Practicum #3: Non-Relational Database

Project Report

DATABASE MANAGEMENT AND DATABASE ANALYTICS (MIS-64082-003)

Presented by: Tejaswini Yeruva

Step 1.

A hypothetical scenario where a non-relational database is the best solution is a marketing agency that needs to store information about its clients and campaigns. The agency has a large number of clients with different needs and campaigns that vary in size and complexity. The agency needs to quickly access, store, and query the data for analytics and insights

Step 2.

The NoSQL database used in this scenario should be document-oriented. Document-oriented databases are better suited for storing large amounts of data that do not have a predictable structure. The data in this scenario is highly unstructured, with clients and campaigns that vary in size and complexity. Document-oriented databases are also highly scalable, allowing the marketing agency to store large amounts of data.

Step 3.

Document-oriented databases are also easier to use than relational databases. The agency does not need to define a schema beforehand, meaning that the data can be easily stored and queried. The agency can also quickly access the data without having to write complex queries. Document-oriented databases are also well suited for data that is constantly changing, as new clients and campaigns can be easily added without having to modify the schema.

Step 4.

In comparison to other NoSQL databases, document-oriented databases are the best choice for this scenario. Key-value stores are not suitable for this scenario as they are limited in their ability to store complex data. Graph databases are also not suitable as they are better suited for data that have relationships between entities.

Step 5.

The document-oriented database should also be able to handle large amounts of data. MongoDB is a good choice for this scenario as it is highly scalable and can handle large amounts of data. MongoDB also has an easy-to-use query language, making storing and retrieving data easy. It also has built-in indexing and sharding, which allows it to easily scale as the number of data increases.

Step 7.

A prototype can be created to demonstrate the functionality of a document-oriented database. The prototype should include example records for clients and campaigns, such as name, contact information, budget, and duration. The prototype should also include example records for clients and campaigns with different complexity levels, such as clients with multiple campaigns and campaigns with multiple goals.

Step 8.

The prototype should also include example queries that are used to access the data. These queries should demonstrate how the data can be easily accessed, stored, and queried. For example, queries can be used to find all clients who have a budget over a certain amount or to find all campaigns that have a certain goal.

Step 9.

The document-oriented database is the best solution for this scenario as it allows for the storage of large amounts of data that is highly unstructured and constantly changing. It is also easy to use, with an intuitive query language and built-in indexing and sharding. A prototype can be created to demonstrate the functionality of this solution and to provide example records and queries.

Q1

The scenario (industry, firm, information requirements, data requirements)

Answer:

Industry: Healthcare

Sector: Healthcare Information

Patient experience surveys, medical records, billing information, and patient records are requirements. Patient name, dob, medical history, insurance information, billing information, and survey responses are all necessary data elements.

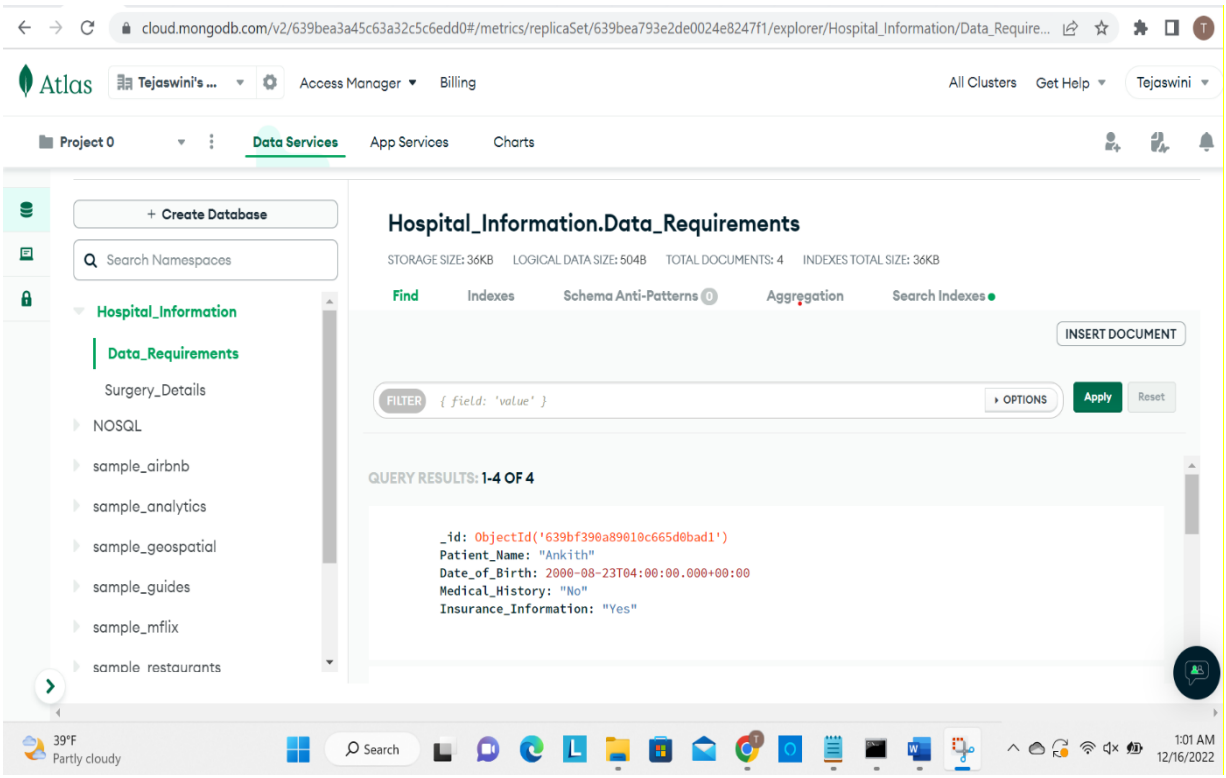
Steps for establishing a MongoDB database collection:

1. I made a **MongoDB** Atlas account.

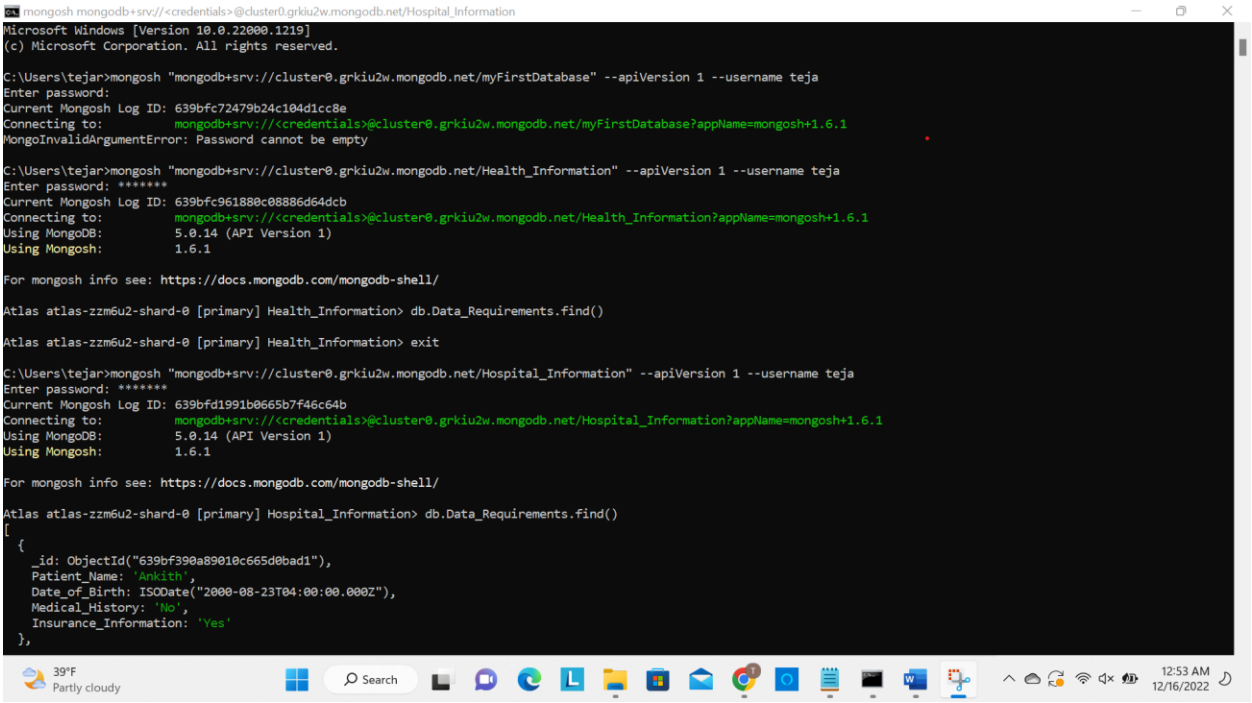
2. A cluster that contains a Username and Password that may be used to access the database cloud was created.

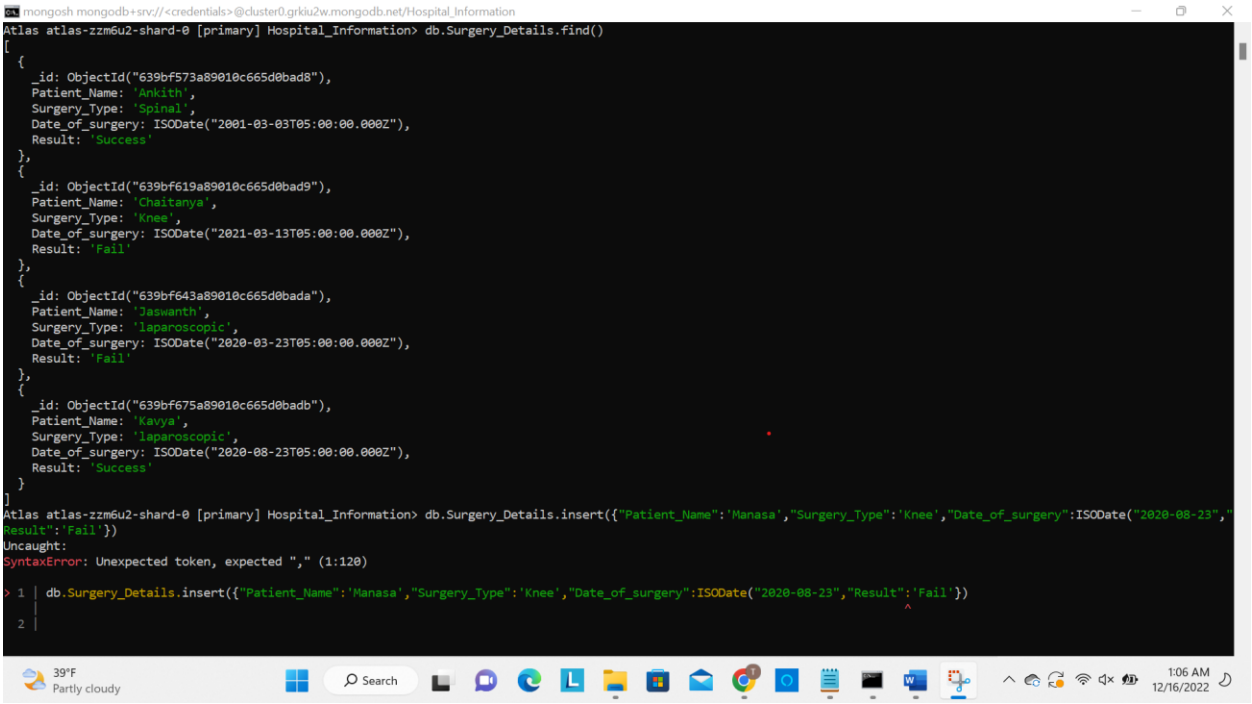
3. After connecting the MongoDB shell (Command prompt) to the local desktop, constructed a collection that includes all the Hospital information from the sample provided above.

