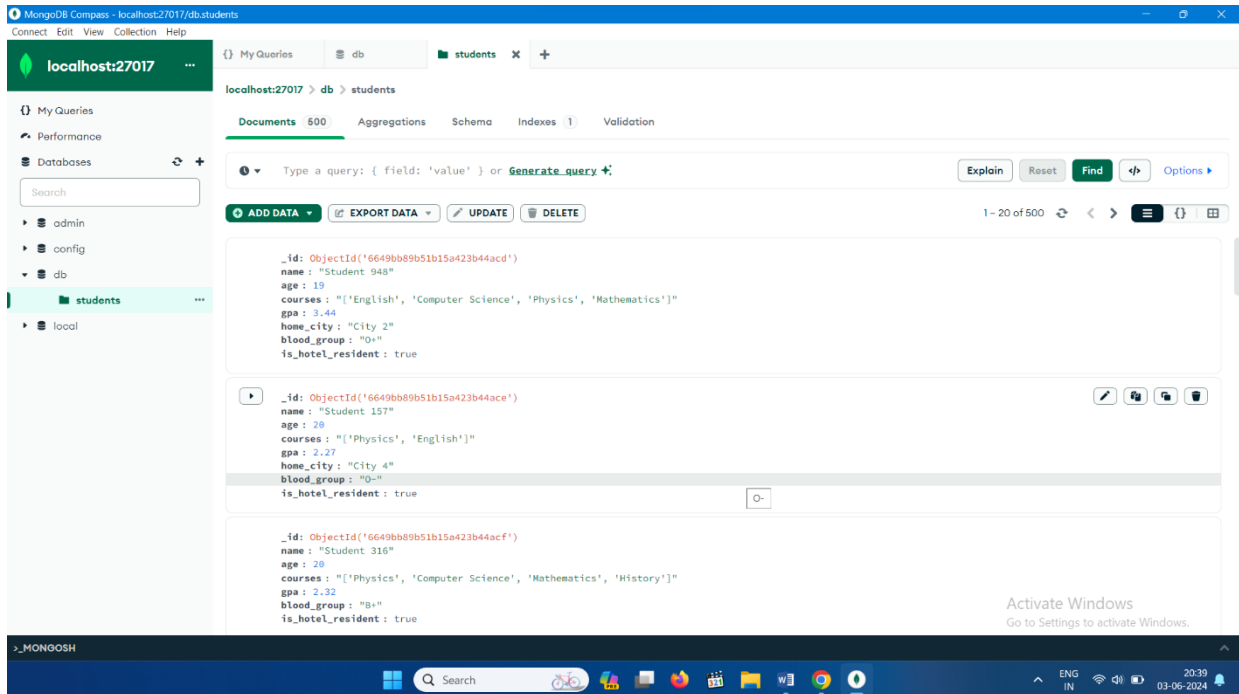


# MONGODB

## MONGODB COMPASS EXPLORATION:



The above image is a screenshot of a MongoDB Compass which is a graphical user interface for interacting with MongoDB databases. In order to perform operations in MongoDB you need to import data which is in csv form to MongoDB compass. Here student.csv file is imported into MongoDB compass . The screenshot shows a database named "students" which contains information about different students. Each student has fields such as name, age, courses, gpa, home city, blood group and is\_hotel\_resident.

In the above image, you can see three students with their details. For instance, the first student's name is "Student 544", age is 15, gpa is 3.44 and courses include English, Computer Science, Physics and Mathematics.

MongoDB Compass is a free, open-source graphical user interface for interacting with MongoDB databases. It is available for Windows, macOS, and Linux. MongoDB Compass can be used to browse, query, and edit data in MongoDB databases. It also provides a number of features for managing MongoDB deployments, such as creating and managing databases, users, and roles.

Each Row can be associated with Documents. Documents can be represented as JSON. Each table can be called as Collection.

