An Analysis of the performance of MongoDB-Cluster using Health data

Project submitted

In partial fulfillment of the requirements for the degree of

MASTER OF COMPUTER APPLICATION

By

SUSMITA SINGHA

Roll No: **MCA226035**

Registration No: **149896 of 2019-2020**

Under the supervision of

Prof. (Dr.) Nandini Mukhopadhyay

Department of Computer Science & Engineering
Faculty of Engineering and Technology
Jadavpur University
Kolkata – 7000 032
India

Jadavpur University

Faculty of Engineering and Technology
Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the project entitled "**An Analysis of the performance of MongoDB-Cluster using Health data**" has been completed by Susmita Singha. This work is carried out under the supervision of Dr. Nandini Mukhopadhyay in partial fulfilment for the award of the degree of Master of Computer Application of the department of Computer Science and Engineering, Jadavpur University, during the session 2021-2022. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

_____

Dr. Nandini Mukhopadhyay
Professor
Project supervisor
Computer Science & Engineering
Jadavpur University

Countersigned:

_____

Prof. (Dr.) Anupam Sinha
Head of the department
Computer Science & Engineering
Jadavpur University

_____

Prof. Chandan Majumder
Dean
Faculty of Engineering & Technology
Jadavpur University

Jadavpur University

Faculty of Engineering and Technology
Department of Computer Science & Engineering

CERTIFICATE OF APPROVAL

The forgoing project is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as prerequisite to the degree for which it has been submitted. It is understood by this approval that the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion in that but approve this project only for the purpose for which it is submitted.

EXAMINER:

_____ _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION OF ORIGINALITYAND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this project contains original work by the undersigned candidate, as part of his Master of Computer Application (MCA) studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material results that are not original to this work.

Candidate's Name: Susmita Singha

Exam Roll No: **MCA226035**

Project Title: **An Analysis of the performance of MongoDB-Cluster using Health data**

Signature with Date:

ABSTRACT

Data has become one of the most valuable assets in today's digital world. The nature of the generated data varies from structured to semi-structured and even completely unstructured data items. Like many other verticals, the educational domain is also generating a huge volume of data with high variation. The main objective is to store and retrieve data in optimal time, efficiently utilizing recourses, and maintaining security. The evolution of different "Not only SQL (NoSQL)" data models to manage this huge volume and variety of data has helped to facilitate many powerful applications like Facebook, Instagram, WhatsApp, etc. NoSQL data models are provided by many vendors as installation-based services and cloud-based services as well. MongoDB is one of the popular document-oriented NoSQL data models. In this project, our objective is to analyze the performance MongoDB on increasing the number of node as well as the size of data . We have deployed the methodology for the stated semi-structured big-data handling with a case study. The performance of the execution of Mongo Query Language (MQL) on different situation (variable data size and number of node).

Table of Contents

CHAPTER # 1

Introduction

Contents:

- Introduction
- Problem Definition
- Aim
- Need of System
- Outline Of The Report

Introduction:

The nature of the data changes its characteristics from structured data to semi-structured and unstructured type with continuous digital exploration in different sectors including industry, research, education, business, healthcare, social network, etc. This variety of huge volumes of data comes from different sectors generated with high speed, and is referred to typically as 'Big Data'. Generally, the composition of Big Data is accumulated with 5Vs' - Volume, Velocity, Variety, Value, and Variability . The issue that comes with this type of data is proper storage, retrieval, analysis, and processing of data. The conventional RDBMS platform is excellent for handling structured schema-based data, but not for Big Data in that extent . Hence, one of the primary focuses of research for the last decade had been to find an alternative of RDBMS that can efficiently process Big data.

The new architecture is proposed as 'NoSQL' instead of SQL to deal with this huge volume and first generated data. NoSQL, the 'Not only SQL' is a schema less data model that supports the data transactions at the terabyte level. While RDBMS supports only vertical scale up, the NoSQL supports both horizontal and vertical scaling. The NoSQL configures the distributed node balanced cluster and has own its query language. The relational data model SQL supports ACID (Atomicity, Consistency, Isolation, Durability) properties. The NoSQL, on the other hand, supports the BASE (Basically Available, Soft state, Eventually Consistent) properties. In general, the NoSQL data models are categorized into four categories (shown in Fig 1.1).

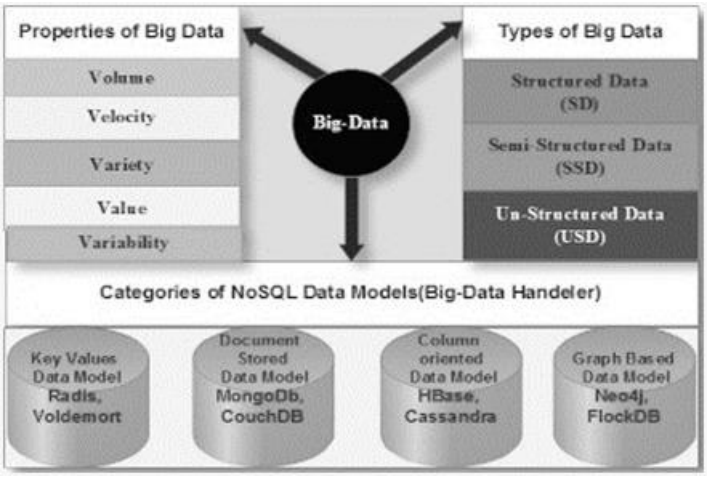


Fig-1.1

Problem Definition:

The innovation of the NoSQL data model is to handle the Big Data. The study on NoSQL shows that the MongoDB data model is one of the popular NoSQL data models. MongoDB can be used confidently in the real time system, especially in case of the online system. Now the question arises regarding the performances and the scalability. In a real life situation, every application asks for its performance factor. The evaluation of the performance of MongoDB cluster in different aspects shall help to decide the future application development using MongoDB designations for distributed system.

Aim:

The objective of this project is to analyze MongoDB query execution time on various sizes of health data by increasing the number of nodes. The project goes through mainly three-step repeatedly only changing the data size and increasing the number of nodes.

1. First we set up MongoDB cluster
2. Then run the python code to generate different size of synthetic health data
3. After that we run MongoDB query and save the result

Need of the System:

MongoDB is designed to **make data easy to access**, and rarely to require joins or transactions, but when we need to do complex querying, it's more than up to the task. The MongoDB Query API allows us to query deep into documents, and even perform complex analytics pipelines with just a few lines of declarative code.

Outline Of The Report:

Chapter 1(Introduction):

This chapter includes introduction of the system, problem definition, the main objective of this project and why do we need to do this Experiment.

Chapter 2(Hardware and Software requirements):

Chapter 2 includes which software and hardware are required to do this experiments and implementation issues.

Chapter 3(Environment Setup):

In this chapter we discuss about the installation of MongoDB , Open SSH connection, how we setup Cluster, Installation of MongoDB Compus and Installation of python.

Chapter 4(Synthetic Health Data Generation):

Chapter 4 includes Code for data generation, Structure of the generated data and Structure of documents.

Chapter 5(Results of Experiment):

Chapter 4 includes all the results of our experiment.

Chapter 6:

In this chapter we have discussion and conclusion.

CHAPTER # 2

Hardware and Software Requirements

Contents:

- Software requirement
- Hardware requirements
- Implementation issues

Software Requirements:

- Technology: MongoDB
- IDE : Pycharm/MongoDB compus
- Server Side Technologies: Python
- Data Base Server: MongoDB
- Operating System:-Linux

Hardware Requirements:

- Processor: Pentium-III (or) Higher
- Ram: 64MB (or) Higher
- Hard disk: 80GB (or) Higher
- Switch based wired network connection

Implementation issues

MongoDB:

MongoDB is a highly flexible and scalable NoSQL database management platform that is document-based, can accommodate different data models, and stores data in key-value sets. It was developed as a solution for working with large volumes of distributed data that cannot be processed effectively in relational models, which typically accommodate rows and tables. Like Hadoop, MongoDB is free and open-source.

Some Key Features of MongoDB Include:

1. It's a query language that is rich and supports text search, aggregation features, and CRUD operations.
2. It requires lesser input and output operations due to embedded data models, unlike relational databases. MongoDB indexes also support faster queries.
3. It provides fault tolerance by creating replica datasets. Replication ensures data is stored on multiple servers, creating redundancy, and ensuring high availability.
4. It features sharding, which makes horizontal scalability possible. This supports increasing data needs at a cost that is lower than vertical methods of handling system growth.
5. It employs multiple storage engines, thereby ensuring the right engine is used for the right workload, which in turn enhances performance.

Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets us work quickly and integrate systems more efficiently.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

CHAPTER # 3

ENVIRONMENT SETUP

Contents:

- Installation of MongoDB
- Open SSH connection
- Cluster setup
- Installation of MongoDB compus
- Python installation

Installation of MongoDB:

MongoDB installation commands on ubuntu 20.04 LTS

1	wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add –
2	echo "deb [arch=amd64,arm64,s390x] http://repo.mongodb.com/apt/ubuntu focal/mongodb-enterprise/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-enterprise.list
3	sudo apt-get update
4	sudo apt-get install -y mongodb-enterprise

Open SSH connection:

1.Open the terminal with Ctrl+Alt+T and install the openssh-server package:

```
$ sudo apt update
$ sudo apt install openssh-server
```

When prompted, enter password and press Enter to continue with the installation.

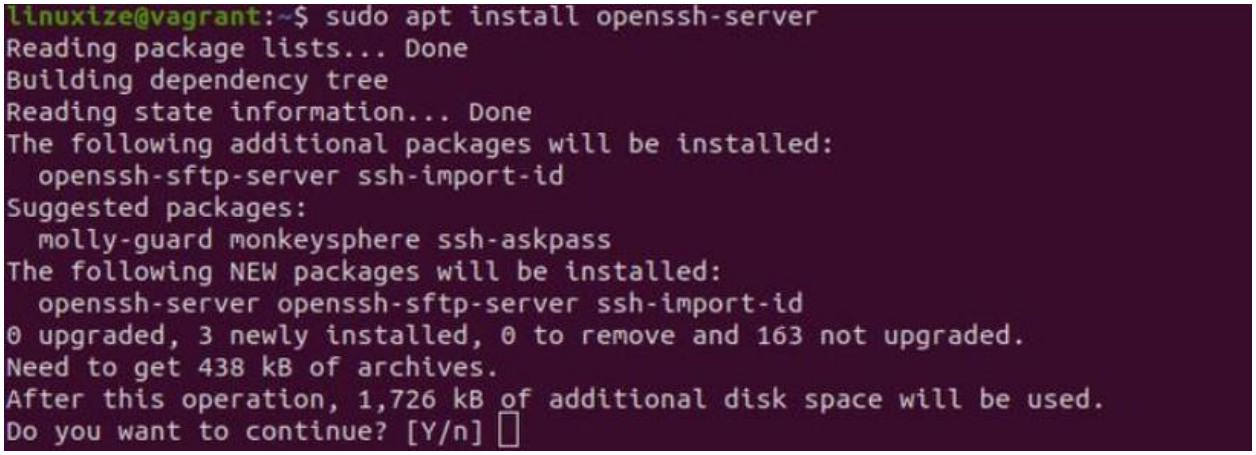


Fig-4.1

2. Once the installation is complete, the SSH service will start automatically. We can verify that SSH is running by typing:

```
sudo systemctl status ssh
```

The output should be the service is running and enabled to start on system boot:

```
● ssh.service - OpenBSD Secure Shell server

   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)

   Active: active (running) since Mon 2020-06-01 12:34:00 CEST; 9h ago

...
```

3.Ubuntu ships with a firewall configuration tool called UFW. If the firewall is enabled on our system, make sure to open the SSH port:

```
sudo ufw allow ssh
```

Connecting to the SSH server:

```
$ ssh username@ip_address
```

Cluster Setup:

This tutorial provides a brief overview of how replication works in MongoDB and outlines how to configure and initiate a replica set with three members. In this example configuration, each member of the replica set will be a distinct MongoDB instance running on separate Ubuntu 20.04 servers.

Understanding MongoDB Replica Sets

As mentioned in the introduction, MongoDB handles replication through an implementation called *replica sets*. Each running instance of MongoDB that's part of a given replica set is referred to as one of its *members*. Every replica set must have one *primary* member and at least one *secondary* member.

The primary member is the main access point for transactions with the replica set and is the only member that can accept write operations. Each replica set can have only one primary member at a time, as replication happens by copying the primary's *oplog* (short for "operations log") and repeating the logged changes on the secondaries' respective data sets. Multiple primaries accepting write operations would lead to data conflicts.

By default, applications will only query the primary member for both read and write operations. We can configure our setup to read from one or more of the secondary members, but since data is transferred asynchronously, reads from secondary nodes can result in old data being served. Thus, such a configuration isn't ideal for every use case.

Step 1 — Configuring DNS Resolution

When it comes time to initialize our replica set in Step 4, we will need to provide an address where each replica set member can be reached by the other two in the set. The MongoDB documentation recommends against using IP addresses when configuring a replica set, since IP addresses can change unexpectedly. Instead, MongoDB recommends using logical DNS hostnames when configuring replica sets.

One way to do this is to configure subdomains for each replication member. Although configuring subdomains would be ideal for a production environment or another long-term

solution, this tutorial will outline how to configure DNS resolution by editing each server's respective hosts files.

hosts is a special file that allows us to assign human-readable hostnames to numerical IP addresses. This means that if the IP address of any of our servers ever changes, we will only have to update the hosts file on the three servers instead of reconfiguring the replica set.

