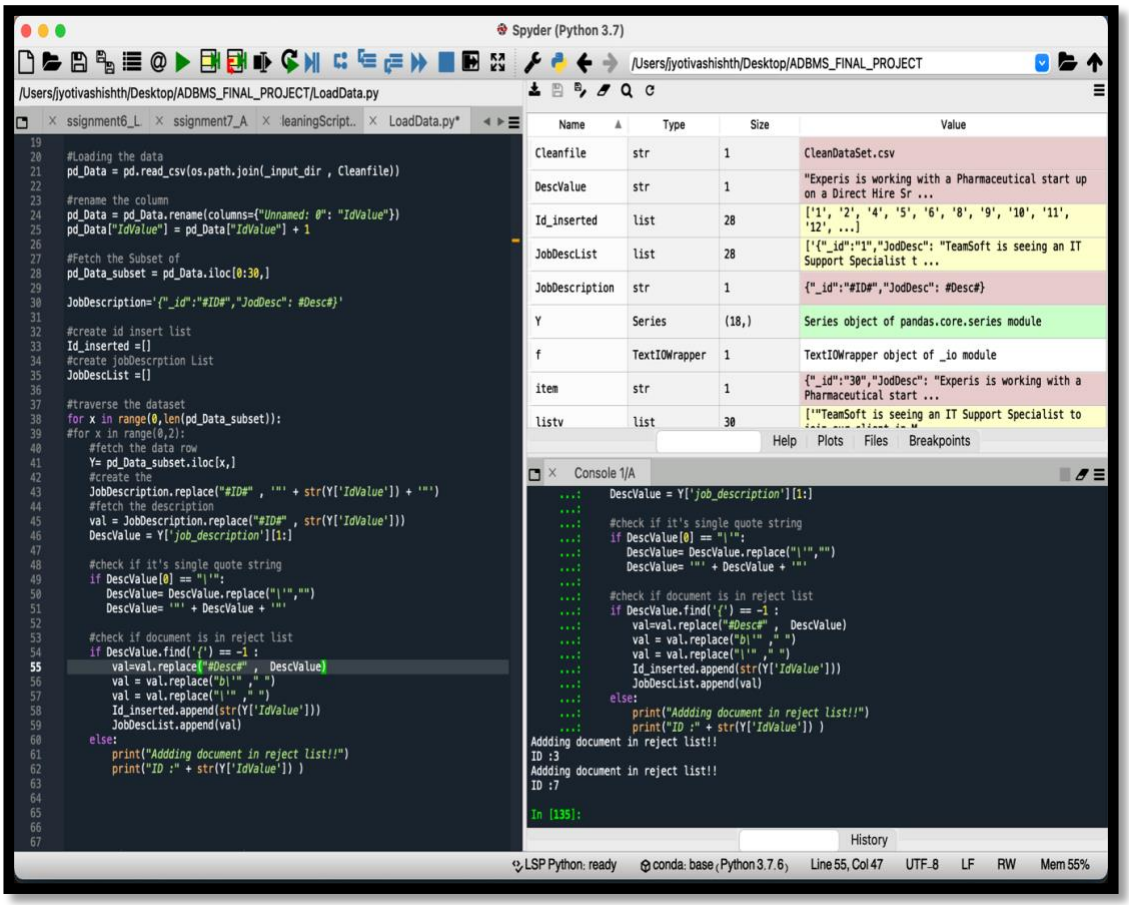


Fig: Execution screenshot of LoadData script

This script is traversing and generating documents and creating a reject list if in case any record doesn't pass the syntax checks and reject it. In the end it will also display the id of rejected records.

Step 2: Loading data into Mongo DB

The output files generated by ‘LoadData.py’ script ‘JobDescription_new_insert.json’ and ‘JobinfoList_new_insert.json’. These files are uploaded using MongoDB compass to insert documents inside collection.