## DOCUMENTS:

A document in MongoDB is a record in a collection, akin to a row in a relational database, but it is stored in a flexible, JSON-like format called BSON (Binary JSON). This flexibility allows for the storage of nested structures and varying data types without a predefined schema. Using Compass, users can easily visualize document structures, modify field values, add new fields, and run queries to filter documents, all without needing to write complex code. This makes MongoDB Compass a powerful tool for both developers and database administrators, enabling efficient management of data and simplification of database operations.

```
{ "greeting" : "Hello, world!" }
```

## COLLECTIONS:

Collections in MongoDB Compass are analogous to tables in a relational database but offer much greater flexibility due to MongoDB's schema-less nature. A collection is a grouping of MongoDB documents, where each document can have a different structure, making it easy to store diverse data within the same collection. In MongoDB Compass, collections are visually represented, allowing users to explore and manage their data seamlessly. Users can create, delete, and rename collections, as well as view detailed statistics about the documents they contain. Compass provides tools to index collections, ensuring efficient query performance, and offers various options to filter, sort, and aggregate data within collections. This user-friendly interface empowers users to perform complex database operations and gain insights into their data without needing extensive MongoDB query language knowledge.

## DATABASE:

In MongoDB Compass, a database serves as a container for collections, providing an organized structure for managing related sets of data. Databases in MongoDB can hold multiple

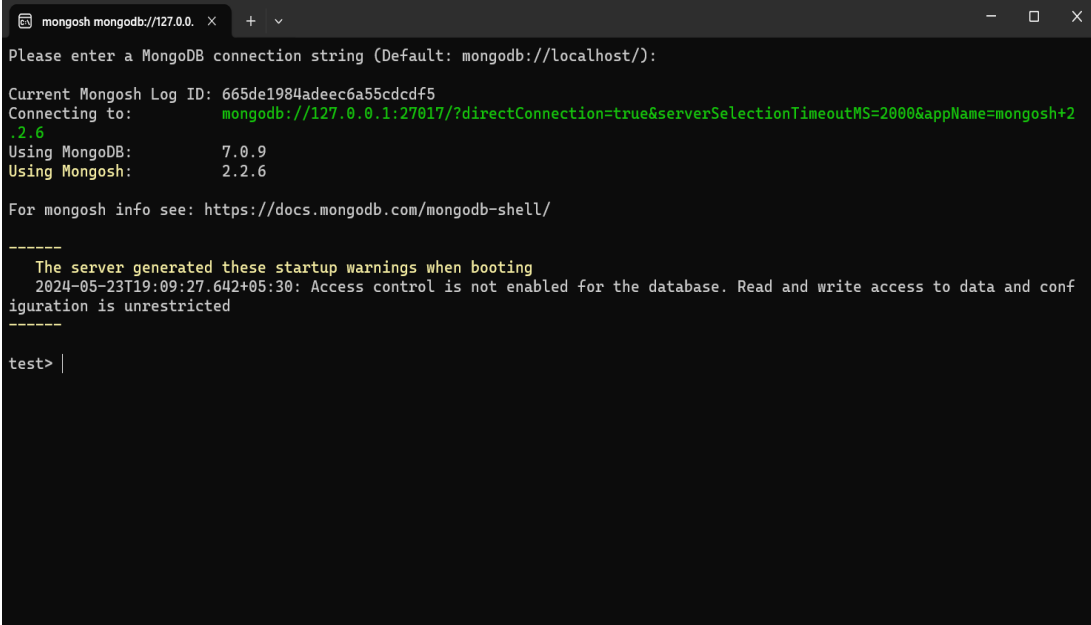
collections, each with its own set of documents, and they enable efficient data segmentation and access control. With Compass, users can easily create, rename, and delete databases, as well as explore their contents through a graphical interface. The intuitive design of Compass allows users to drill down from the database level to individual collections and documents, providing a comprehensive view of the data hierarchy. Additionally, Compass offers functionalities such as monitoring database performance, managing indexes, and executing queries, making it a powerful tool for database administration and data analysis. This seamless interaction with databases in MongoDB Compass enhances productivity and simplifies the management of complex data structures.