On Linux and other Unix-like systems, hosts is stored in the /etc/ directory. On **each of our three servers**, edit the file with our preferred text editor. Here, we'll use nano:

```
$ sudo nano /etc/hosts
```

After the first few lines which configure the **localhost**, add an entry for each member of the replica set. These entries take the form of an IP address followed by the human-readable name of our choice, as in this example:

```
IP_address    any_hostname
```

We can configure our servers to use whatever hostname we would like, but it can be helpful to make each hostname descriptive. In examples throughout this guide, the three servers will use these hostnames:

- **mongo0.replset.member**
- **mongo1.replset.member**
- **mongo2.replset.member**

Using these hostnames, our /etc/hosts files would look similar to the following highlighted lines:

```
. . .
127.0.0.1 localhost

203.0.113.0 mongo0.replset.member
203.0.113.1 mongo1.replset.member
203.0.113.2 mongo2.replset.member
. . .
```

Step 2 — Updating Each Server's Firewall Configurations with UFW

Assuming we followed the prerequisite initial server setup guide we will have set up a firewall on each of the servers on which we have installed MongoDB and enabled access for the OpenSSH UFW profile. This is an important security measure, as these firewalls currently block connections to any port on our servers, save for ssh connections that present keys which align with those in each server's respective authorized_keys file.

However, these firewalls will also block the MongoDB instances on each server from communicating with one another, preventing us from initiating the replica set. To correct this, we will need to add new firewall rules to allow each server access to the port on the other two servers on which MongoDB is listening for connections.

On **mongo0**, run the following ufw command to allow **mongo1** access to port 27017 on **mongo0**:

```
sammy@mongo0:~$ sudo ufw allow from mongo1_server_ip to any port 27017
```

Be sure to change mogno1_server_ip to reflect our **mongo1** server's actual IP address. Note that ufw commands will not work with hostnames configured in the hosts file, so be sure to use our servers' actual IP addresses in this command and the following ones. Also, if we have updated the Mongo instance on this server to use a non-default port, be sure to change 27017 to reflect the port that our MongoDB instance is actually using.

Then add another firewall rule to give **mongo2** access to the same port:

```
sammy@mongo0:~$ sudo ufw allow from mongo2_server_ip to any port 27017
```

Next, update the firewall rules for our other two servers. Run the following commands on **mongo1**, making sure to change the IP addresses to reflect those of **mongo0** and **mongo2**, respectively:

```
sammy@mongo1:~$ sudo ufw allow from mongo0_server_ip to any port 27017
sammy@mongo1:~$ sudo ufw allow from mongo2_server_ip to any port 27017
```

After adding these UFW rules, each of our three MongoDB servers will be allowed to access the port used by MongoDB on the other two servers. However, we won't be able to test this yet, since the Mongo instance on each server is currently blocking any external connections. After we enable replication by updating each MongoDB instance's configuration file in the next step, we will be able to perform this test.

Step 3 — Enabling Replication in Each Server's MongoDB Configuration File

At this point, we have edited our servers' /etc/hosts files to configure hostnames which will resolve to each one's IP address. We have also opened up each of our servers' firewalls to allow the other two servers access to the default MongoDB port, 27107. Now we are ready to begin configuring the MongoDB installation on each server to enable replication.

This step outlines how to do this by editing MongoDB's configuration file, /etc/mongod.conf. We must complete every procedure in this step on each server, but for demonstration purposes we will use **mongo0** in examples.

On **mongo0**, open the MongoDB configuration file in our preferred text editor:

```
sammy@mongo0:~$ sudo nano /etc/mongod.conf
```

Even though we have opened up each server's firewall to allow the other servers access to port 27017, MongoDB is currently bound to 127.0.0.1, the local loopback network interface. This means that MongoDB is only able to accept connections that originate on the server where it's installed.

To allow remote connections, we must bind MongoDB to our servers' publicly-routable IP addresses in addition to 127.0.0.1. This way, our MongoDB installation will be able to listen to connections made to our MongoDB server from remote machines.

Find the network interfaces section. It will look like this by default:

```
. . .
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1
. . .
```

Append a comma to the line beginning with bindIp: followed by **mongo0**'s hostname or public IP address. This example uses the hostname configured in Step 1:

```
. . .
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1,mongo0.replset.member
. . .
```

Next, find the line that reads #replication: towards the bottom of the file. It will look like this:

```
. . .
#replication:
. . .
```

Uncomment this line by removing the pound sign (#). Then add a replSetName directive below this line followed by a name which MongoDB will use to identify the replica set:

```
. . .
replication:
  replSetName: "rs0"
. . .
```

In this example, the replSetName directive's value is "rs0".

Note that there are two spaces before the replSetName directive and that the name is wrapped in quotation marks ("), both of which are necessary for this configuration to be read properly.

After updating these two sections of the file, net and replication, they will look like this:

```
. . .
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1,mongo0.replset.member
. . .
replication:
  replSetName: "rs0"
. . .
```

Save and close the file. Then make these same changes to the /etc/mongod.conf files on **mongo1** and **mongo2**. After doing so, these updated sections will look like this in **mongo1**'s configuration file:

```
. . .
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1,mongo1.replset.member
. . .
replication:
  replSetName: "rs0"
. . .
```

And here's how these sections will look in **mongo2**'s configuration file

```
. . .
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1,mongo2.replset.member
. . .
replication:
  replSetName: "rs0"
. . .
```

To reiterate, the IP address or hostname we add to each server's bindIp directive must be that of the server whose mongod.conf file we are editing.

After making these changes to each server's mongod.conf file, save and close each file. Then, restart the mongod service **on each server** by issuing the following command:

```
$ sudo systemctl restart mongod
```

Step 4 — Starting the Replica Set and Adding Members

Now that we have configured each of our three MongoDB installations, we can open up a MongoDB shell to initiate replication and add each as a member.

For demonstration purposes, the examples in this step will use the MongoDB instance on **mongo0** to initiate the replica set. However, we can initiate replication from any server whose mongod.conf file has been appropriately configured.

On **mongo0**, open up the MongoDB shell:

```
sammy@mongo0:~$ mongo
```

From the prompt, we can initiate a replica set from the mongo shell by running the rs.initiate() method. However, running this method by itself would only initiate replication for the machine on which we run the method, and we would then need to add our other Mongo instances by issuing an rs.add() method for each member.

Recall that MongoDB stores its data in JSON-like structures known as *documents*. Because we have already edited the mongod.conf file on each of our servers to configure the three Mongo instances for replication, we can instead include a document that holds each member's configuration details within the rs.initiate method. This will allow us to start the replica set and add each member at once, rather than having to run multiple separate methods.

To do this, begin an rs.initiate() method by typing the following and pressing ENTER:

```
> rs.initiate(
```

Assuming that we entered all the details correctly, once we press ENTER after typing the closing parenthesis the method will run and initiate the replica set. If the method returns "ok" : 1 in the output, it means that the replica set was started correctly:

```
Output
{
    "ok" : 1,
    "$clusterTime" : {
        "clusterTime" : Timestamp(1612389071, 1),
        "signature" : {
            "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
            "keyId" : NumberLong(0)
        }
    },
    "operationTime" : Timestamp(1612389071, 1)
}
```

Installation of MongoDB compus:

1.Download MongoDB compass for linux:

go to the official website of MongoDB. Go to the downloads page and select the version, platform, and type of file to be downloaded as shown in the image. This will directly download the file into the downloads folder of our system.

Fig-4.2

2. open ubuntu command terminal:

From the Application launcher run Terminal**.** On the Terminal use the below command to switch to the Downloads directory because whatever we get from the browser goes, by default, into it.

```
cd Downloads
```

3.Install MongoDB compass on ubuntu:

Now, let's use the APT package manager to install Compass on Ubuntu Linux

```
sudo apt install ./mongodb-compass_*_amd64.deb
```

4.Run compass GUI:

If we want to start Compass from the command terminal then simply type – **mongodb-compass** and hit the Enter key.

5.Connect MongoDB database:

Now, if we have the Database instance installed of MongoDB on our local system where we have installed the Compass GUI then simply click on the **Connect** button.

Python installation:

1. Update the packages list and install the prerequisites:

```
$ sudo apt update
$ sudo apt install software-properties-common
```

2. Add the deadsnakes PPA to our system's sources list:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
```

3. Once the repository is enabled, we can install Python 3.9 by executing:

```
$ sudo apt install python3.9
```

4. Verify that the installation was successful by typing:

```
$ python3.9 --version
```

CHAPTER # 4

Synthetic Health Data Generation

Contents:

- Code for data generation
- Structure of the generated data(collections)
- Structure of Documents

Code for data generation:

```
from random import randint
from random import randrange
from datetime import timedelta
from datetime import datetime
def get_database():
    from pymongo import MongoClient
    import pymongo
    client = MongoClient('localhost', 27017)

    # Create the database for our example (we will use the same database throughout the tutorial
    #return client['user_shopping_list']

    # Create the database for our example (we will use the same database throughout the tutorial
    return client['temp_dbs']
startingLimit = 1
executionEndLimit = 1000
oneItterationLimit = 100
personStart = 1
doctorStart = 1
patientStart = 1
```

```
historyStart =1
aHistoryStart =1
complaintStart =1
treatmentEpiStart = 1
visitStart =1
medicationStart= 1
therapyStart =1
surgeryStart = 1
diagnosisStart = 1
diseaseStart = 1
investigationStart = 1
continuousMoniteringStart = 1
sensorDataStart = 1



def getRandomFamilyRelation():
    relations =
    ["Father","Mother","Parent","Son","Daughter","Child","Husband","Wife","Spouse","Brother","Sister","Sib
    ling","Grandfather","Grandmother","Grandparents","Grandson","Granddaughter","grandchild","Uncle","
    Aunt","Parent's sibling","nephew","Niece","Sibling's Child","Cousin","aunt's Child","Uncle's Child"]
    randomIndex = randint(0,25)
    return relations[randomIndex]

def getRandomOccupation():
    occupation = ["Accountant", "Actors", "Actuary", "Ambassadors", "Artist", "Athlete", "Auto mechanic",
    "Bricklayer", "Broker", "Businessman", "Charterted Accountant", "Chemical technologist", "Coach",
    "Comedian", "Composers", "Consultant", "Curator", "Dental hygienist", "Dental technician", "Dietitian",
    "Doctor", "Drafter", "Drafter", "Electrician", "Engineers", "Film directors", "Hairdresser", "Health
    Assistance", "Joiner", "Judges", "Lawyer", "Machinist", "Mechanic", "Mechanical Engineer", "Medical
    laboratory scientist", "Midwife", "Military Personnel", "Network administrator", "Nurse", "Nurse educator",
    "Nurse practitioner", "Paramedic", "Pharmacy technician", "Physician", "Physiotherapist", "Pilot",
    "Plasterer", "Plumber", "Politician", "Professors", "Prosthetist", "Quantity surveyor", "Radiographer",
    "Refrigeration Mechanic", "Reporter", "researcher", "Scientist", "Scientists", "Service", "Social Workers",
    "Software Developer", "Sonographer", "Special Education Teacher", "Sportspeople", "Student", "Surgeon",
    "Surveyor", "Tailor", "Teacher", "Technical writer", "Technician", "Technologist", "Test engineer", "Tool
    and die maker", "Tradesman", "Writers", "Yoga Gurus", "Youth Worker"]
    randomIndex = randint(0,1)
    return occupation[randomIndex]

def getRandomReligion():
    religion = ["Christianity",
        "Islam",
        "Hinduism",
```

```
        "Buddhism",
        "Folk religion",
        "Taoism",
        "Shinto",
        "Falun Gong",
        "Sikhism",
        "Judaism",
        "Korean shamanism",
        "Caodaism",
        "Bahá'í Faith",
        "Tenriism",
        "Jainism",
        "Cheondoism",
        "Hoahaoism"]
    randomIndex = randint(0,16)
    return religion[randomIndex]



def getRandomFoodHabit():
    fh = ["Veg","Non-Veg"]
    randomIndex = randint(0,1)
    return fh[randomIndex]

def getRandomBloodgroup():
    bg = ["A+","B+", "AB+", "O+","A-","B-", "AB-", "O-"]
    randomIndex = randint(0,7)
    return bg[randomIndex]


def getRandomDate():
    start = datetime.strptime('1/1/1945 1:30 PM', '%d/%m/%Y %I:%M %p')
    end = datetime.strptime('28/3/2022 4:50 AM', '%d/%m/%Y %I:%M %p')
    delta = end - start
    int_delta = (delta.days * 24 * 60 * 60) + delta.seconds
    random_second = randrange(int_delta)
    return start + timedelta(seconds=random_second)




def getRandomGender():
    randomIndex = randint(0,2)
    gender = ["Male","Female","Others"]
```

```
    return gender[randomIndex]



def getRandomPhone():
    lenghth = 10

    text = randint(9000000000,9999999999)

    return text

def getRandomPincode():
    text = randint(7000000000,7999999999)
    return text


def randomStr(length):
    characters = 'abcdefghijklmnopqrstuvwxyz'
    charactersLength = len(characters)
    randomString = ''
    for i in range(0,length):
        randomString=randomString+characters[randint(0, charactersLength - 1)]

    return randomString


def getRandomEmail():
    length = randint(1,10)
    domains = ["gmail.com","hotmail.com","rediffmail.com","yahoo.com","outlook.com"]
    text = ""
    characters = 'abcdefghijklmnopqrstuvwxyz0123456789'

    charactersLength = len(characters)
    randomString = ''
    for i in range(0,length):
        randomString=randomString+characters[randint(0, charactersLength - 1)]
        randomIndex=randint(0,4)

    return randomString +"@"+ domains[randomIndex]
def getPostalAddressType():
    po = ["Permanent","Present"]
    randomIndex = randint(0,1)
    return po[randomIndex]
```

```
def getRandomSpecialization():

    doctorSpecilization = ["Allergist",
        "Immunologist",
        "Anesthesiologist",
        "Audiologist",
        "Cardiologist",
        "Dermatologist",
        "Dentist",
        "ENT doctor",
        "Endocrinologist",
        "Epidemiologist",
        "Gastroenterologist",
        "General Physician",
        "Gynecologist",
        "General Surgeon",
        "Hematologist/Oncologist",
        "Internal Medicine Physician",
        "Infectious Disease Specialist",
        "Medical Geneticist",
        "Microbiologist",
        "Nephrologist",
        "Neurologist",
        "Neurosurgeon",
        "Neonatologist",
        "Nurse-Midwifery",
        "Occupational Medicine Physician",
        "Ophthalmologist",
        "Oral and Maxillofacial Surgeon",
        "Orthopaedic Surgeon",
        "Otolaryngologist",
        "Oncologist",
        "Obstetrician",
        "Pathologist",
        "Pediatrician",
        "Plastic Surgeon",
        "Podiatrist",
        "Psychiatrist",
        "Pulmonary Medicine Physician",
        "Radiation Onconlogist",
        "Radiologist",
        "Rheumatologist",
        "Urologist",
```