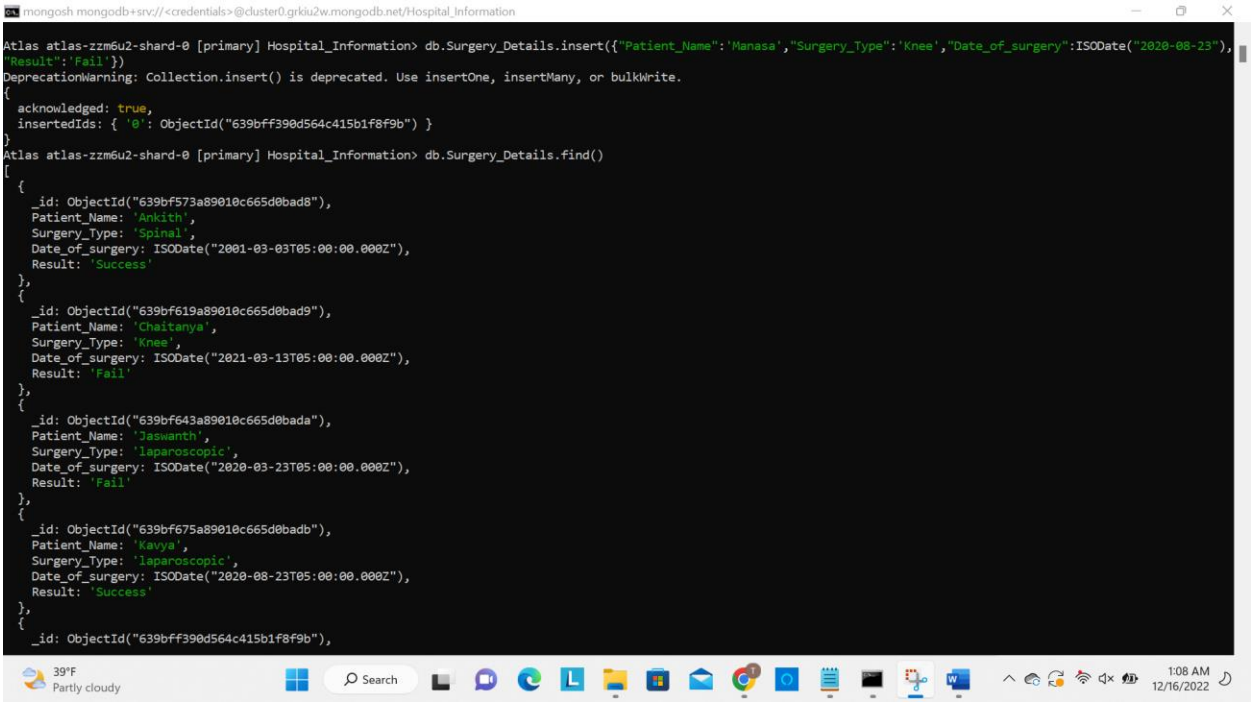
The database that was established contains collections for surgery details and data requirements under the category of “Hospital Information."

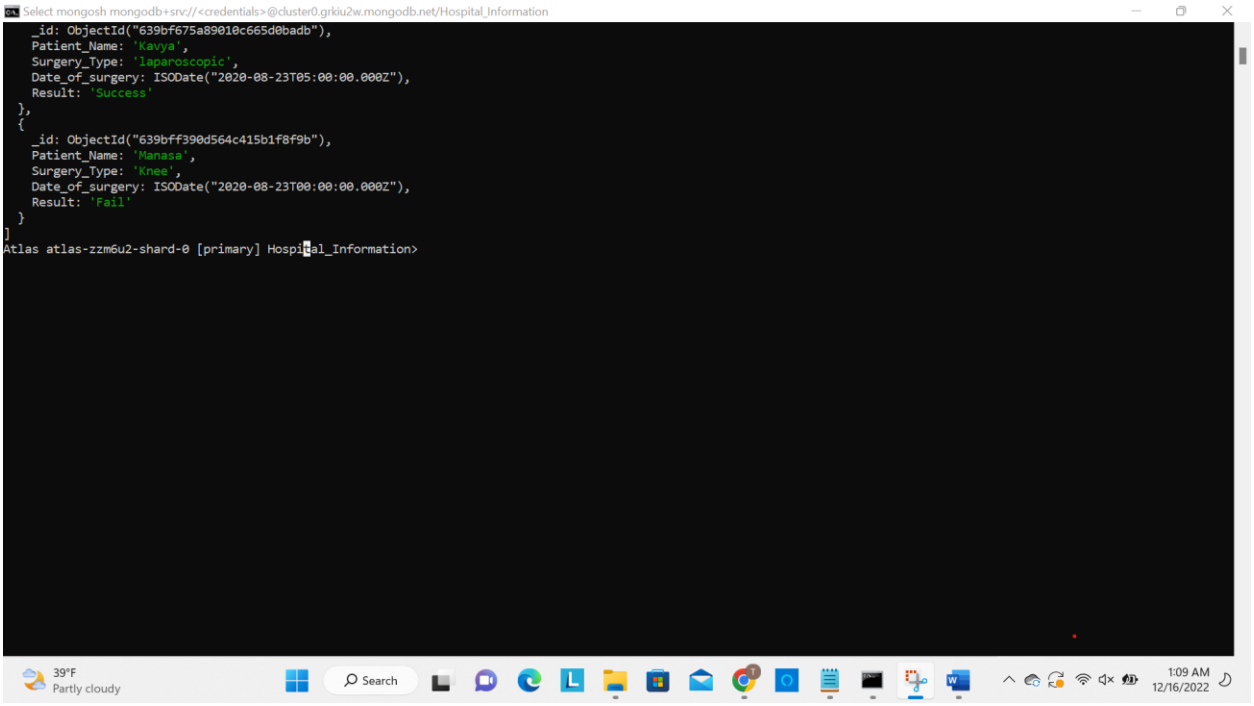


Information can be acquired via the command shell by utilizing commands like **insert**, **find**, and **findOne**.