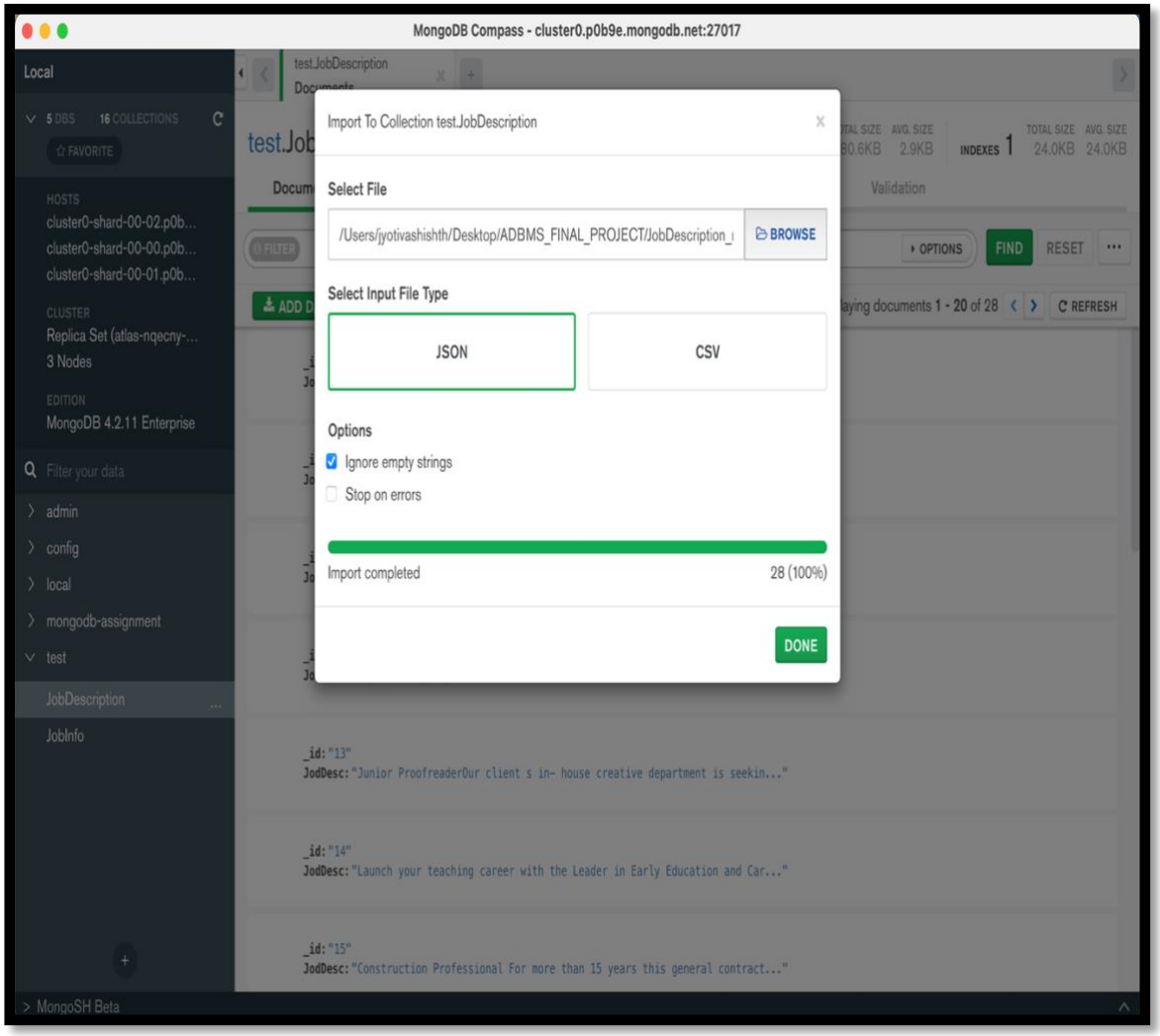


Fig: Upload JobDescription documents

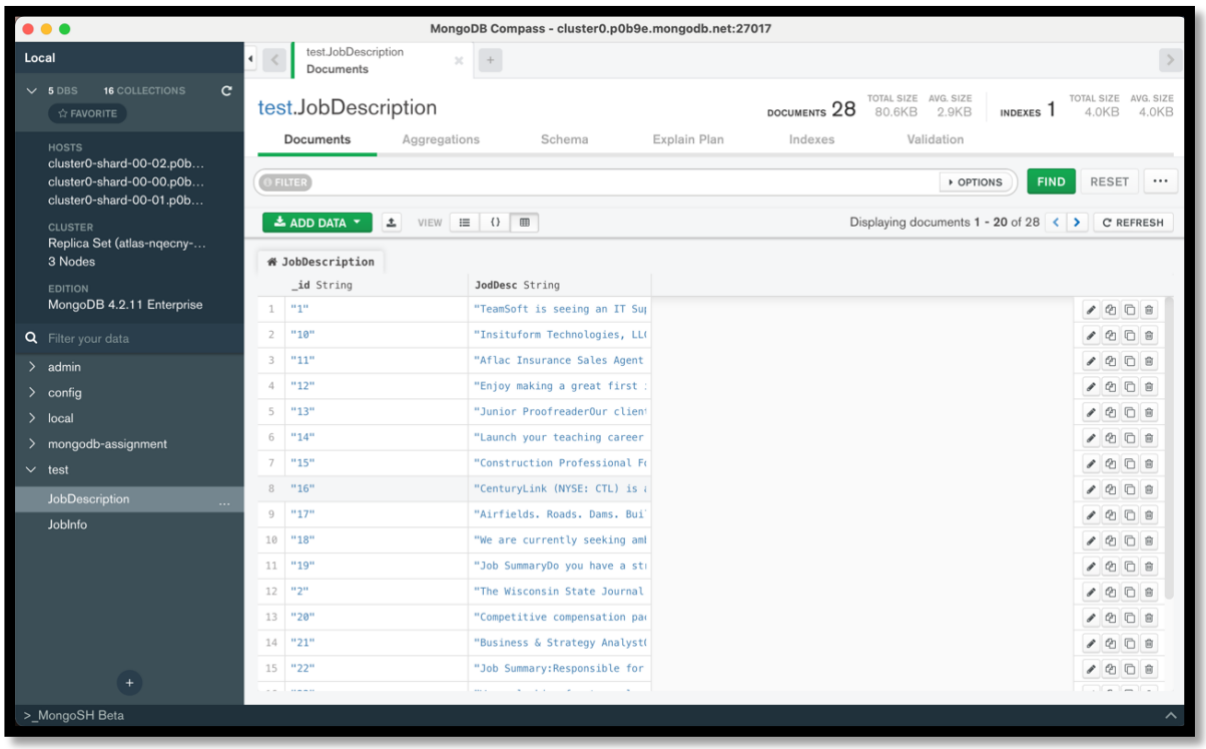


Fig: Validating uploaded JobDescription documents

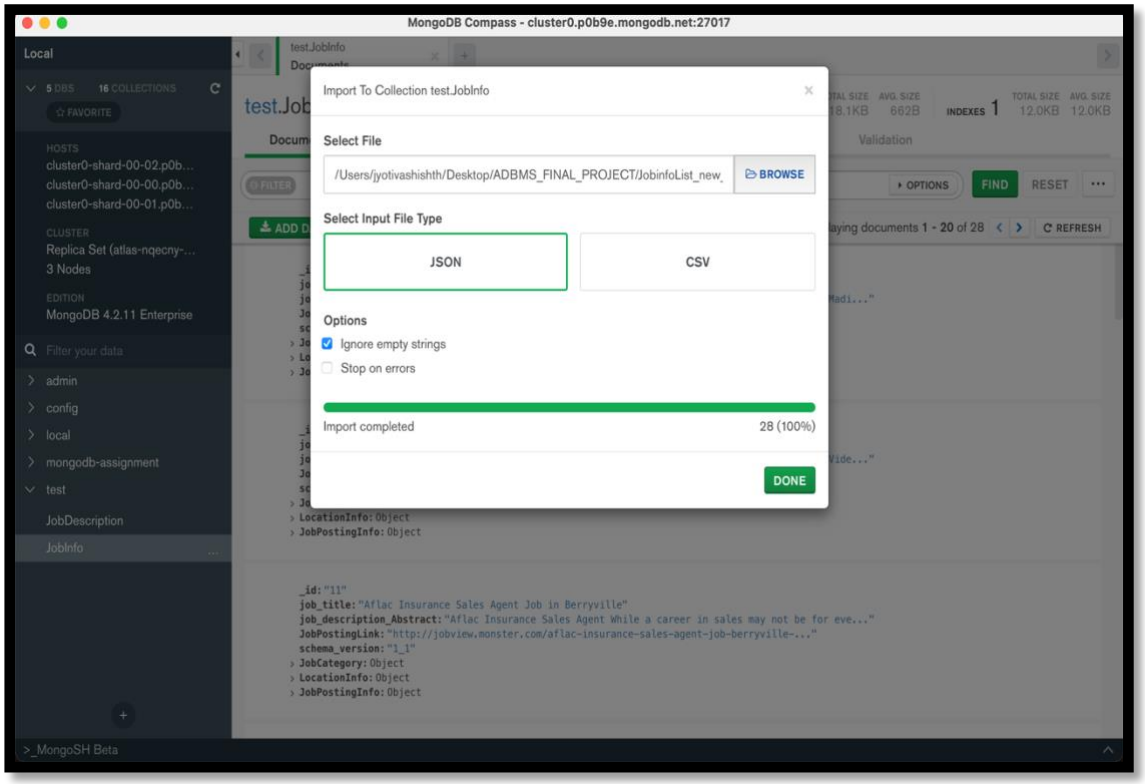


Fig: Upload JobInfo documents

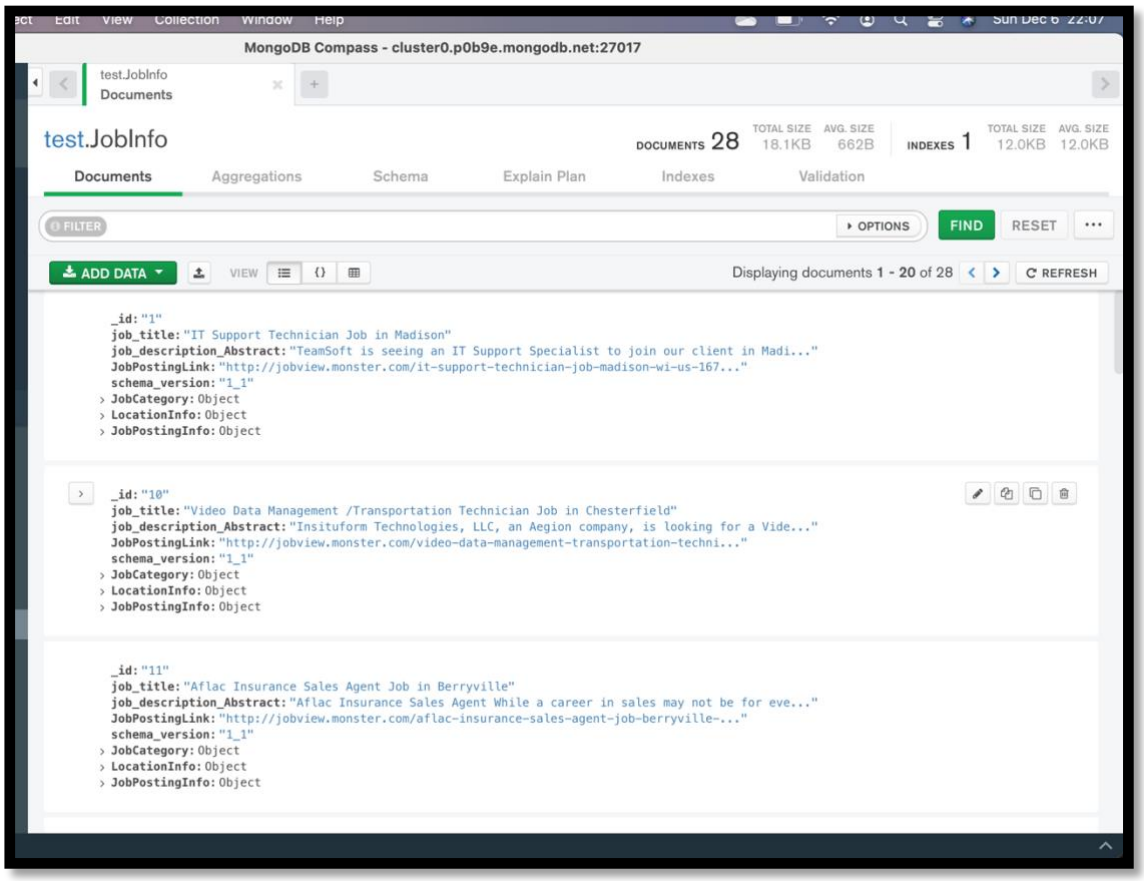


Fig: Validating uploaded JobInfo documents

QUERYING LOADED DATA

MongoDB compass also provides you with MongoDB shell. I have used MongoDB shell to query data from MongoDB collection created

Query 1:

Joining jobInfo and JobDescription based on _id fields in both collections

MongoDB Query:

```
db.getCollection('JobInfo').aggregate([{
$lookup:{
        from: "JobDescription",
        localField: "_id",
        foreignField : "_id",
        as: "studentUnits"
    }}])
```

Result Screenshot:

[{ _id: '1', job_title: 'IT Support Technician Job in Madison', job_description_Abstract: 'TeamSoft is seeing an IT Support Specialist to join our client in Madison, WI. The ideal candidate must have at least 6 years of experience in the fie...', JobPostingLink: 