```
        "Veterinary"]

    randomIndex = randint(0,41)
    return doctorSpecilization[randomIndex]




# This is added so that many files can reuse the function get_database()
if __name__ == "__main__":

    # Get the database
    dbname = get_database()
    collection_name = dbname["Persion"]
    doctors= dbname["Doctors"]
    patients=dbname["Patient"]
    history=dbname["History"]
    a_history=dbname["allergy-history"]
    complaint=dbname["complaint"]
    treatment_episode=dbname["treatment_episode"]
    treatmentPlan=dbname["treatmentPlan"]
    visit=dbname["visit"]
    medication=dbname["medication"]
    therapy=dbname["therapy"]
    surgerys=dbname["surgerys"]
    diagnosis=dbname["diagnosis"]
    disease=dbname["disease"]
    investigation=dbname["investigation"]
    continous_monitor=dbname["continous_monitor"]
    sensor_data=dbname["sensor_data"]
   # personTableAttri=[]
    for pr in range(0,executionEndLimit-1,oneItterationLimit):
        for i in range(1,oneItterationLimit):

            personTableAttri={"Id": personStart,"name": randomStr(randint(1 , 8))+" "+randomStr(randint(1 ,
    5)),"occupation": getRandomOccupation(),"socioEconomicStatus": randomStr(randint(1 , 200)),"religion":
    getRandomReligion(),"addressType": getPostalAddressType(),"addressLine1": randomStr(randint(1 ,
    10)),"addressLine2": randomStr(randint(1 , 10)),"cityTownVillagePoliceStation": randomStr(randint(1 ,
    9)),"district": randomStr(randint(1 , 15)),"state": randomStr(randint(1 , 10)),"pinCode":
    getRandomPincode() ,"countryCode": "IN","contactNo": getRandomPhone(),"email":
    getRandomEmail(),"gender": getRandomGender(),"dateOfBirthage": getRandomDate(),"bloodGroup":
    getRandomBloodgroup(),"foodHabbit":getRandomFoodHabit()}
```

```
        collection_name.insert_one(personTableAttri)
        #personStart=personStart+1
        #print(personTableAttri)

        if(personTableAttri["occupation"] == "Doctor"):
            doctorTableRow = {"drId": doctorStart,"personId": personStart,"organizationId":
randomStr(randint(1 , 50)) ,"specialization": getRandomSpecialization()}
            doctors.insert_one(doctorTableRow)
            doctorStart=doctorStart+1




        patientTableRow ={"ptId": patientStart,"Id": personStart,"alternatePtIds": personStart,"ecpId":
randint(1,executionEndLimit),"ecpRelation": getRandomFamilyRelation(),"cPAHomeId":
randint(1,executionEndLimit),"cPAHomeRelation": getRandomFamilyRelation() }
        patients.insert_one(patientTableRow)
        historyCount =randint(1,15)
        for i in range(1,historyCount):
            historyTableRow = {"historyId": historyStart ,"ptId": personStart,"familyHistoryDetails":
randomStr(randint(1,50)),"pastHistoryDetails": randomStr(randint(1,80)),"presentHistoryDetails":
randomStr(randint(1,30)),"personalHistoryDetails": randomStr(randint(1,10))}
            history.insert_one(historyTableRow)
            historyStart=historyStart+1

        aHistoryCount = randint(1,15)
        for i in range(1,aHistoryCount):
            aHistoryTableRow = {"allergyHId": aHistoryStart,"ptId": personStart,"allergyFrom":
randomStr(randint(1,50)),"allergySevrity": randomStr(randint(1,80)),"allergyStatus":
randomStr(randint(1,10))}
            a_history.insert_one(aHistoryTableRow)
            aHistoryStart=aHistoryStart+1


        complaintCount = randint(1,5)
        for x in range(1,complaintCount):
            complaintTableRow = {"complaintId":complaintStart, "ptId": personStart, "ComplaintDetails":
randomStr(randint(1,200)), "ComplaintStatus" : "active"}
            complaint.insert_one(complaintTableRow)

    #//=====complaint [one patient have multiple complaint]//
```

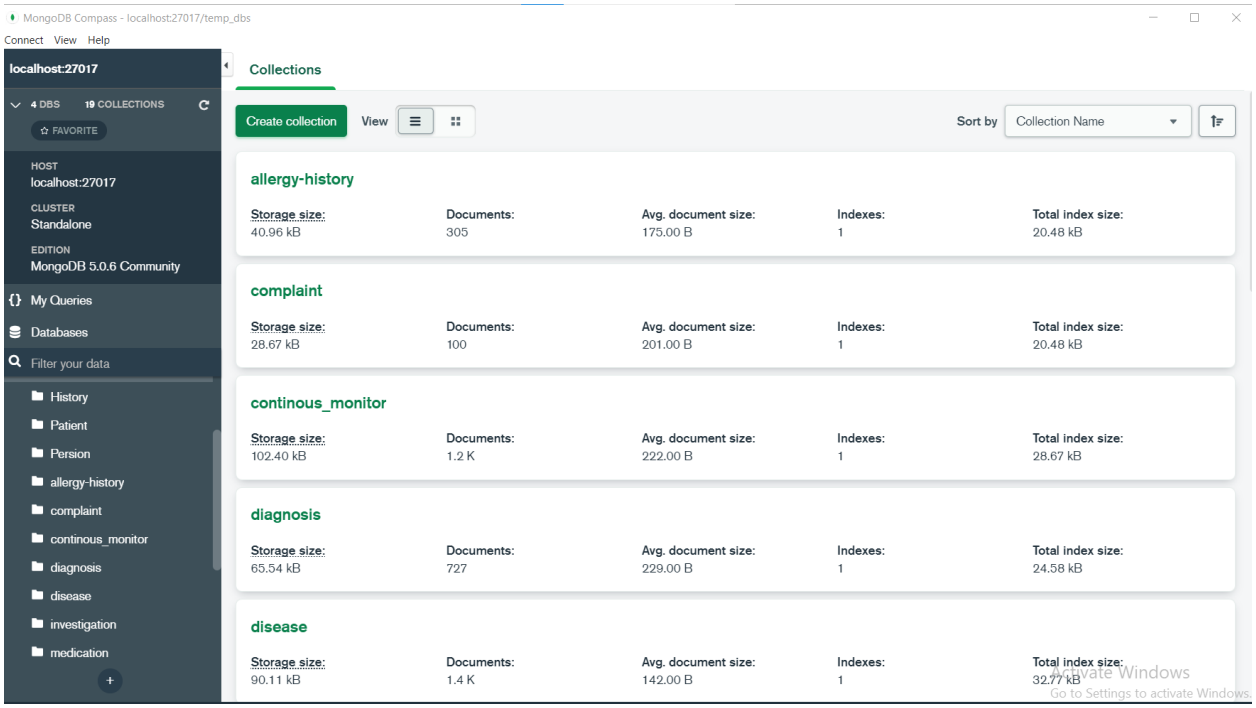
```
        treatmentEpiCount = randint(1,5)
        for y in range(1,treatmentEpiCount):
            treatmentEpiRow = {"trtEpId": treatmentEpiStart ,"ptId": personStart,"compId":
complaintStart,"startTime": getRandomDate(),"endTime": getRandomDate(),"status": "active"}
            treatment_episode.insert_one(treatmentEpiRow)

            visitCount = randint(1,5)
            for z in range(1,visitCount):
                visitType = "returning";
                if ( x== 1):
                    visitType = "new"

                visitRow = {"vId": visitStart,"ptId": personStart,"complaintId": complaintStart,"trtEpId":
treatmentEpiStart,"visitType": visitType,"visitNumber": randint(1,10),"timestamp":
getRandomDate(),"reason": randomStr(randint(1,50)),"observation": randomStr(randint(1,50)),"summery":
randomStr(randint(1,50)),"diagnosisId": randint(1,50),"sysBP": randint(60,100),"disBP":
randint(70,200),"pulse": randint(70,140),"temp": randint(97,105),"tempSource": randomStr(randint(1,50))
,"respirationRate": randint(70,100) ,"height": randint(2,200),"weight": randint(1,200),"investigationId":
randint(1,50),"treatmentPlanId":randint(1,50)}
                visit.insert_one(visitRow)

            #//one visit contain one TREATMENT PLAN so visit and treatment plan should be generated in
one loop
                treatmentPlanRow = {"treatmentPlanId":  visitStart,"ptId": personStart,"compId":
complaintStart,"trtEpId": treatmentEpiStart,"vId": visitStart,"treatmentPlanDetail":
randomStr(randint(1,100)),"treatmentPlanProcedure": randomStr(randint(1,100)),"treatmentPlanReferral":
randomStr(randint(1,100)),"treatmentPlanOtherTreatmentPlan":
randomStr(randint(1,100)),"treatmentPlanOtherTreatmentDetail":
randomStr(randint(1,100)),"medicationId": medicationStart,"surgeryId": therapyStart,"therapyId":
therapyStart,"careId":randint(1,100)}
                treatmentPlan.insert_one(treatmentPlanRow)



            #//=: one treatment plan contain multiple MEDICATION
                medicationCount = randint(1,5)
                for m in range(1,medicationCount):
                    medicationRow = {"medicationId": medicationStart,"ptId": personStart,"compId":
complaintStart,"trtEpId": visitStart,"vId": visitStart,"TreatmentPlanId":
randomStr(randint(1,20)),"MedicationName": randomStr(randint(1,5)),"MedicationDrugCode":
randomStr(randint(1,10)),"MedicationDrugIdentifier": randomStr(randint(1,2)),"MedicationStrength":
randomStr(randint(1,5)),"MedicationDose": randomStr(randint(1,4)),"MedicationRoute":
randomStr(randint(1,6)),"MedicationFrequency": randomStr(randint(1,3)),"MedicationDuration":
randomStr(randint(1,20)),"MedicationStartTimestamp": getRandomDate(),"MedicationEndTimestamp":
```

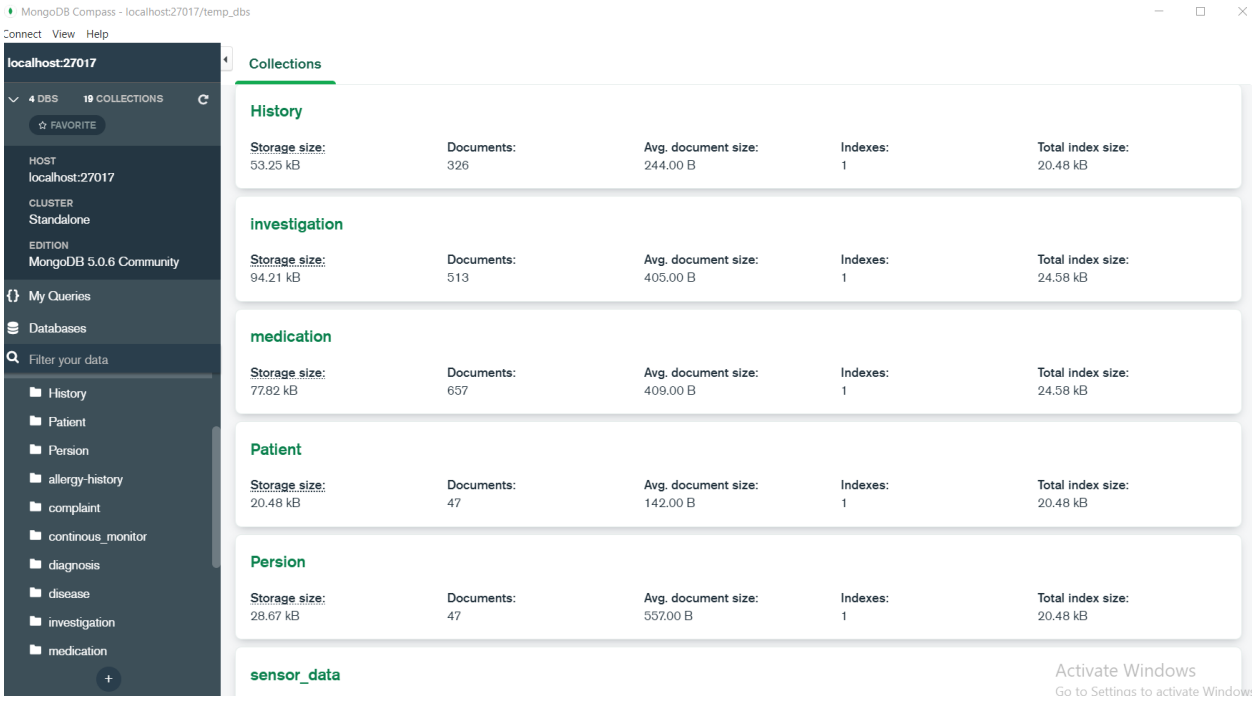
```
getRandomDate()}
                medication.insert_one(medicationRow)
                medicationStart=medicationStart+1



            #//=: one treatment plan contain multiple THERAPY
                therapyCount =randint(1,5)
                for m in range(1,therapyCount):
                    therapyRow = {"therapyId": therapyStart,"ptId": personStart,"compId":
complaintStart,"trtEpId": treatmentEpiStart,"vId": visitStart,"treatmentPlanId": visitStart,"therapyDetails":
randomStr(randint(1,30)),"therapyDuration": getRandomDate()}
                    therapy.insert_one(therapyRow)
                    therapyStart=therapyStart+1

            #//one treatment episode contains multiple visit


            #//=: one treatment plan content multiple SURGERIES
                surgeryCount = randint(1,5)
                for m in range(1,surgeryCount):
                    surgeryRow = {"surgeryId": surgeryStart,"ptId": personStart,"compId":
complaintStart,"trtEpId": treatmentEpiStart,"vId": visitStart,"treatmentPlanId":visitStart ,"surgeryName":
randomStr(randint(1,30)),"surgeryDetails": randomStr(randint(1,100)) ,"surgeryDetail":
randomStr(randint(1,200))}
                    surgerys.insert_one(surgeryRow)
                    surgeryStart=surgeryStart+1