mongosh mongodb+srv://<credentials>@cluster0.grkiu2w.mongodb.net/Hospital_Information
Microsoft Windows [Version 10.0.22000.1219]
(c) Microsoft Corporation. All rights reserved.
C:\Users\tejar>mongosh "mongodb+srv://cluster0.grkiu2w.mongodb.net/myFirstDatabase" --apiVersion 1 --username teja
Enter password:
Current Mongosh Log ID: 639bfc72479b24c104d1cc8e
Connecting to: mongodb+srv://<credentials>@cluster0.grkiu2w.mongodb.net/myFirstDatabase?appName=mongosh+1.6.1
MongoInvalidArgumentError: Password cannot be empty
C:\Users\tejar>mongosh "mongodb+srv://cluster0.grkiu2w.mongodb.net/Health_Information" --apiVersion 1 --username teja
Enter password: *******
Current Mongosh Log ID: 639bfc961880c08886d64dcb
Connecting to: mongodb+srv://<credentials>@cluster0.grkiu2w.mongodb.net/Health_Information?appName=mongosh+1.6.1
Using MongoDB: 5.0.14 (API Version 1)
Using Mongosh: 1.6.1
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
Atlas atlas-zzm6u2-shard-0 [primary] Health_Information> db.Data_Requirements.find()
Atlas atlas-zzm6u2-shard-0 [primary] Health_Information> exit
C:\Users\tejar>mongosh "mongodb+srv://cluster0.grkiu2w.mongodb.net/Hospital_Information" --apiVersion 1 --username teja
Enter password: *******
Current Mongosh Log ID: 639bfd1991b0665b7f46c64b
Connecting to: mongodb+srv://<credentials>@cluster0.grkiu2w.mongodb.net/Hospital_Information?appName=mongosh+1.6.1
Using MongoDB: 5.0.14 (API Version 1)
Using Mongosh: 1.6.1
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
Atlas atlas-zzm6u2-shard-0 [primary] Hospital_Information> db.Data_Requirements.find()
[
{
_id: ObjectId("639bf390a89010c665d0bad1"),
Patient_Name: 'Ankith',
Date_of_Birth: ISODate("2000-08-23T04:00:00.000Z"),
Medical_History: 'No',
Insurance_Information: 'Yes'
},
39°F
Partly cloudy
Search
12:53 AM
12/16/2022


mongosh mongodb+srv://<credentials>@cluster0.grkiu2w.mongodb.net/Hospital_Information
Atlas atlas-zzm6u2-shard-0 [primary] Hospital_Information> db.Surgery_Details.find()
[
{
_id: ObjectId("639bf573a89010c665d0bad8"),
Patient_Name: 'Ankith',
Surgery_Type: 'Spinal',
Date_of_surgery: ISODate("2001-03-03T05:00:00.000Z"),
Result: 'Success'
},
{
_id: ObjectId("639bf619a89010c665d0bad9"),
Patient_Name: 'Chaitanya',
Surgery_Type: 'Knee',
Date_of_surgery: ISODate("2021-03-13T05:00:00.000Z"),
Result: 'Fail'
},
{
_id: ObjectId("639bf643a89010c665d0bada"),
Patient_Name: 'Jaswanth',
Surgery_Type: 'laparoscopic',
Date_of_surgery: ISODate("2020-03-23T05:00:00.000Z"),
Result: 'Fail'
},
{
_id: ObjectId("639bf675a89010c665d0badb"),
Patient_Name: 'Kavya',
Surgery_Type: 'laparoscopic',
Date_of_surgery: ISODate("2020-08-23T05:00:00.000Z"),
Result: 'Success'
}
]
Atlas atlas-zzm6u2-shard-0 [primary] Hospital_Information> db.Surgery_Details.insert({"Patient_Name":'Manasa',"Surgery_Type":'Knee',"Date_of_surgery":ISODate("2020-08-23","
Result":'Fail'})
Uncaught:
SyntaxError: Unexpected token, expected "," (1:120)
> 1 | db.Surgery_Details.insert({"Patient_Name":'Manasa',"Surgery_Type":'Knee',"Date_of_surgery":ISODate("2020-08-23","Result":'Fail'})
2 |
39°F
Partly cloudy
Search
1:06 AM
12/16/2022





Q2.

a discussion describing the form that the NoSQL DB should take (including choosing a specific DB). You should consider other options in your discussion.

Answer:

There are many alternatives to pick from when deciding what shape, a NoSQL database should take. One well-liked option is MongoDB, a document-oriented database that stores data in objects that resemble JSON. Because MongoDB is so highly scalable, it is perfect for applications with large datasets. Furthermore, it offers strong search and aggregation capabilities. But there are more options besides MongoDB. Another popular option is Cassandra, a distributed database made to handle enormous amounts of data while maintaining high availability. Applications that need quick queries and high throughput should use Cassandra. Redis comes in last. For storing data in a key-value format, Redis is an in-memory data structure store. Applications that need high-performance and distributed caching should use Redis. The final shape a NoSQL database should take will rely on the requirements of the application. Cassandra or MongoDB can be the better option if the application needs to scale. Redis might be the best choice if performance is a top concern.

Q3.

a discussion of why your non-relational solution would work best - a key grading point is justifying your choice (business issues such as cost may also contribute to your decision in addition to technical requirements). Describe the specific characteristics of your scenario that suggests your NoSQL solution is the best choice. Describing why it can be utilized is important but insufficient; convince me that it should be the solution - not just that it can be the solution.

Answer.

Since it offers a cost-effective, effective, and scalable solution for my data storage needs, my non-relational solution would be the most effective. I can store my data in a form that is simple to access, edit, and query by utilizing a NoSQL database. The data is simply scalable and reliably distributed across several servers. I don't have to worry about the hassle of maintaining a database schema or the requirement to convert my data into a schema-based format because NoSQL databases are schema-less. NoSQL databases are excellent for managing enormous amounts of data. The database may be scaled to meet application requirements with little effort and expense because the data is spread across numerous servers. Massive datasets can be stored in NoSQL databases since they are optimized for handling large amounts of data. Finally, NoSQL databases are excellent for applications that need high availability and minimal latency. For instance, a NoSQL database is the best choice if my application must reply to requests quickly or must be accessible always. NoSQL databases are a great option for applications that need to handle huge volumes of traffic because they can accommodate numerous concurrent requests. In conclusion, my non-relational solution offers a cost-effective, effective, and scalable solution, making it the ideal choice for my data storage needs. It is also

appropriate for applications that need high availability, low latency, and the ability to handle massive amounts of data. A NoSQL database is the greatest option for my circumstance because of all these advantages.

Q4.

Implement a barebones prototype. Provide a set of example records to show the type of data that your solution is designed to support. Try to give example data for as many realistic types of records as you can.