'http://jobview.monster.com/it-support-technician-job-madison-wi-us-167855963.aspx?mescoid=1500134001001&jobPosition=20', schema_version: '1_1', JobCategory:{ sector: 'IT/Software Development', Job_type: 'Full Time Employee' }, LocationInfo: { Country_code: 'US', state: 'WI', city: 'Madison' }, JobPostingInfo: { Job_Board: 'monster', has_Expired: 'No' }, studentUnits: [{ _id: '1', JodDesc: 'TeamSoft is seeing an IT Support Specialist to join our client in Madison, WI. The ideal candidate must have at least 6 years of experience in the field. They need to be familiar with a variety of the field s concepts, practices, and procedures as this position relies on extensive experience and judgment to plan and accomplish goals. Required Skills: Call tracking software Phone based technical support Problem documentation and communication Remote Desktop Management Tools Respond to customer requests General understanding of LANDesk Microsoft Office 2007 SuiteFind out why TeamSoft is the Madison area s technology leader with over 200 IT consultants. Owned, operated, and managed by IT consultants, TeamSoft is the clear choice for your career. Join the TeamSoft family today!Did you know? TeamSoft offers salaried options for many positions that include 5 weeks PTO and great benefits! TeamSoft has been in business since 1996 TeamSoft s owners are local IT professionals who possess a wealth of experience in application development, business analysis, and project management TeamSoft s Recruiters are knowledgeable, friendly, and ready to help you make your next great career move TeamSoft offers a full benefits suite to all of our W2 consultants, including a 401k plan with a 100% vested 4% dollar for dollar match Our targeted selection process is designed to get to know your strengths and career interests and provide you with the best chance for success in your new position Our longest running consultant has been with us for 16 years Consultants can participate in our charitable giving committee. To date, TeamSoft s charitable giving committee has donated over $100,000!TeamSoft is owned by local IT professionals. Our team also has deep experience with IT staffing. Our clients understand this and rely on our expertise. That is why TeamSoft is the Madison Area s largest privately owned IT consulting firm, with more than 200 current consultants! With over 50 active clients, TeamSoft will give you the greatest exposure to the widest range of IT careers.Our owners possess a combined 60+ years of IT consulting experience. Our recruiting and Account Management staff are very well-versed in our client markets. We get you. We know you. We know what you do every day. And we know what you deal with on the job.You take your career seriously and so do we. At TeamSoft you are not just a resume or number. Our goal is to create long-term partnerships with each of our consultants, and according to feedback from many of them, we are very good at this. Click HERE to see what our consultants have to say about working with TeamSoft.TeamSoft -- Promote yourself!TeamSoft is an equal employment opportunity employer functioning under an Affirmative Action Plan.', schema_version: '1_1' }] }