            #//one visit contain multiple DIAGNISIS
                diagnosisCount = randint(1,5)
                for m in range(1,diagnosisCount):
                    dianosisRow = {"diagInVId": diagnosisStart,"ptId": personStart,"compId":
complaintStart,"trtEpId": treatmentEpiStart,"vId": visitStart,"diagInVType":
randomStr(randint(1,10)),"diagInVCode": randomStr(randint(1,10)),"diagInVCodeName":
randomStr(randint(1,4)),"diagInVDescription": randomStr(randint(1,100))}
                    diagnosis.insert_one(dianosisRow)

                    diseaseCount = randint(1,5)
                    for n in range(1,diseaseCount):
                        diseaseRow = { "diseaseId": diseaseStart,"diagInVId": diagnosisStart,"diseaseName":
randomStr(randint(1,10)),"diseaseDet": randomStr(randint(1,100))}

                        disease.insert_one(diseaseRow)
```

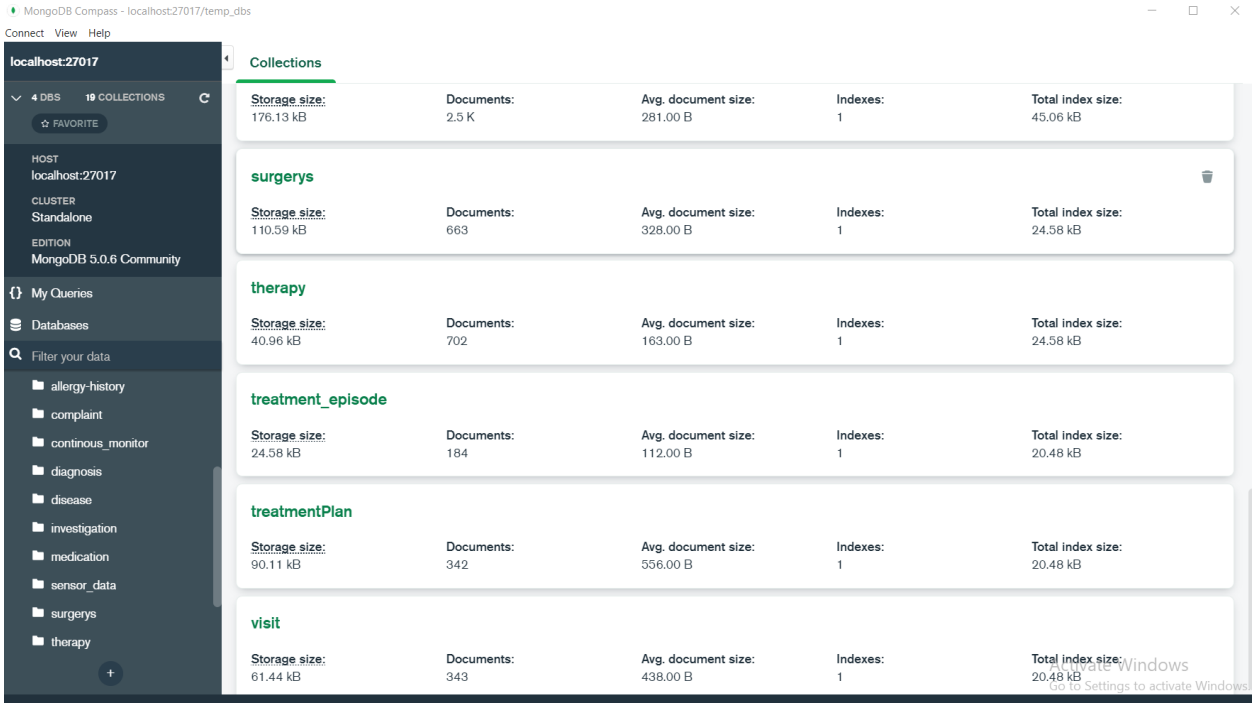
```
            diseaseStart=diseaseStart+1
        diagnosisStart=diagnosisStart+1


    investigationCount = randint(1,4)

    for m in range(1,investigationCount):
        investigationRow = {"investigationId": investigationStart,"ptId": personStart,"compId":
complaintStart,"trtEpId": treatmentEpiStart,"vId": visitStart,"drId":
randint(0,100),"investigationPrescribeTimestamp": getRandomDate(),"investigationDetail":
randomStr(randint(1,100)),"investigationResults": randomStr(randint(1,100)),"investigationTimestamp":
getRandomDate(),"investigationFile": randomStr(randint(1,100)),"CMId": continuousMoniteringStart}

        investigation.insert_one(investigationRow)

        continuousMoniteringcount = randint(1,6)

        for n in range(1,continuousMoniteringcount):
        #//echo "<br/>  continousM: ".$n;
            continuousMoniteringRow = {"CMId": continuousMoniteringStart,"ptId":
personStart,"compId": complaintStart,"trtEpId": treatmentEpiStart,"vId": visitStart,"drId":
randint(1,100),"investigationId": investigationStart,"CMStartTimeStamp":
getRandomDate(),"CMEndTimeStamp": getRandomDate(),"CMDetail": randomStr(randint(1,100))}

            continous_monitor.insert_one(continuousMoniteringRow)
            #//=: one continous monitering contain ABSTRACTED SENSOR DATA
            sensorDataCount=randint(1,5)

            for p in range(1,sensorDataCount):
                sensorDataRow = {"asdId": sensorDataStart,"ptId": personStart,"compId":
complaintStart,"trtEpId": treatmentEpiStart,"vId": visitStart,"drId": randint(1,100),"investigationId":
investigationStart,"CMId": continuousMoniteringStart,"asdId":
randomStr(randint(1,10)),"asdParameterName":
randomStr(randint(1,10)),"asdStartTimeStamp":getRandomDate() ,"asdEndTimeStamp":
getRandomDate(),"readingTaken": randomStr(randint(1,10)),"asdFile":
randomStr(randint(1,10)),"listOfSensorOutputId": randomStr(randint(1,10))}
                sensor_data.insert_one( sensorDataRow)
                sensorDataStart=sensorDataStart+1

            continuousMoniteringStart=continuousMoniteringStart+1
        investigationStart=investigationStart+1