Answer:

My basic model is an online database that keeps track of student grades. The example records below demonstrate the kinds of data that my solution is intended to support:

-Student Name: John Smith

-Course: English Literature

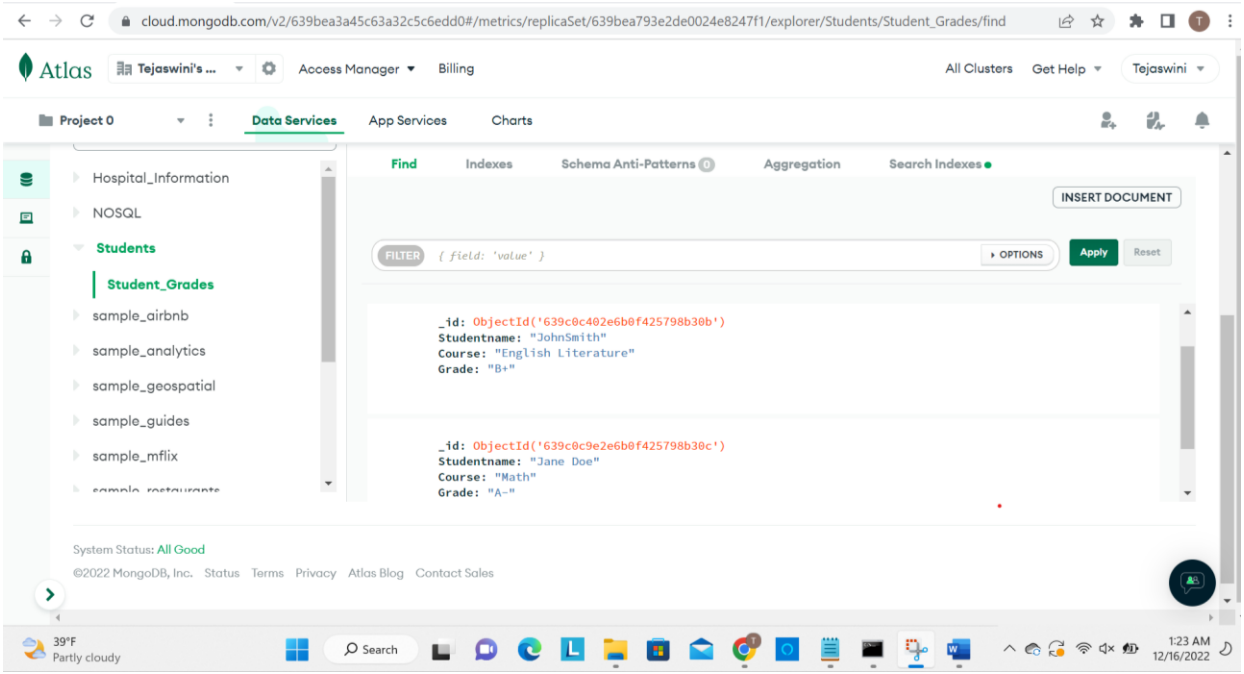
-Grade: B+

-Student Name: Jane Doe

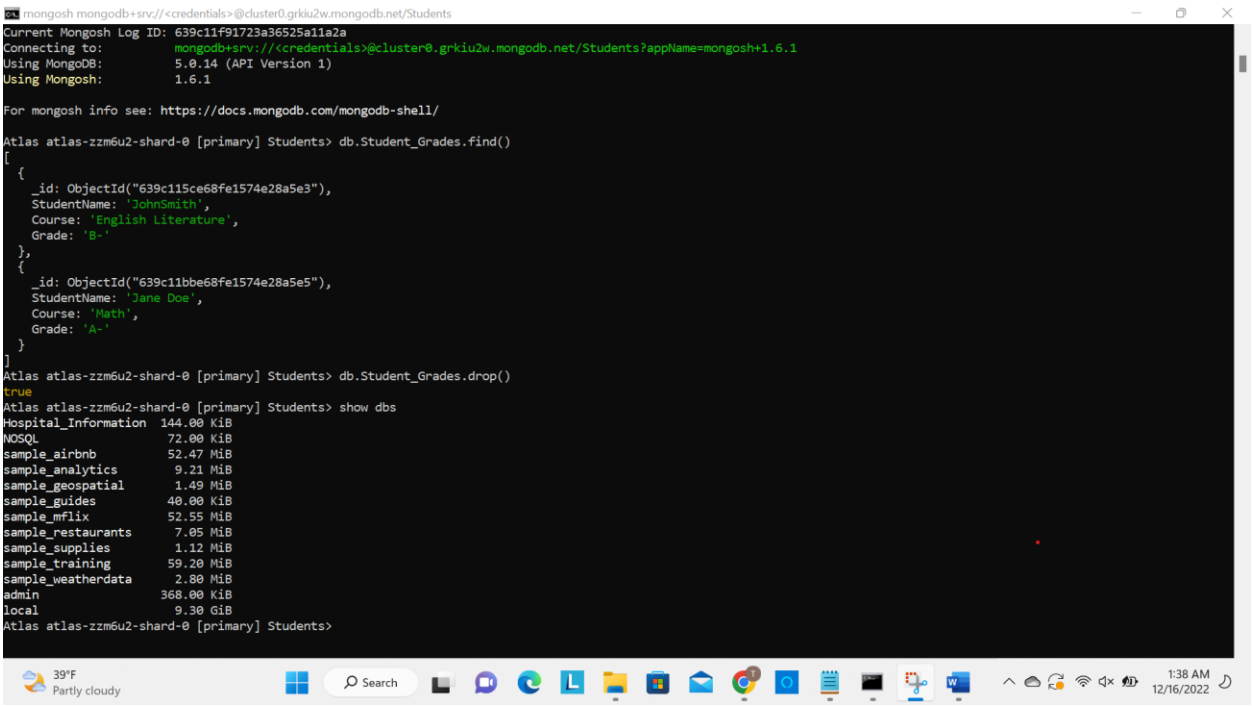
-Course: Math

-Grade: A-

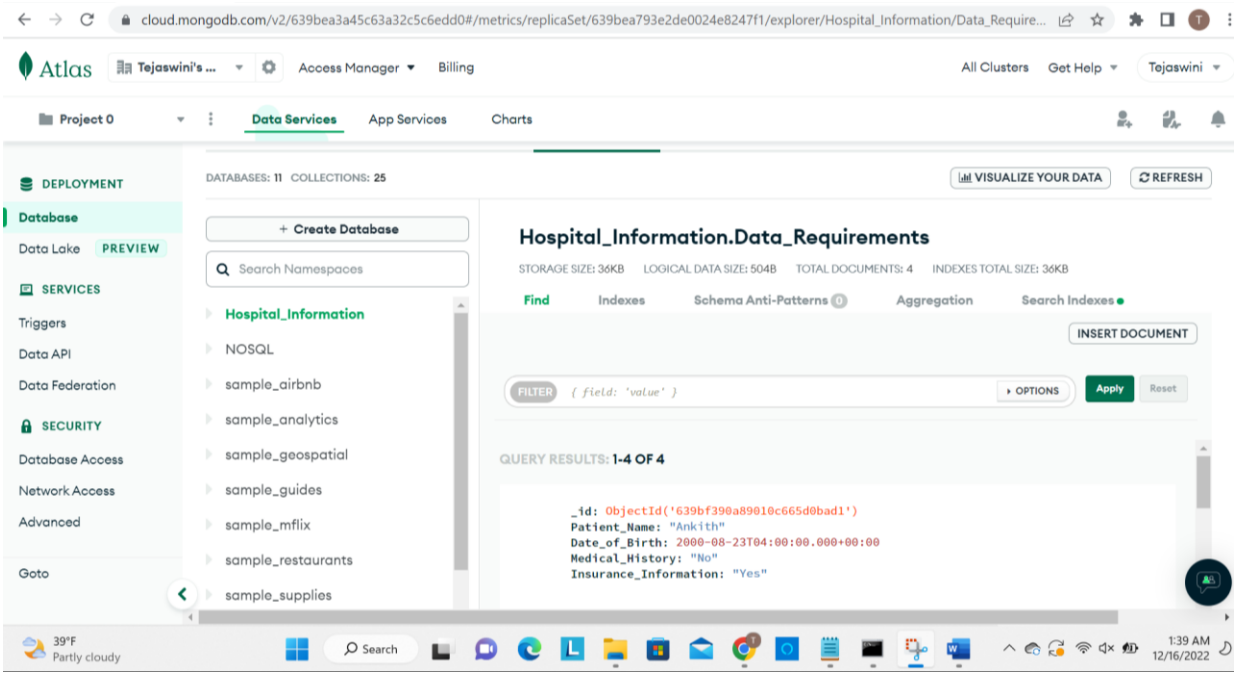
Below are the screenshots of the collections created in the MongoDB with the student details with their name, course, and their grades.



Using commands like find (), dop(), and show DBS would help us to retrieve the information as shown in the below screenshot.



After using the above commands we can see that the collection “Students” is dropped from the whole database from the screenshot below.



Q5