## DATATYPE:

In MongoDB, data types are crucial for defining the nature of data stored in documents. MongoDB supports a wide array of data types, reflecting its versatility and flexibility in handling diverse data structures. Common data types include String, Number (int, long, double, decimal), Boolean, Date, Array, Object (embedded document), Null, and Binary Data. Additionally, MongoDB supports unique data types such as ObjectId, which is a 12-byte identifier used as a primary key, and Min/Max Key, which represent the lowest and highest BSON values respectively. This variety allows for the storage of complex and hierarchical data within a single document. Each data type is efficiently encoded in BSON (Binary JSON) format, ensuring optimal performance and storage. Understanding these data types is essential for designing effective schemas and optimizing queries in MongoDB, allowing for a more efficient and scalable database system.

```
{
  "name" : "John Doe",
  "address" : {
    "street" : "123 Park Street",
    "city" : "Anytown",
    "state" : "NY"
  }
}
```

## MONGODB SHELL EXPLORATION:



```
mongosh mongodb://127.0.0.1:27017
Please enter a MongoDB connection string (Default: mongodb://localhost/):

Current Mongosh Log ID: 665de1984adeec6a55cdcdf5
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.6
Using MongoDB:      7.0.9
Using Mongosh:      2.2.6

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

-----
The server generated these startup warnings when booting
2024-05-23T19:09:27.642+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

test> |
```

MongoDB Shell, often referred to as "mongosh," is a command-line tool used to interact with MongoDB databases. It allows users to perform various database operations using JavaScript commands. With MongoDB Shell, you can connect to your MongoDB server, create and modify databases and collections, insert, update, and delete documents, and run queries to retrieve data. It's a powerful tool for developers and administrators because it provides direct access to the database, enabling detailed data manipulation and management. The Shell is particularly useful for scripting and automating database tasks, making it an essential tool for efficiently managing MongoDB databases.

## Few Commands to test after connections

Command	Expected Output	Notes
show dbs	admin 40.00 KiB config 72.00 KiB db 128.00 KiB local 40.00 KiB	All Databases are shown
use db	switched to db db	Connect and use db
show collections	Students	Show all tables
db.foo.insert({"bar" : "baz"})		Insert a record to collection. Create Collection if not exists

## Few Commands to test after connections

Command	Notes
db.foo.batchInsert([{"_id" : 0}, {"_id" : 1}, {"_id" : 2}])	Insert more than one document
db.foo.find()	Print all rows
db.foo.remove()	Remove foo table

### ADD,UPDATE:

### INSERT:

In MongoDB, the insert functions are used to add new documents to a collection. These functions allow you to store data within your MongoDB database.

```

test> show dbs
admin   40.00 KiB
config 72.00 KiB
db      56.00 KiB
local   72.00 KiB
test> use db
switched to db db
db> show collections
students
db> db.foo.insert({"bar" : "baz"})
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('6661ba8902b7ff5ee7cdcdf6') }
}

```

In the provided MongoDB shell session, the user performs several operations. Initially, the `show dbs` command lists all available databases on the server, revealing four databases: `admin`, `config`, `db`, and `local`. The user then switches to the `db` database using the `use db` command, which is confirmed by the message `switched to db db`. Next, the `show collections` command lists the collections within the `db` database, showing a single collection named `students`. The user attempts to insert a document `{"bar": "baz"}` into a new collection named `foo` using the `db.foo.insert()` method. Although this method is deprecated, as indicated by the warning to use `insertOne`, `insertMany`, or `bulkWrite` instead, the insertion is successful. The response confirms that the operation was acknowledged and provides the inserted document's ID. For future compatibility, it is recommended to use `insertOne` for single document insertions to avoid deprecation issues.

## FIND:

In MongoDB, the `find` method is used to query a collection and retrieve documents that match specified criteria. It returns a cursor to the matching documents,

allowing for efficient retrieval and iteration through the results. By default, `find` returns all documents in a collection if no criteria are provided, and can be combined with various query operators to

```
TypeError: db.foo.find() is not a function
db> db.foo.find()
[ { _id: ObjectId('6661ba8902b7ff5ee7cdcdf6'), bar: 'baz' } ]
```

The command `db.foo.find()` is executed in the MongoDB shell to query all documents within the `foo` collection. The output shows a single document, `{ _id: ObjectId('6661ba8902b7ff5ee7cdcdf6'), bar: 'baz' }`, indicating that the `foo` collection contains one document. This document has an automatically generated unique identifier, `_id: ObjectId('6661ba8902b7ff5ee7cdcdf6')`, which serves as its primary key, ensuring each document in the collection can be uniquely identified. The document also contains a field `bar` with the value `'baz'`. The `find` method returns an array of matching documents, and in this case, it displays the only document present in the `foo` collection.