    #//one visit contain multiple INVESTIGATION
```

```
            visitStart=visitStart+1
          treatmentEpiStart=treatmentEpiStart+1
        complaintStart=complaintStart+1
      patientStart=patientStart+1
      personStart=personStart+1
print(str(pr+oneItterationLimit)+" data generated....." )
```

Structure of the generated data(Collections):

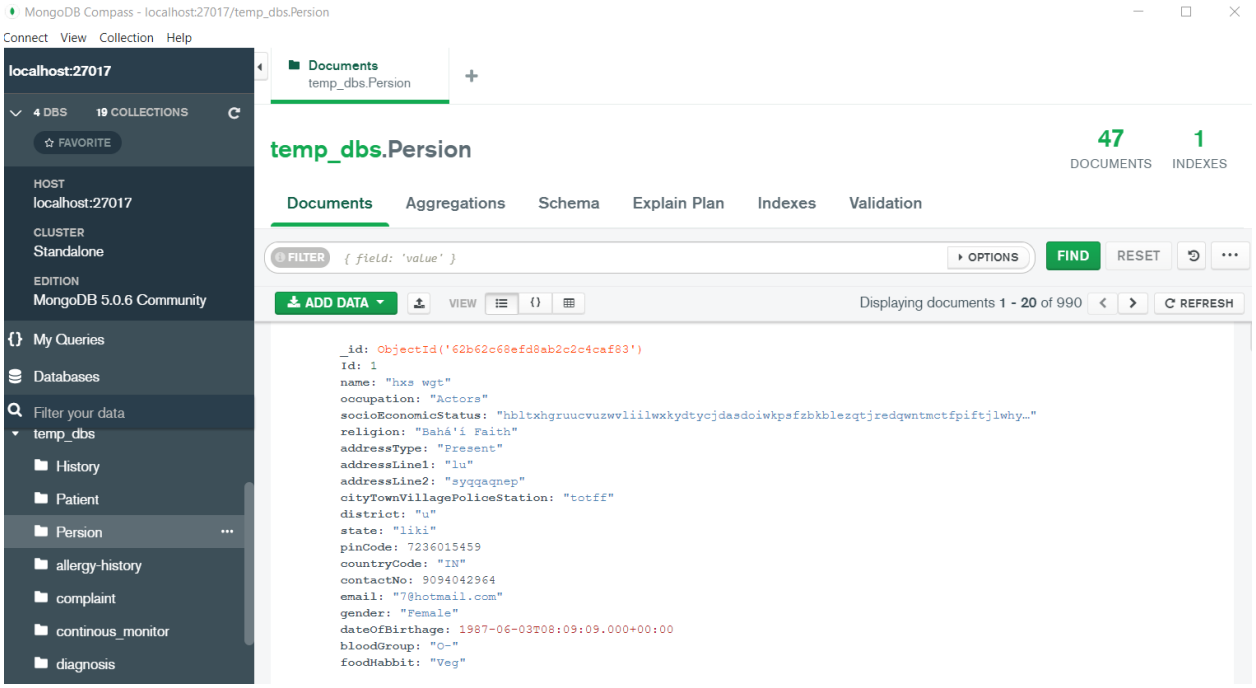



MongoDB Compass - localhost:27017/temp_dbs
Connect View Help
localhost:27017
4 DBS 19 COLLECTIONS
FAVORITE
HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 5.0.6 Community
My Queries
Databases
Filter your data
History
Patient
Persion
allergy-history
complaint
continous_monitor
diagnosis
disease
investigation
medication
Collections
History
Storage size: 53.25 kB
Documents: 326
Avg. document size: 244.00 B
Indexes: 1
Total index size: 20.48 kB
investigation
Storage size: 94.21 kB
Documents: 513
Avg. document size: 405.00 B
Indexes: 1
Total index size: 24.58 kB
medication
Storage size: 77.82 kB
Documents: 657
Avg. document size: 409.00 B
Indexes: 1
Total index size: 24.58 kB
Patient
Storage size: 20.48 kB
Documents: 47
Avg. document size: 142.00 B
Indexes: 1
Total index size: 20.48 kB
Persion
Storage size: 28.67 kB
Documents: 47
Avg. document size: 557.00 B
Indexes: 1
Total index size: 20.48 kB
sensor_data
Activate Windows
Go to Settings to activate Windows.

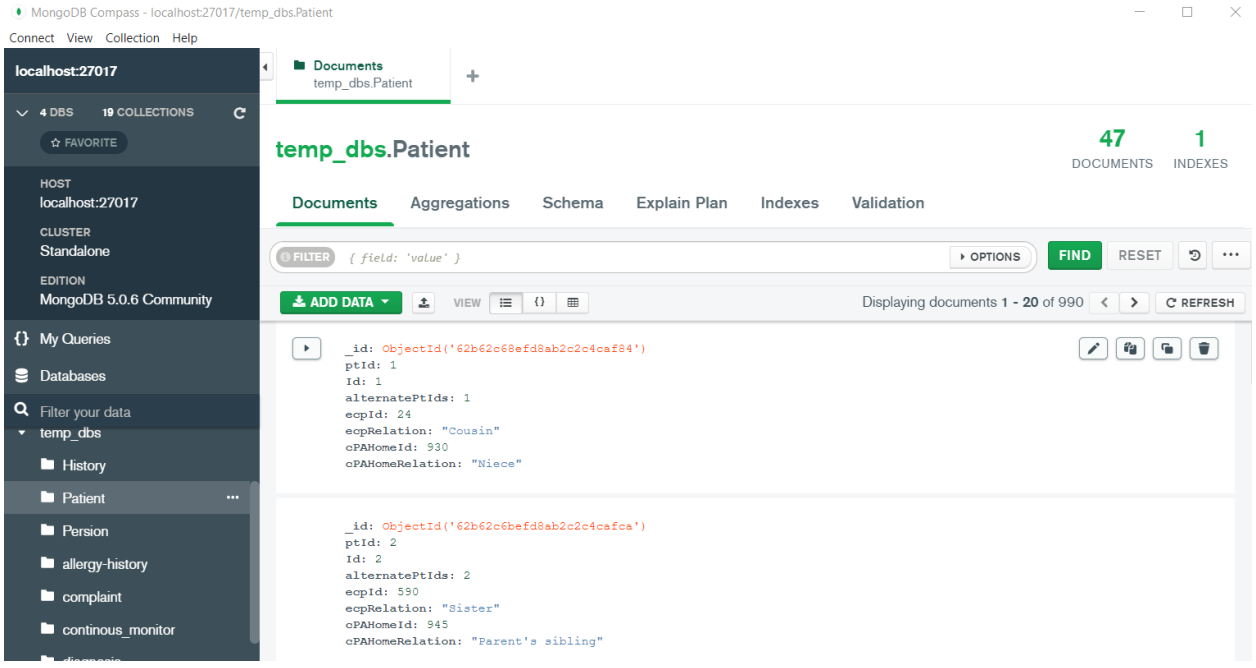

MongoDB Compass - localhost:27017/temp_dbs
Connect View Help
localhost:27017
4 DBS 19 COLLECTIONS
FAVORITE
HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 5.0.6 Community
My Queries
Databases
Filter your data
allergy-history
complaint
continous_monitor
diagnosis
disease
investigation
medication
sensor_data
surgerys
therapy
Collections
Storage size: 176.13 kB
Documents: 2.5 K
Avg. document size: 281.00 B
Indexes: 1
Total index size: 45.06 kB
surgerys
Storage size: 110.59 kB
Documents: 663
Avg. document size: 328.00 B
Indexes: 1
Total index size: 24.58 kB
therapy
Storage size: 40.96 kB
Documents: 702
Avg. document size: 163.00 B
Indexes: 1
Total index size: 24.58 kB
treatment_episode
Storage size: 24.58 kB
Documents: 184
Avg. document size: 112.00 B
Indexes: 1
Total index size: 20.48 kB
treatmentPlan
Storage size: 90.11 kB
Documents: 342
Avg. document size: 556.00 B
Indexes: 1
Total index size: 20.48 kB
visit
Storage size: 61.44 kB
Documents: 343
Avg. document size: 438.00 B
Indexes: 1
Total index size: 20.48 kB
Activate Windows
Go to Settings to activate Windows.

Structure of Documents:

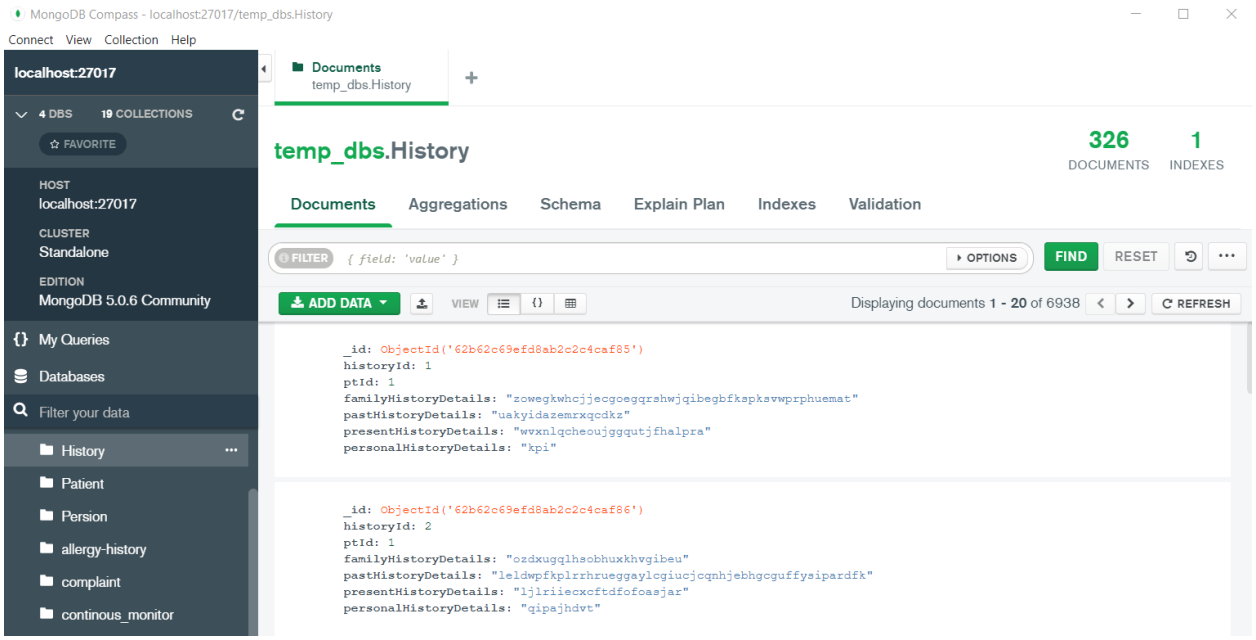