implement your solution and submit screenshots demonstrating its functionality. You should tie this discussion back to the originally hypothesized requirements. To be clear, you are not responsible for creating an application that uses this data. Requirements inform you of what an application would need to be capable of and you should describe how the non-implemented application would use your designed data.

Answer:

The database's main function is to keep track of user information for an online store and the available goods. It is intended to contain both product data, like item availability and pricing, as well as customer data, including contact details, payment options, and transaction history.

The database is designed with the following tables:

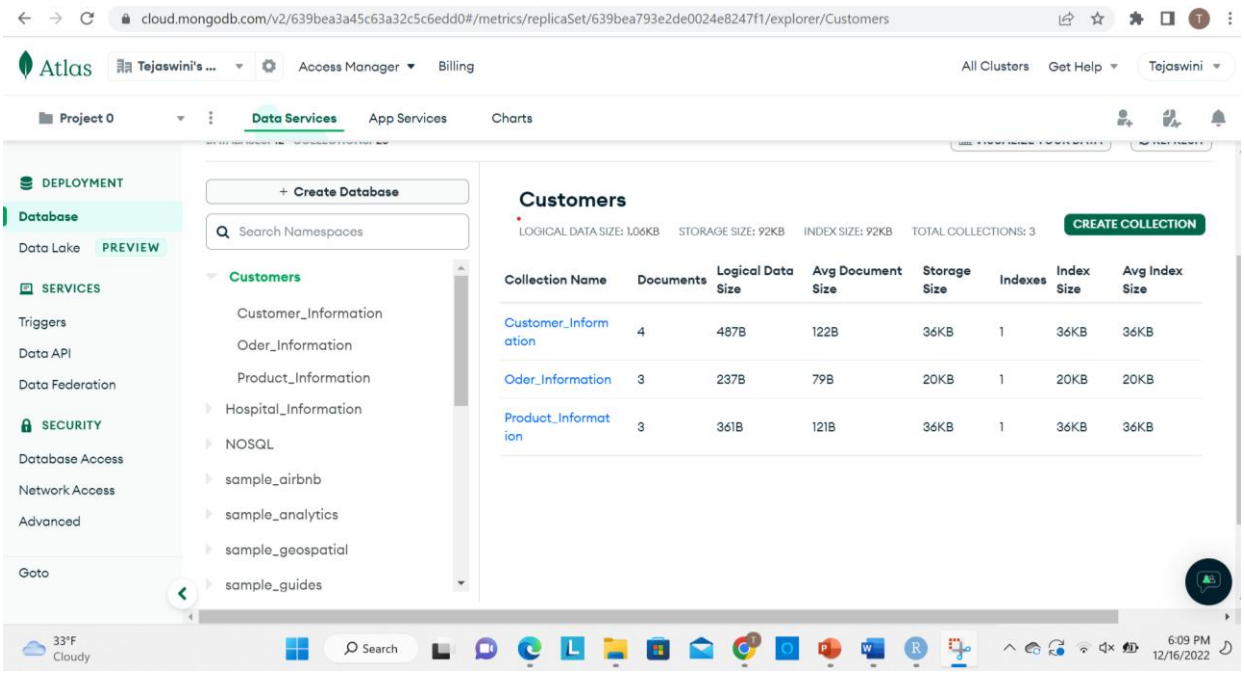
Customers: Information about a customer's contact details, payments made, and past orders are kept in this table. Customers' names, emails, phone numbers, addresses, payment methods, and order histories are all listed in the following columns.