Query 2:

Fetching the unique Job_type and fetch the count of no of positions opened in each Job_type.

MongoDB Query: Fetching distinct Job_Type values loaded into documents

```
db.JobInfo.aggregate([{$unwind:"$JobCategory"},{$group:{_id:"$J
obCategory. Job_type"}}])
```

Result Screenshot:

```
> db.JobInfo.aggregate([{$unwind:"$JobCategory"},{$group:{_id:"$JobCategory.Job_type"}}])
< [ { _id: 'Full Time Employee' },
    { _id: 'Part Time Employee' },
    { _id: 'Full Time Temporary,Contract,Project' },
    { _id: 'nan' },
    { _id: 'Full Time, Employee' },
    { _id: 'Full Time' } ]
>
```

MongoDB Query: Fetching distinct Job_Type values along with their count for loaded into documents.

MongoDB Query:

```
db.JobInfo.aggregate([
      { "$unwind": "$JobCategory" },
      { "$group": { "_id": "$JobCategory.Job_type ",
       "count": { "$sum": 1 }
}}
])
```

Result Screenshot:

```
< [ { _id: 'Full Time Employee', count: 10 },
    { _id: 'Part Time Employee', count: 1 },
    { _id: 'Full Time Temporary,Contract,Project', count: 1 },
    { _id: 'nan', count: 4 },
    { _id: 'Full Time, Employee', count: 2 },
    { _id: 'Full Time', count: 10 } ]
>
```

Query 3:

Fetching the unique cities and fetch the count of no of positions opened in each city.

MongoDB Query: Fetching distinct city values loaded into documents

db.JobInfo.aggregate([{$unwind:"$LocationInfo"},
{$group:{_id:"$LocationInfo.city"}}])

Result Screenshot:

```
> db.JobInfo.aggregate([{$unwind:"$LocationInfo"},{$group:{_id:"$LocationInfo.city"}}])
< [ { _id: 'Houston' },
    { _id: 'Denver' },
    { _id: 'Camphill' },
    { _id: 'Berryville' },
    { _id: 'Natick' },
    { _id: 'Madison' },
    { _id: 'Carter Lake' },
    { _id: 'Columbus' },
    { _id: 'Boston' },
    { _id: 'Des Moines' },
    { _id: 'Dixon' },
    { _id: 'San Carlos' },
    { _id: 'Austin' },
    { _id: 'Sr. Process Engineer' },
    { _id: 'Chesterfield' },
    { _id: 'Charlottesville' },
    { _id: 'nan' } ]
>
```

MongoDB Query: Fetching distinct city values loaded into documents along with count

```
db.JobInfo.aggregate([
    { "$unwind": "$LocationInfo" },
    { "$group": {    "_id": "$LocationInfo.city",
    "count": { "$sum": 1 }    }} ])
```