1. View of Person document(data):



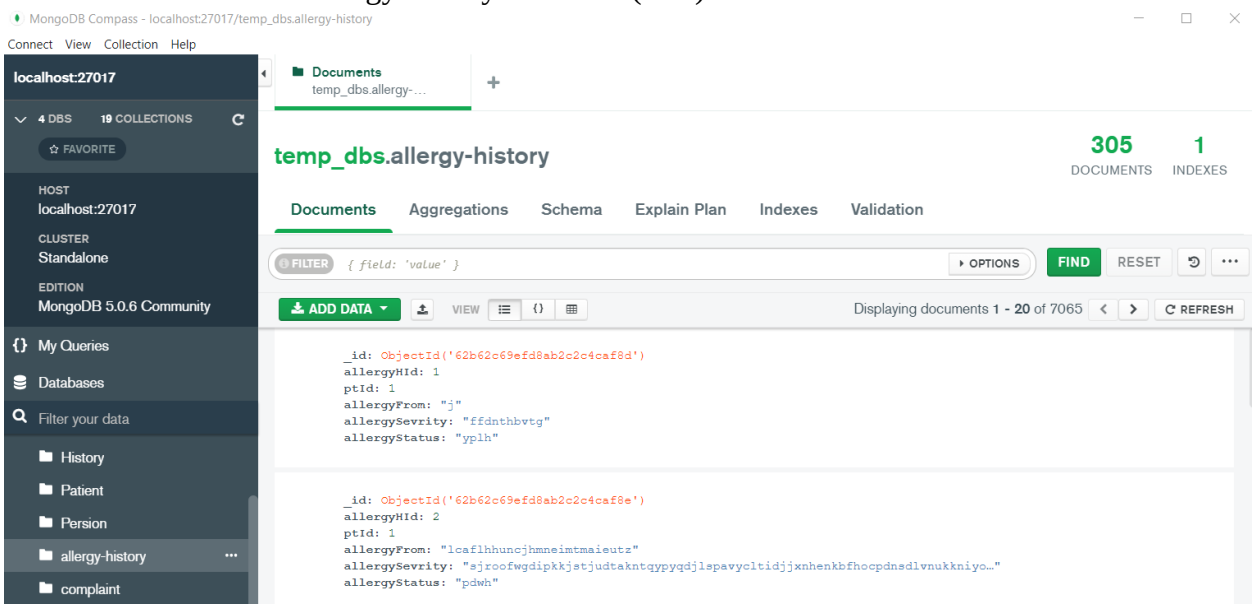
2. View of Patient document(data):



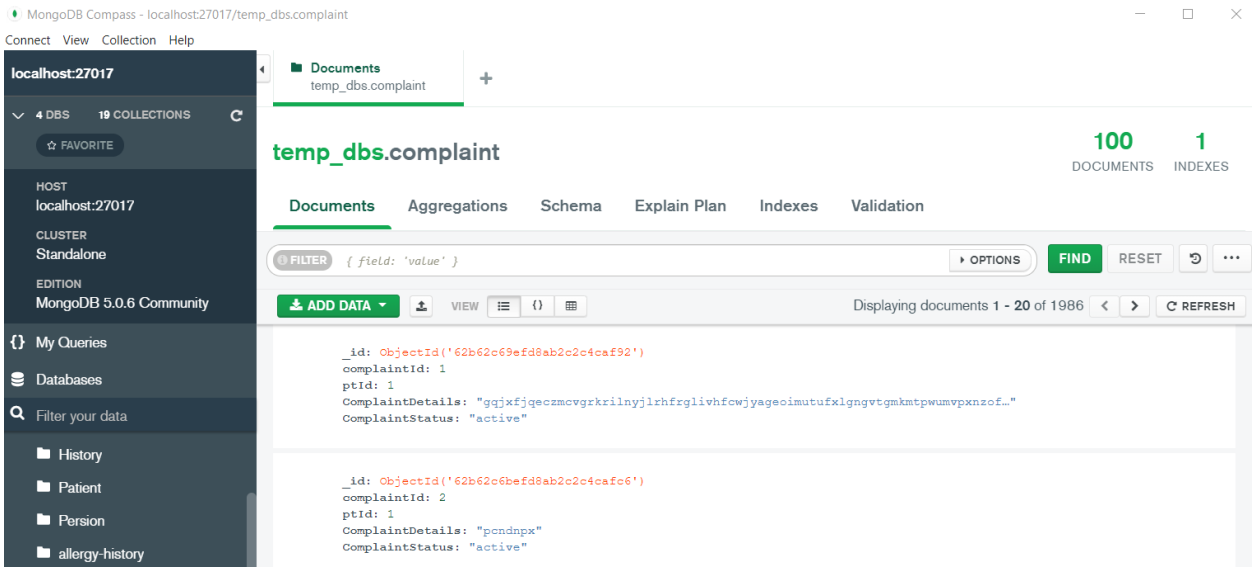
3. View of Doctor document(data):
4. View of History document(data):



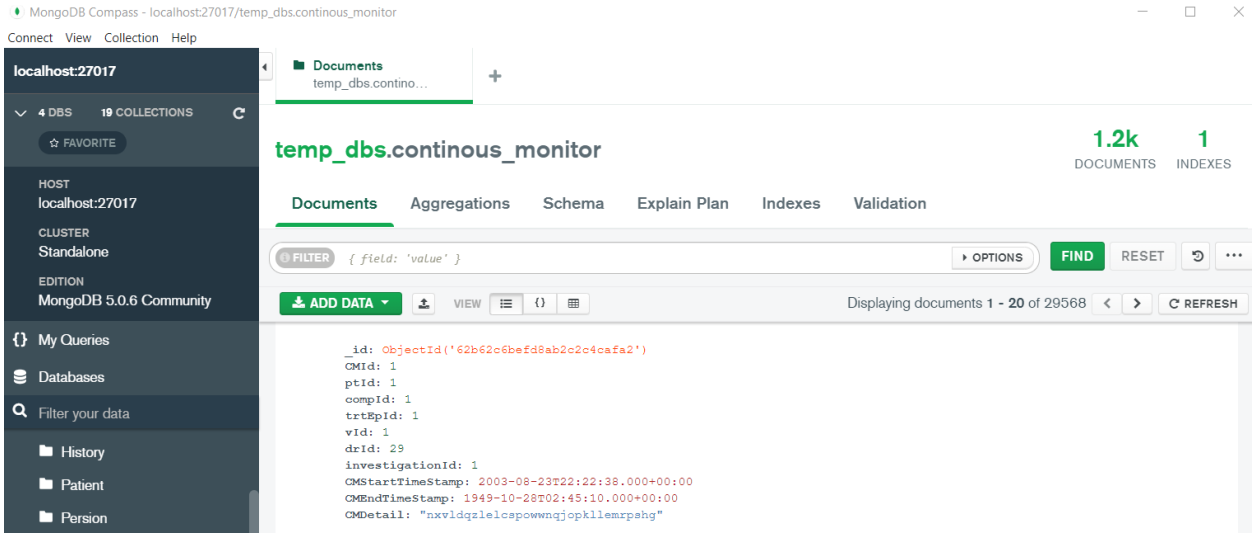
5. View of Allergy-history document(data):



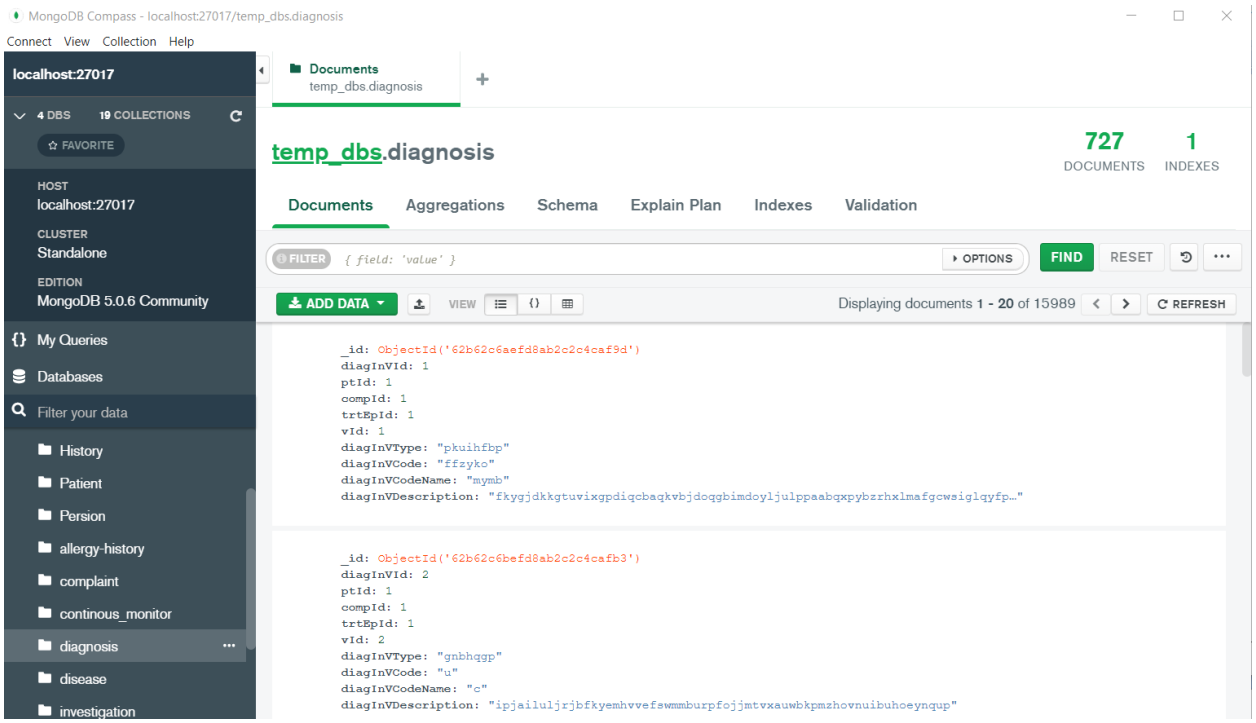
6. View of Complaint document(data):



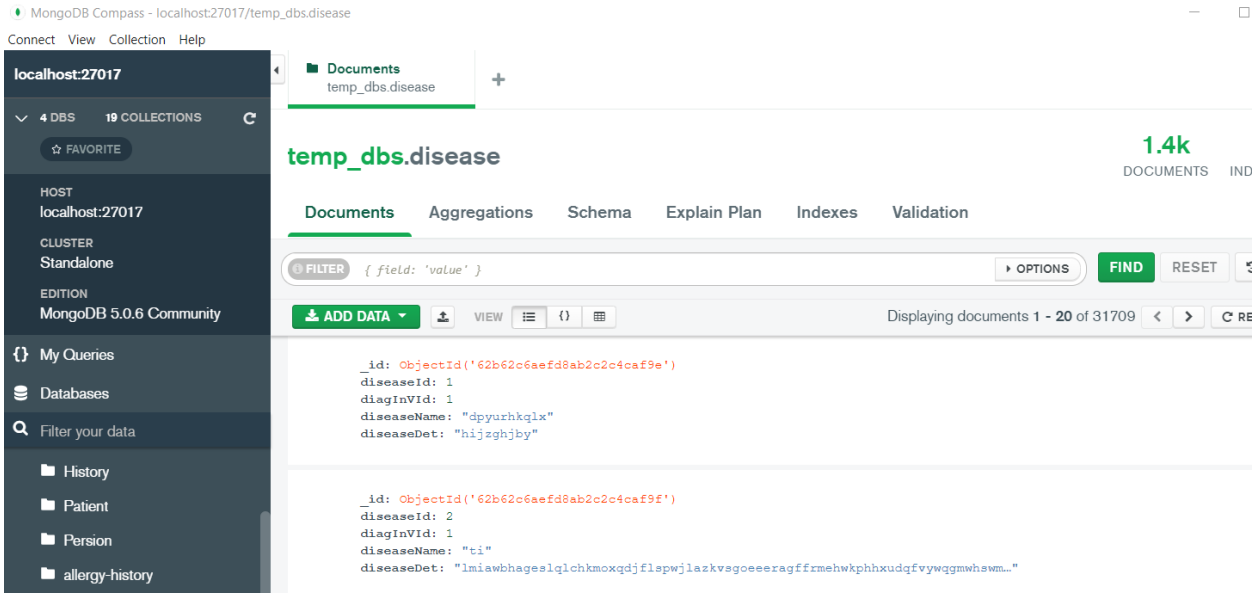
7. View of Continuous-monitor document(data):



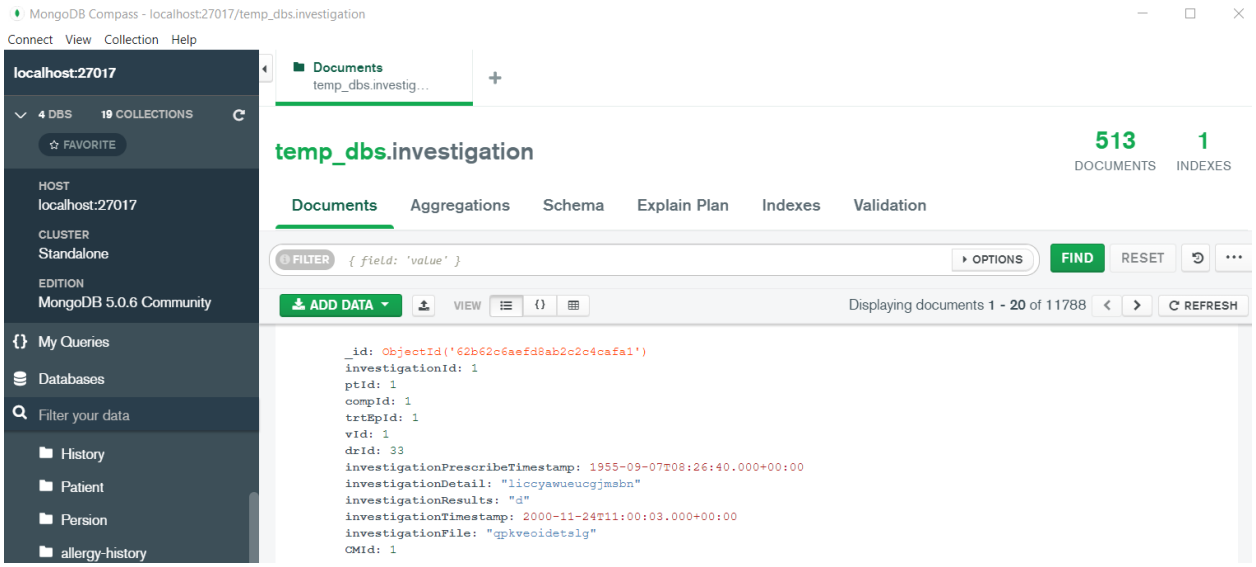
8. View of Diagnosis document(data):



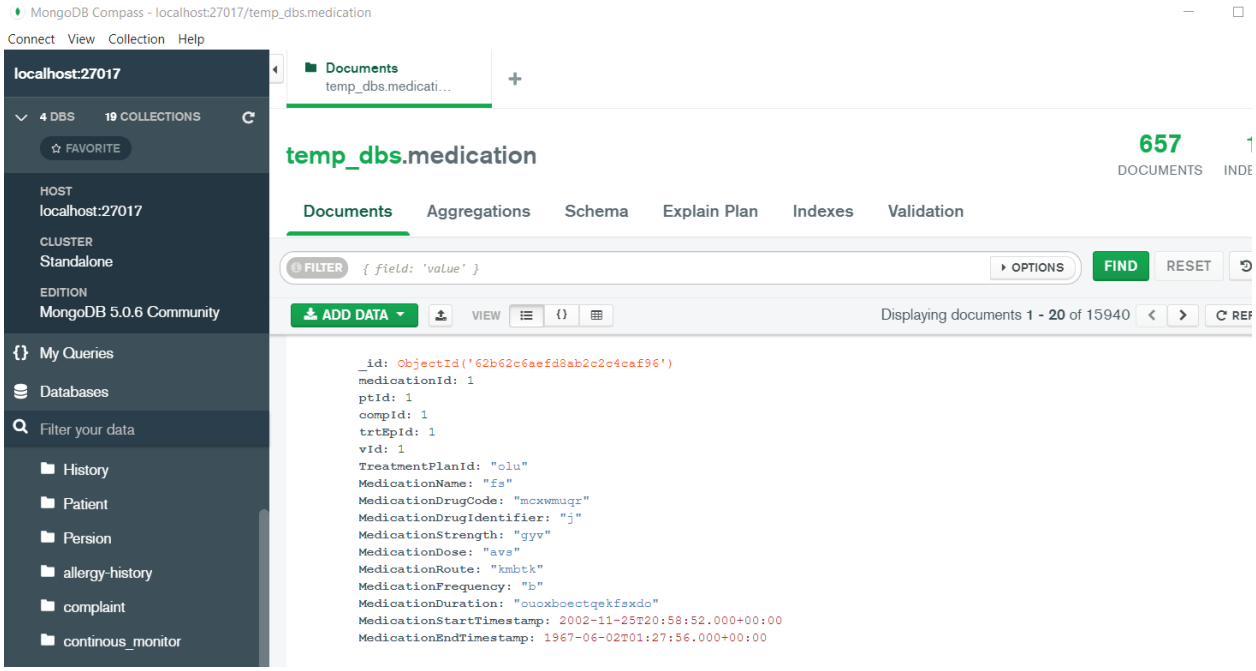
9. View of Disease document(data):



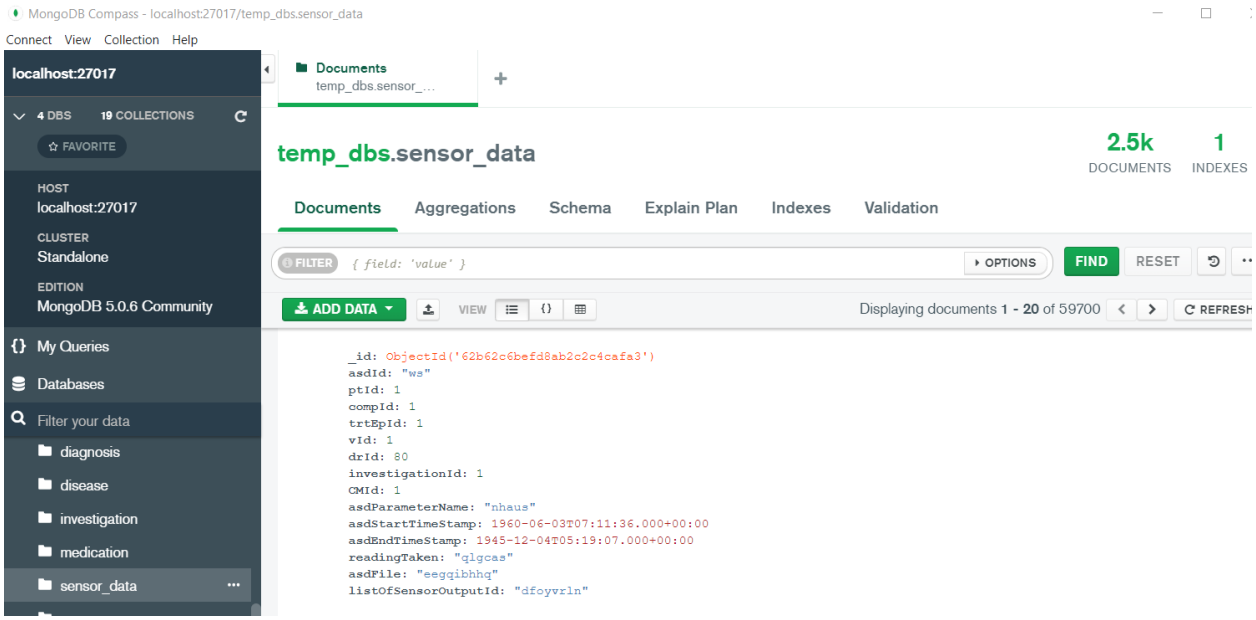
10. View of Investigation document(data):



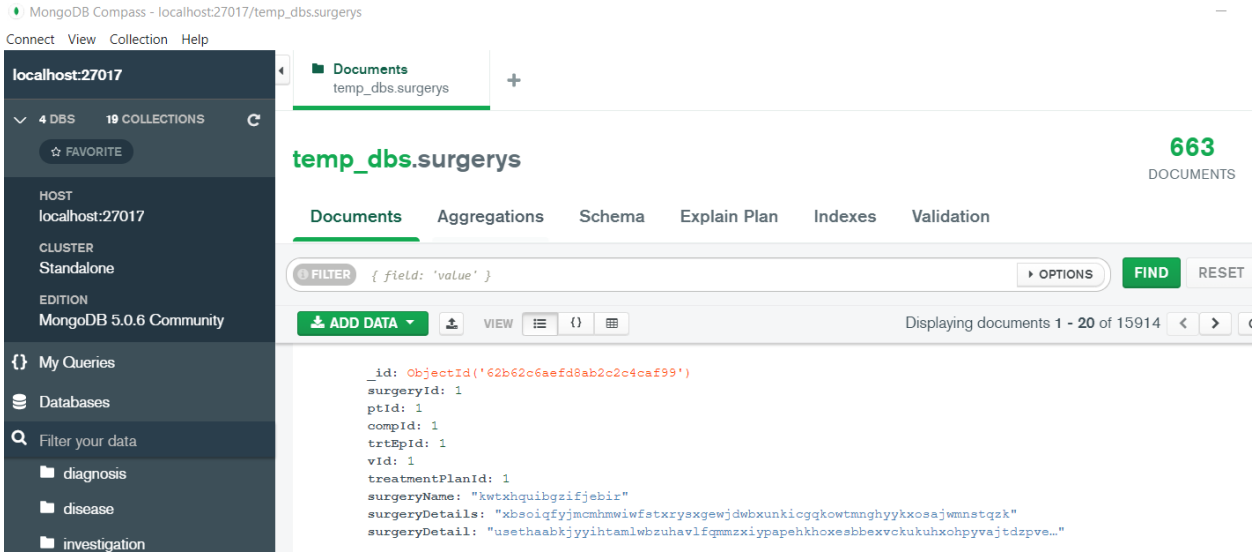
11. View of Medication document(data):



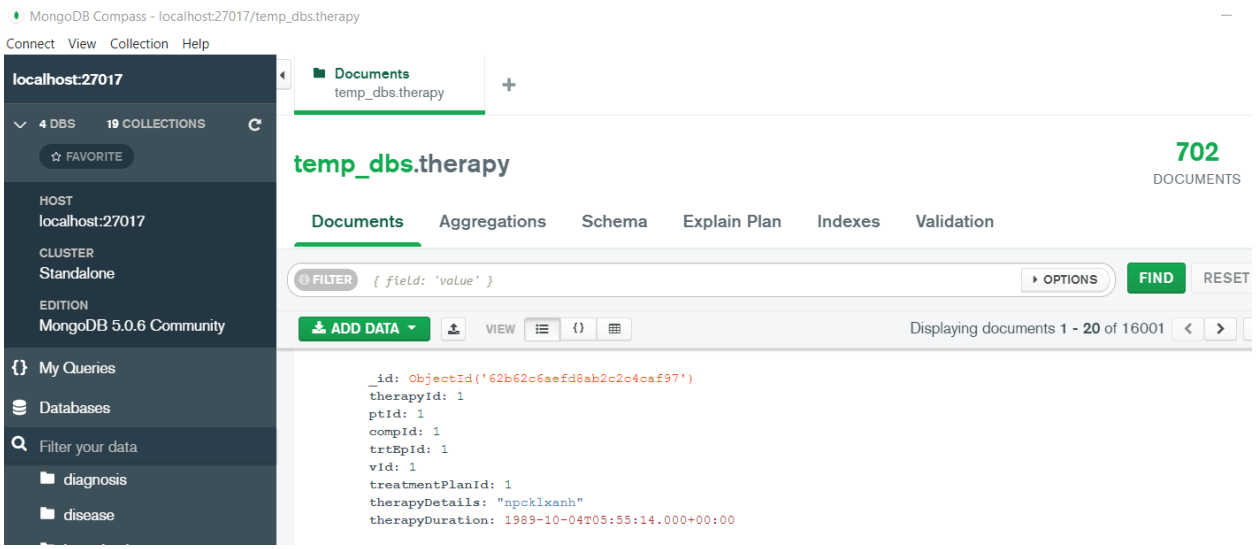
12. View of Sensor-data document(data):



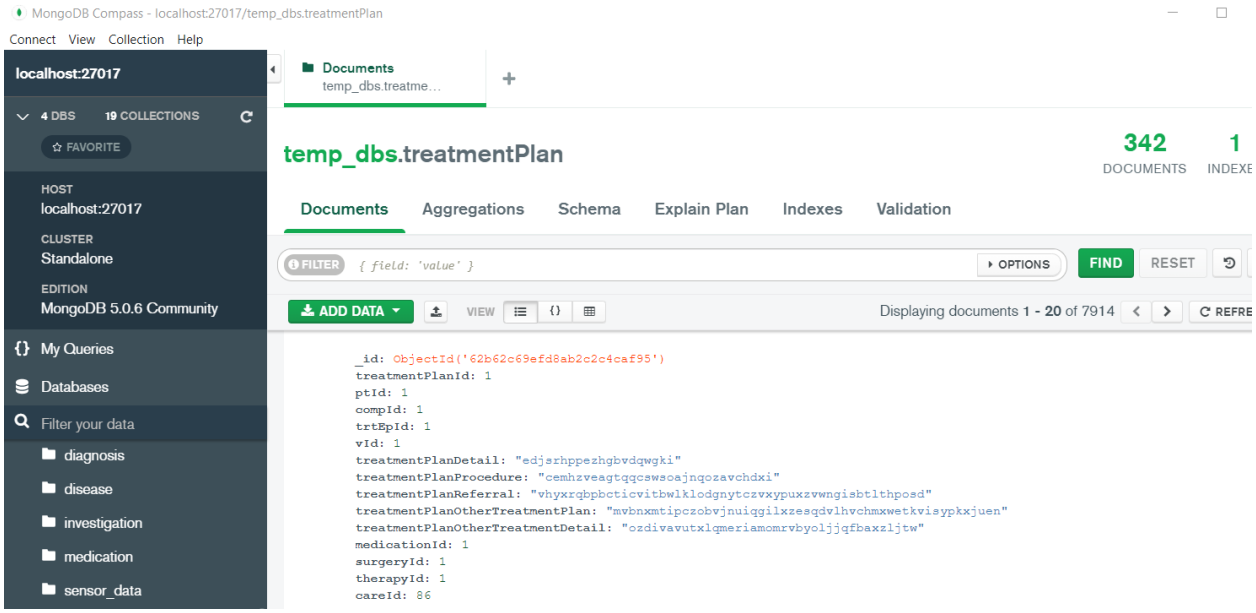
13. View of Surgery document(data):



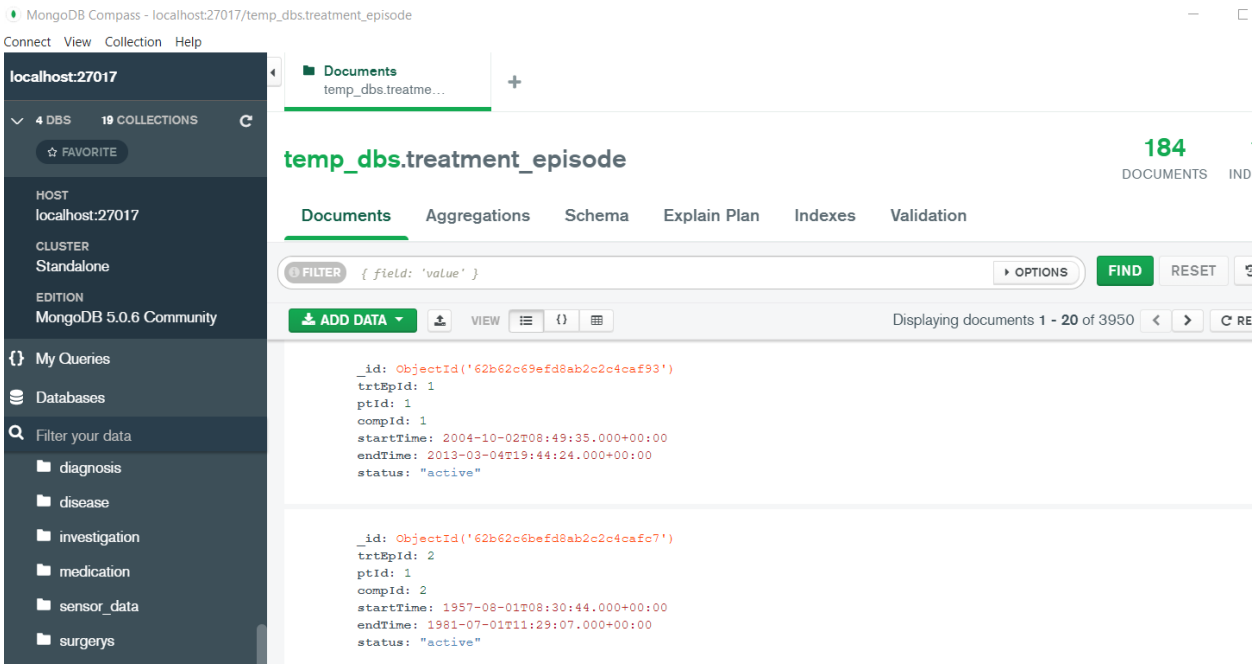
14. View of Therapy document(data):



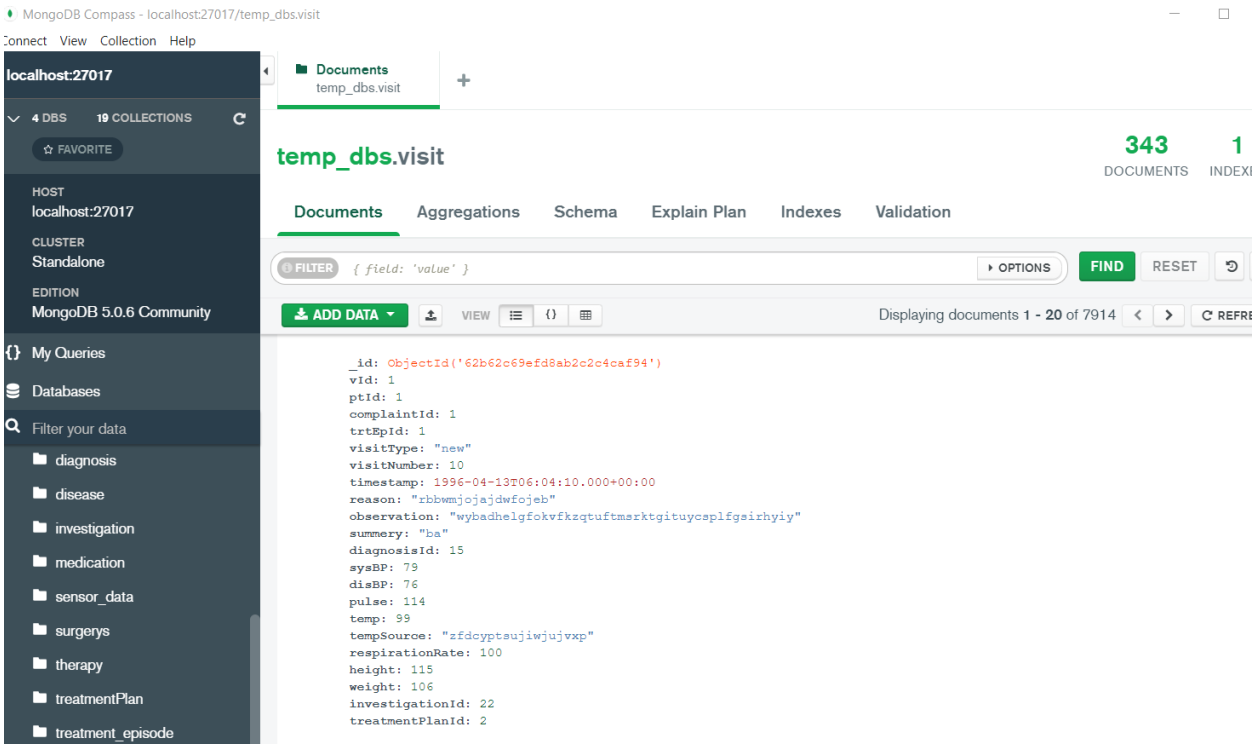
15. View of Treatment-Plan document(data):



16. View of Treatment-episode document(data):



17. View of visit document(data):



CHAPTER # 5

RESULT OF THE EXPERIMENT

Contents:

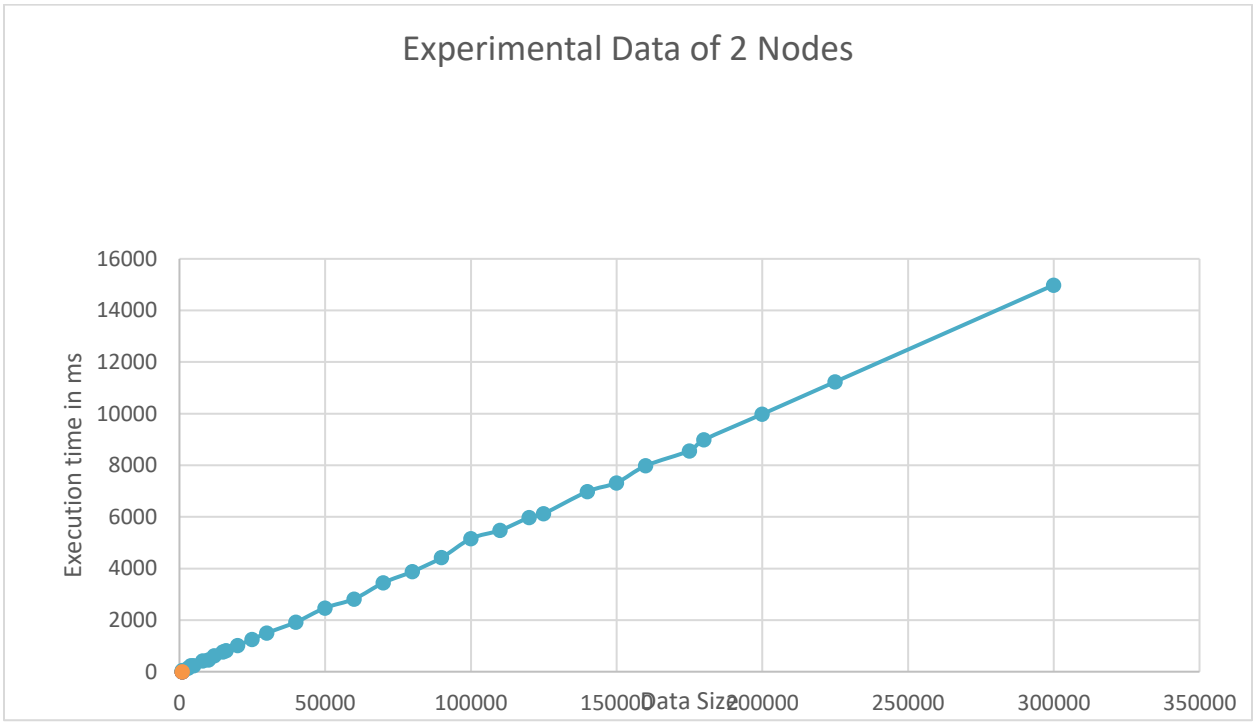
- Experimental data on different aspects
- Comparison between all aspects

Experimental data on different aspects:

1. **Two node with different data size:**
   we make MongoDB-cluster with two node and execute search query on data of different sizes.

1	size of the dataset	MongoDB query execution time(ms)
2	1000	56.38
3	3000	140.34
4	4000	233.79
5	5000	237.7
6	8000	428.61
7	10000	453.8
8	12000	623.43
9	15000	769.55
10	16000	818.25
11	20000	1017.8
12	25000	1256.59
13	30000	1500.12
14	40000	1924.2
15	50000	2474.21
16	60000	2821.8
17	70000	3448.3
18	80000	3883.4
19	90000	4422.4
20	100000	5159.2
21	110000	5479.8
22	120000	5976.53
23	125000	6127.06
24	140000	6982.57
25	150000	7322.68
26	160000	7982.3
27	175000	8562
28	180000	8987.9
29	200000	9983.46
30	225000	11234.77
31	300000	14986.39

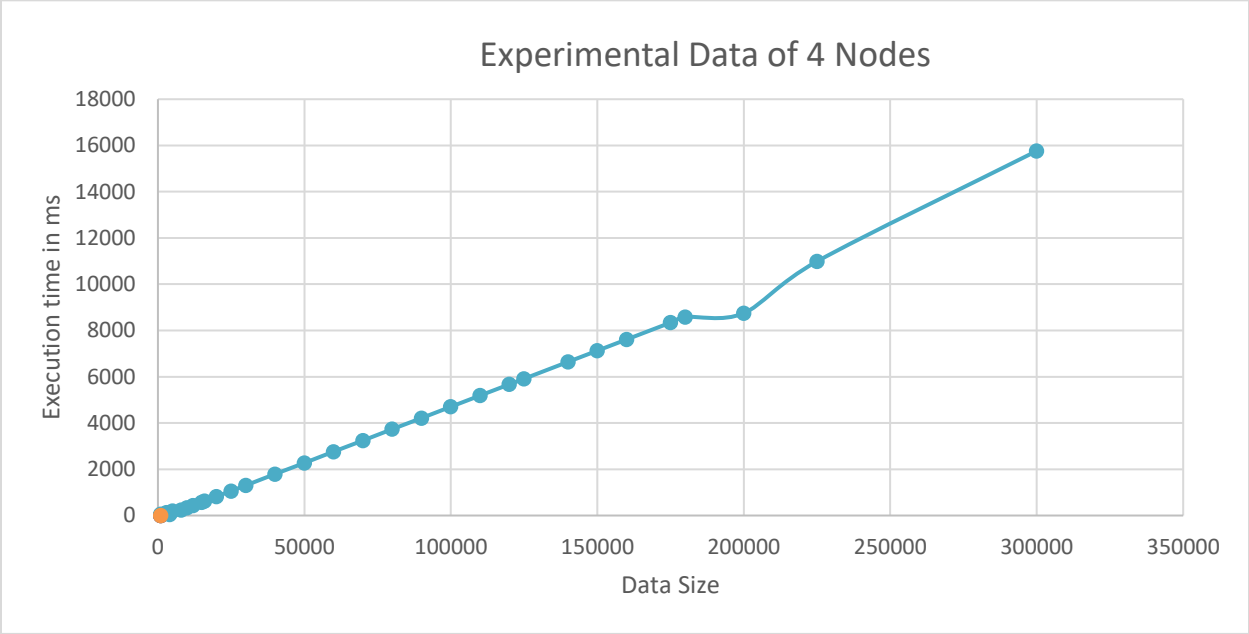
Graph:



2. **Four node with different data size:**
   Here we increase the number of nodes and repeat.

	A	B
1	size of the dataset	MongoDB query execution time(ms)
2	1000	52.98
3	3000	130.81
4	4000	39.96
5	5000	196.21
6	8000	234.11
7	10000	338.67
8	12000	428.26
9	15000	573.87
10	16000	622.41
11	20000	823.58
12	25000	1059.24
13	30000	1301.93
14	40000	1793.41
15	50000	2272.68
16	60000	2763.24
17	70000	3243.43
18	80000	3733.07