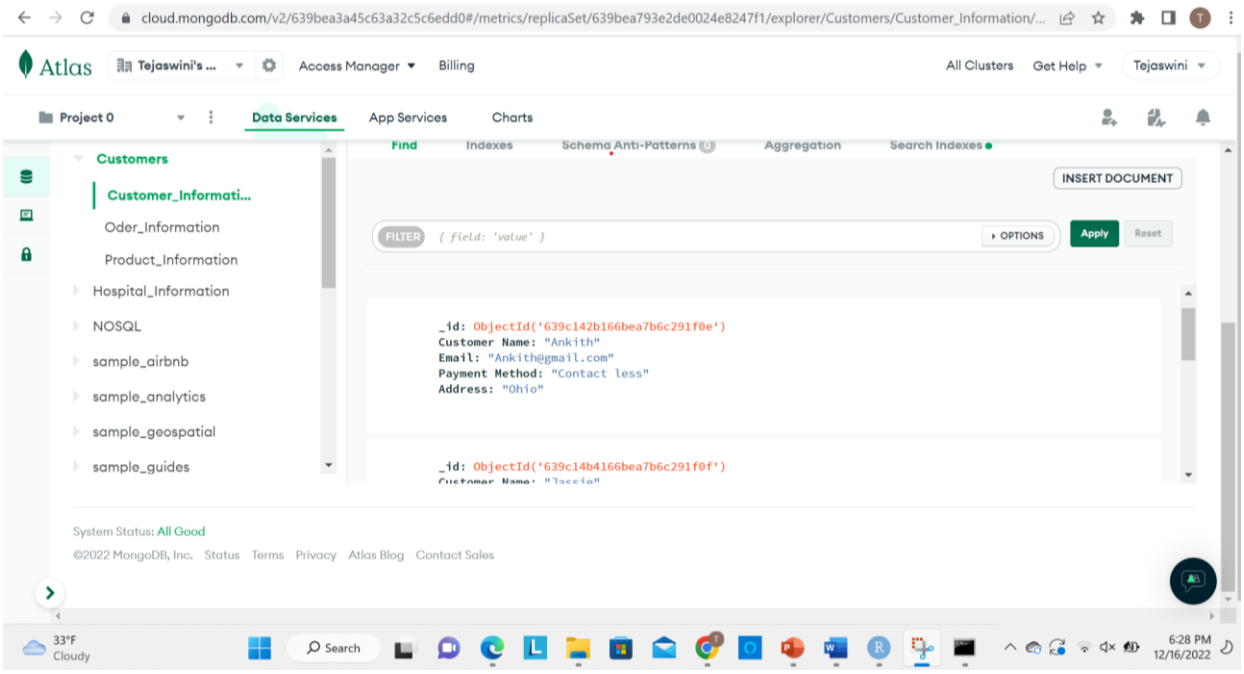
Products: Information about the products, including their availability and costs, is kept in this table. It has the following columns: product ID, product name, product description, product availability, and product price.

Orders: Information regarding customer orders are kept in this table. Order ID, customer ID, product ID, quantity, and order date are the columns that make up this list.

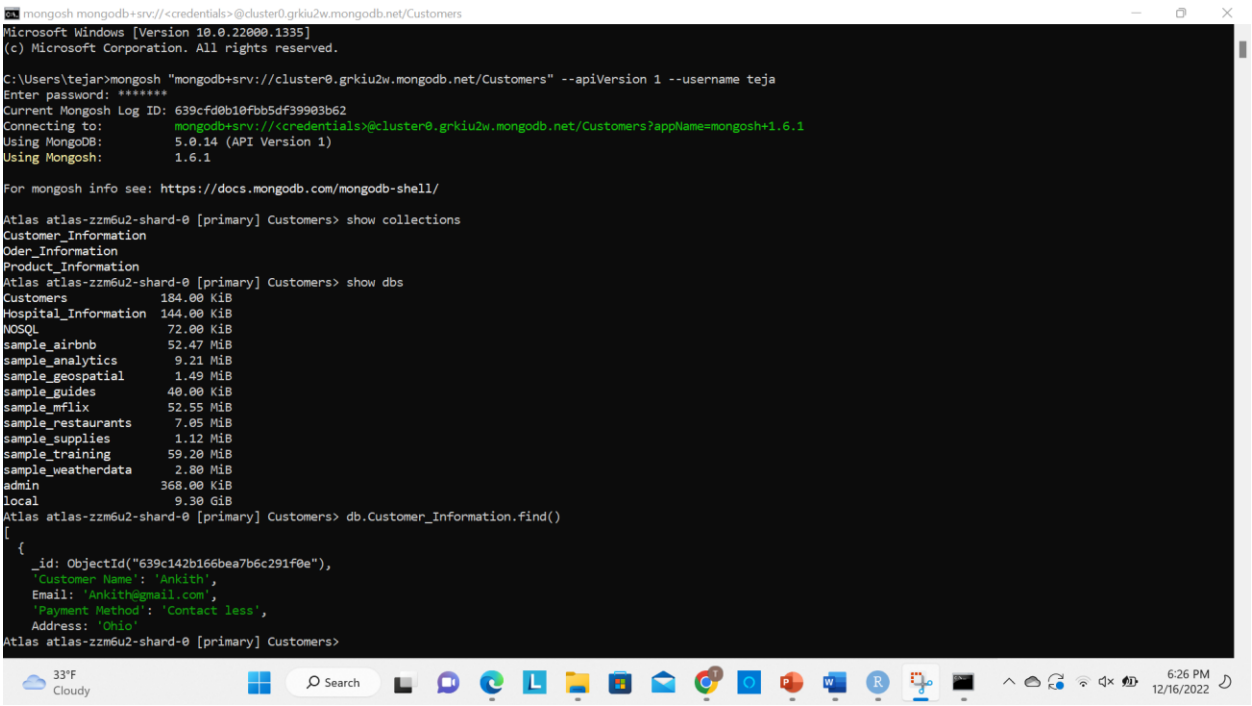
Steps to create the above database:

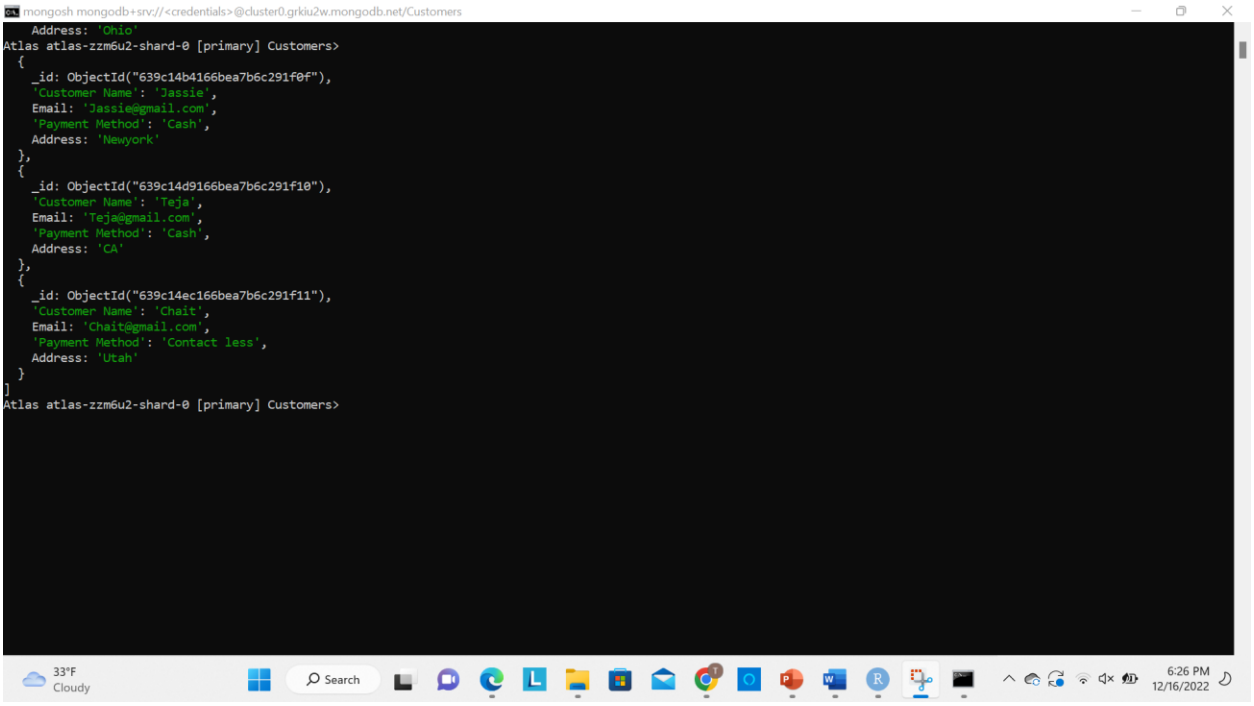
1. Created a cluster named “Customers” that includes the Customer Information, Product Information, and order information as collections.
2. Customer Information collection or table includes Customer Name, Email, Payment Method, and Address.
3. Product Information collection or table includes Product ID, Product Name, Product Description, and Product Price.
4. Order Information collection or table includes Order ID, Product ID, and order Date.

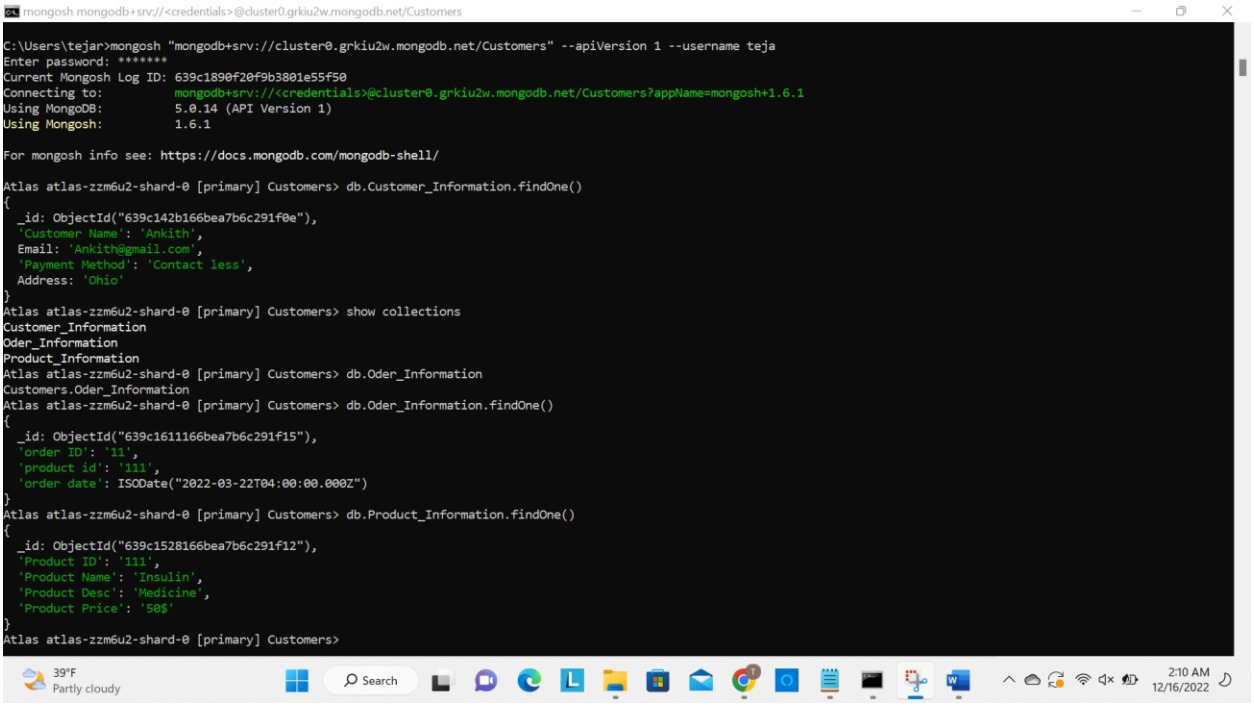




The commands used to retrieve the data are **find**, **findOne**, **show dbs,** and **show collections**.







Both client and product data can be stored and retrieved from the database. Customers and their orders can be connected using the customer ID from the Customers table or collection. The product ID in the Products table can also be used to connect products with the orders they are associated with in the Orders table.

Customers' contact details, payment methods, and order histories can all be tracked in the database. Tracking product information, including availability and cost, is another use for it. Orders can be tracked using the database, which stores information on the customer who placed the order as well as the product and amount they requested.

Note: Ideas and basic MongoDB commands are sourced from the internet along with the basic functionality. A few other samples are also looked up in the MongoDB Atlas platform to view the functionality and implementation.