Result Screenshot:

```
  ])
[ { _id: 'Denver', count: 3 },
  { _id: 'Camphill', count: 1 },
  { _id: 'Berryville', count: 1 },
  { _id: 'nan', count: 2 },
  { _id: 'Natick', count: 1 },
  { _id: 'Madison', count: 2 },
  { _id: 'Columbus', count: 1 },
  { _id: 'Boston', count: 1 },
  { _id: 'Carter Lake', count: 1 },
  { _id: 'Des Moines', count: 1 },
  { _id: 'Dixon', count: 1 },
  { _id: 'San Carlos', count: 1 },
  { _id: 'Austin', count: 2 },
  { _id: 'Sr. Process Engineer', count: 1 },
  { _id: 'Charlottesville', count: 1 },
  { _id: 'Chesterfield', count: 1 },
  { _id: 'Houston', count: 7 } ]
```

CONSTRAINTS

Below are few constraints on MongoDB implementation and should be considered while implementing schema in MongoDB

- By default, MongoDB will automatically abort any multi-document transaction that runs for more than 60 seconds.[16]
- There are no hard limits to the number of documents that can be read within a transaction. As a best practice, no more than 1,000 documents should be modified within a transaction.[16]
- In MongoDB 4.0, a transaction is represented in a single oplog entry, therefore must be within the 16MB document size limit. If this limit is exceeded, the transaction will be aborted and fully rolled back. The transaction should therefore be decomposed into a smaller set of operations that can be represented in 16MB or less.[16]

WRITE DURABILITY

MongoDB uses write concerns to control the level of write guarantees for data durability.
With stronger write concerns, write operations wait until MongoDB applies and acknowledges the operation.
With stronger write concerns, write operations wait until MongoDB applies and acknowledges the operation. This is MongoDB's default configuration. The behavior can be further tightened by also opting to wait for replication of the write to:
• A single secondary
• A majority of secondaries
• A specified number of secondaries
• All of the secondaries – even if they are deployed in different data centers (users should evaluate the impacts of network latency carefully in this scenario) .[16]

Bibliography

1.) https://www.mongodb.com/blog/post/transitioning-from-relational-databases-to-mongodb
2.) https://docs.mongodb.com/manual/core/data-modeling-introduction/
3.) https://docs.mongodb.com/manual/core/schema-validation/
4.) https://docs.mongodb.com/manual/reference/database-references/
5.) https://www.mongodb.com/presentations/data-modeling-with-mongodb
6.) https://www.mongodb.com/presentations/advanced-schema-design-patterns
7.) https://docs.mongodb.com/manual/reference/sql-aggregation-comparison/
8.) https://docs.mongodb.com/manual/reference/sql-comparison/
< performing common operations like update/inserts>
9.) https://docs.mongodb.com/manual/reference/operator/aggregation/
< Examples of mathematical operations >
10.) https://docs.mongodb.com/master/core/transactions/
< API for performing transactions>
11.) https://www.mongodb.com/scale/relational-vs-non-relational-database
12.) https://docs.mongodb.com/manual/core/data-model-design/#data-modeling-embedding
13.) https://docs.mongodb.com/manual/reference/database-references/#document-references
14.) https://docs.mongodb.com/manual/core/transactions/

White Paper:

15.) MongoDB_operations_best_practices-iksgd9n62y.pdf, published by MongoDB
16.) RDBMStoMongoDBMigration.pdf, published by MongoDB

MongoDB Tutorials:

17.) A Complete Methodology of Data Modeling for MongoDB
https://www.youtube.com/watch?v=DUCvYbcgGsQ
18.) Data Modeling with MongoDB
https://www.youtube.com/watch?v=yuPjoC3jmPA
19.) Advanced Schema Design Patterns
https://www.youtube.com/watch?v=bxw1AkH2aM4