19	90000	4214.17
20	100000	4702.9
21	110000	5187.82
22	120000	5672.73
23	125000	5912.98
24	140000	6642.57
25	150000	7126.41
26	160000	7612.4
27	175000	8339.85
28	180000	8582.23
29	200000	8745.3
30	225000	10987.1
31	300000	15761.8

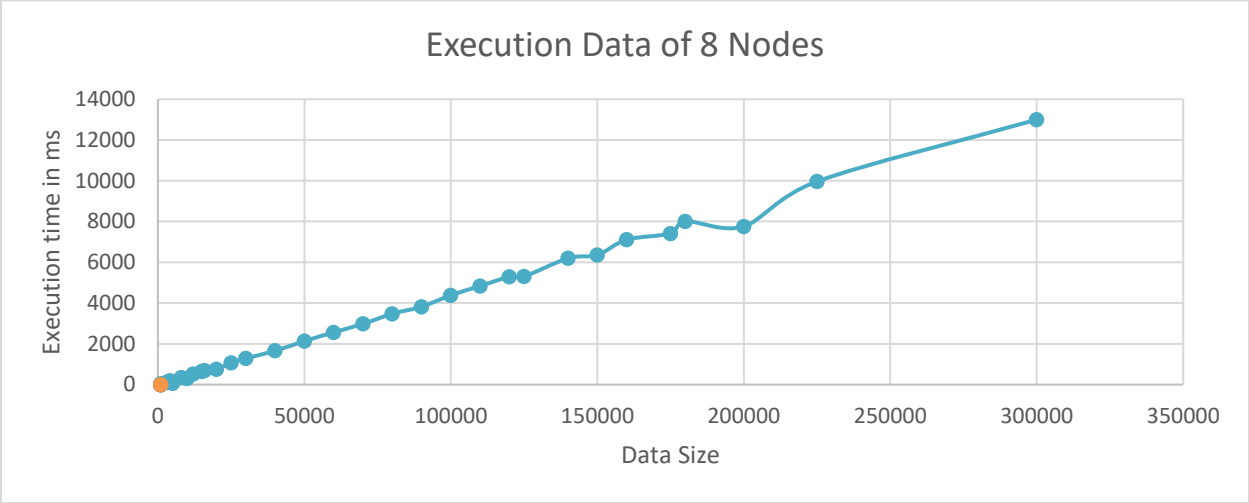
Graph:



3. **Eight node with different data size**:
   Here we increase the number of nodes(8) and repeat.

1	size of the dataset	MongoDB query execution time(ms)
2	1000	49.93
3	3000	110.2
4	4000	184.13
5	5000	68.12
6	8000	353.34
7	10000	295.15
8	12000	522.55
9	15000	649.46
10	16000	691.77
11	20000	749.22
12	25000	1072.49
13	30000	1284.01
14	40000	1657.36
15	50000	2130.08
16	60000	2565.5
17	70000	2976.14
18	80000	3473.64
19	90000	3822.2
20	100000	4381.78
21	110000	4835.85
22	120000	5289.92
23	125000	5302.82
24	140000	6198.06
25	150000	6360.4
26	160000	7106.2
27	175000	7417.98
28	180000	8014.33
29	200000	7754.9
30	225000	9963.5
31	300000	12997.4

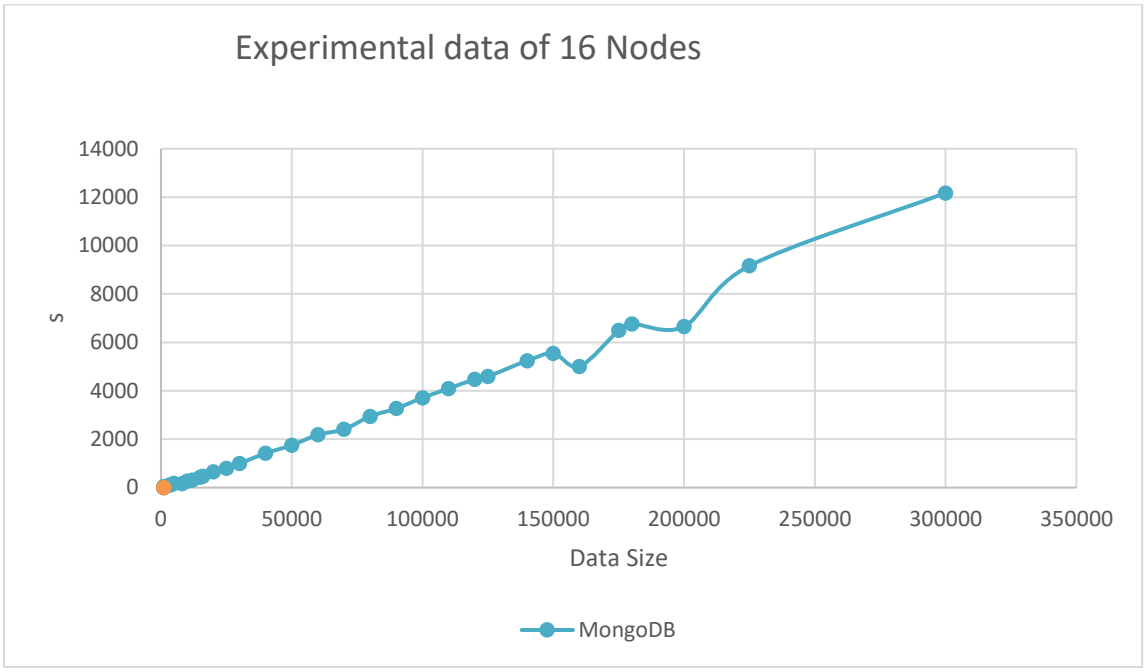
Graph:



4. **Sixteen node of different data size:**
   Here we increase the number of nodes(16) and repeat.

1	size of the dataset	MongoDB
2	1000	45.56
3	3000	103.6
4	4000	119.4
5	5000	178.78
6	8000	158.85
7	10000	269.94
8	12000	310.49
9	15000	424.2
10	16000	462.12
11	20000	652.24
12	25000	803.29
13	30000	992.84
14	40000	1416.85
15	50000	1751.1
16	60000	2181.47
17	70000	2409.17
18	80000	2946.08
19	90000	3267.34
20	100000	3710.69
21	110000	4093
22	120000	4475.3
23	125000	4594.14
24	140000	5239.91
25	150000	5541.85
26	160000	5006.52
27	175000	6498.56
28	180000	6769.14
29	200000	6656.6
30	225000	9163.32
31	300000	12175.1

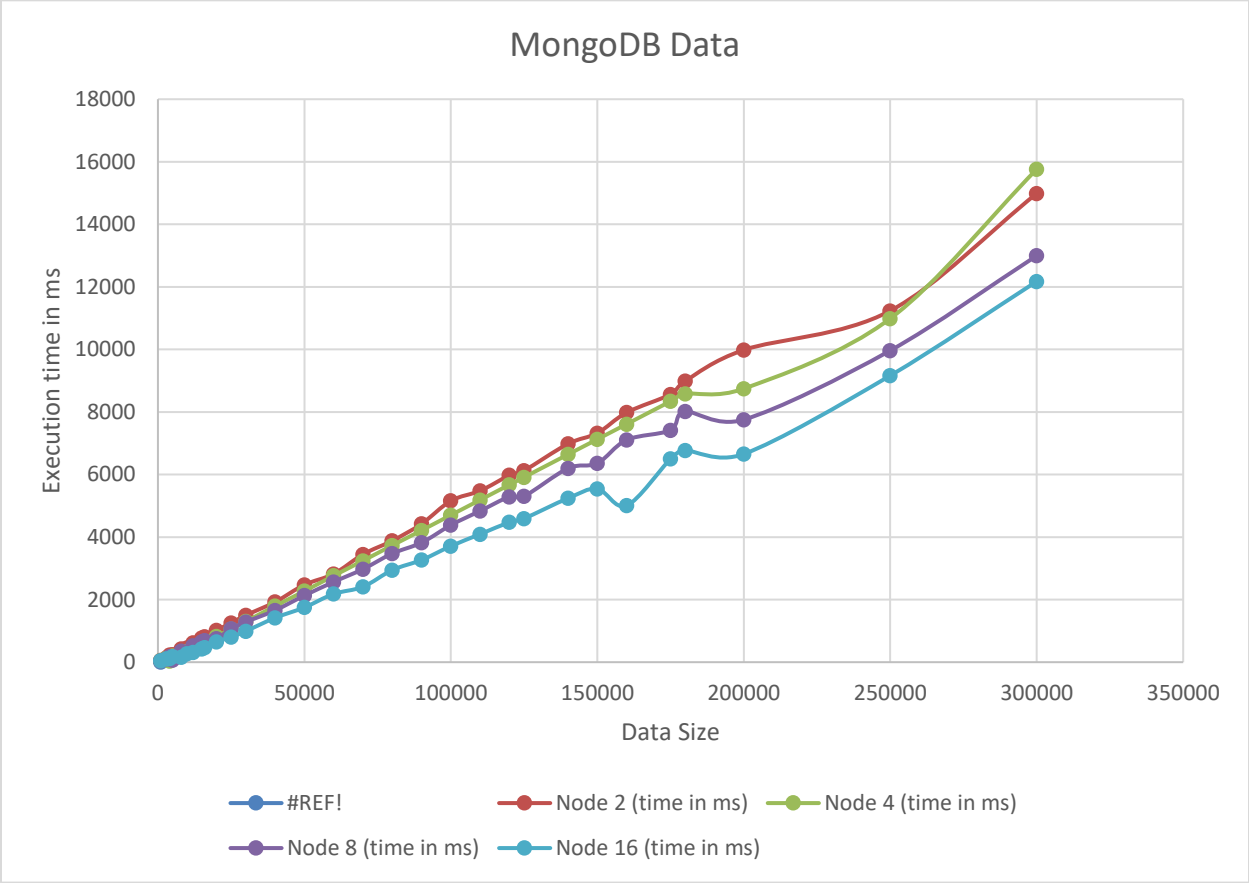
Graph:


Experimental data of 16 Nodes
14000
12000
10000
8000
6000
4000
2000
0
s
0
50000
100000
150000
200000
250000
300000
350000
Data Size
MongoDB

Comparison between all aspects:

1	size of the dataset	Node 2 (time in ms)	Node 4 (time in ms)	Node 8 (time in ms)	Node 16 (time in ms)
2	1000	56.38	52.98	49.93	45.56
3	3000	140.34	130.81	110.2	103.6
4	4000	233.79	39.96	184.13	119.4
5	5000	237.7	196.21	68.12	178.78
6	8000	428.61	234.11	353.34	158.85
7	10000	453.8	338.67	295.15	269.94
8	12000	623.43	428.26	522.55	310.49
9	15000	769.55	573.87	649.46	424.2
10	16000	818.25	622.41	691.77	462.12
11	20000	1017.8	823.58	749.22	652.24
12	25000	1256.59	1059.24	1072.49	803.29
13	30000	1500.12	1301.93	1284.01	992.84
14	40000	1924.2	1793.41	1657.36	1416.85
15	50000	2474.21	2272.68	2130.08	1751.1
16	60000	2821.8	2763.24	2565.5	2181.47
17	70000	3448.3	3243.43	2976.14	2409.17
18	80000	3883.4	3733.07	3473.64	2946.08
19	90000	4422.4	4214.17	3822.2	3267.34
20	100000	5159.2	4702.9	4381.78	3710.69
21	110000	5479.8	5187.82	4835.85	4093
22	120000	5976.53	5672.73	5289.92	4475.3
23	125000	6127.06	5912.98	5302.82	4594.14
24	140000	6982.57	6642.57	6198.06	5239.91
25	150000	7322.68	7126.41	6360.4	5541.85
26	160000	7982.3	7612.4	7106.2	5006.52
27	175000	8562	8339.85	7417.98	6498.56
28	180000	8987.9	8582.23	8014.33	6769.14
29	200000	9983.46	8745.3	7754.9	6656.6
30	250000	11234.77	10987.1	9963.5	9163.32
31	300000	14986.39	15761.8	12997.4	12175.1

Graph:



Comparison:- That graph show that if the number of nodes increases and executes a query on the same size of data then it takes less time .when we work with two nodes it takes the highest time. After that, we increased the number of nodes and execute a query on the same size of new data . Then we saw that the performance of the cluster is increased when the number of nodes is also being increased.

CHAPTER # 6

CONCLUSION

DISCUSSION

The test result shows that in all of the cases, the cluster provides a better result when the number of more then other. Alternative the most of the configuration management take more time and also to generate the data is time consuming . The user interface of the MongoDB Compus needs a thorough knowledge about its infrastructural changes as it is tangible and nontangible cost is also changes accordingly.

CONCLUSION

In this experimental study, we have constructed different aspect on various size of data as well as different number of nodes. These are used to evaluate theperformance for the development of different operations. Theevaluation of performance on the cluster domains illustrates the comparative merits and demerits of integratingMongoDB for different types of operations used in common applications. If the number of node increasing the MongoDB cluster give better performance . We have also shown that the query takes more time to execute if data size is increased . The stored data of the same file takes less time if we run same query of same data set. In this experimental evaluation, the cost of any type (tangible cost or non-tangible cost) is taken into consideration. In future, itmight be interesting to explore whether the security of MongoDB cluster can be enhanced by integrating it along with the different technology.

Bibliography

BIBLIOGRAPHY

- Wikipedia
- https://www.digitalocean.com
- https://www.mongodb.org/
- https://www.python.org/
- https://www.geeksforgeeks.org/python/

REFERENCE BOOKS

Scaling Mongodb: Sharding, Cluster Setup, And Administration Paperback – 1 January 2015