MS: UNIT- 5: MongoDB

Prepared By,
M.Gouthamm,Asst.Prof, CSE, MRUH

Understanding MongoDB

- At the core of most large-scale web applications and services is a high-performance data storage solution. The backend data store is responsible for storing everything from user account information to shopping cart items to blog and comment data.

- Good web applications must store and retrieve data with accuracy, speed, and reliability. Therefore, the data storage mechanism you choose must perform at a level that satisfies user demand. Several different data storage solutions are available to store and retrieve data needed by your web applications. The three most common are direct file system storage in

1. **Files** – .txt, .csv, .xml, .json, etc.,
2. **Relational databases** – MySQL, MS SQL, Postgres, Oracle, etc.,
3. **NoSQL databases** – MongoDB, Cassandra (Facebook), Redis, Apache HBase (Google), Neo4J, Elasticsearch,Amazon DynamoDB

MongoDB

Why NoSQL?

- The concept of NoSQL (Not Only SQL) consists of technologies that provide storage and retrieval without the tightly constrained models of traditional SQL relational databases.

The motivation behind NoSQL is mainly

- simplified designs, horizontal scaling, and finer control of the availability of data.
- **MongoDB** is an open source NoSQL database management program. NoSQL (Not only SQL) is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. **MongoDB is a tool that can manage document-oriented information, store or retrieve information.**
- MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.
- Structured Query Language (SQL) is a standardized programming language that is used to manage relational databases. SQL normalizes data as schemas and tables, and every table has a fixed structure.

MongoDB

- Instead of using tables and rows as in relational databases, as a NoSQL database, the MongoDB architecture is made up of collections and documents.
- Documents are made up of key-value pairs -- MongoDB's basic unit of data. Collections, the equivalent of SQL tables, contain document sets.
- MongoDB offers support for many programming languages, such as C, C++, C#, Go, Java, Python, Ruby and Swift.

MongoDB Work?

How does MongoDB work?

- MongoDB environments provide users with a server to create databases with MongoDB. MongoDB stores data as records that are made up of **collections and documents.**
- Documents contain the data the user wants to store in the MongoDB database. Documents are composed of field and value pairs. They are the basic unit of data in MongoDB.
- The documents are similar to **JavaScript Object Notation (JSON)** but use a variant called **Binary JSON (BSON).** The benefit of using BSON is that it accommodates more data types. The fields in these documents are like the columns in a relational database.
- Values contained can be a variety of data types, including other documents, arrays and arrays of documents, according to the MongoDB user manual. Documents will also incorporate a primary key as a unique identifier. A document's structure is changed by adding or deleting new or existing fields.

MongoDB Work?

Understanding Collections

- MongoDB groups data together through collections. A collection is simply a grouping of documents that have the same or a similar purpose. A collection acts similarly to a table in a traditional SQL database, with one major difference i.e., In MongoDB, a collection is not enforced by a strict schema. This reduces the need to break items in a document into several different tables, which is often done in SQL implementations.

Understanding Documents

- A document is a representation of a single entity of data in the MongoDB database. A collection is made up of one or more related objects. A major difference between MongoDB and SQL is that documents are different from rows. Row data is flat, meaning there is one column for each value in the row. However, in MongoDB, documents can contain embedded sub-documents, thus providing a much closer inherent data model to your applications.

In fact, the records in MongoDB that represent documents are stored as BSON, which is a lightweight binary form of JSON, with field:value pairs corresponding to JavaScript property:value pairs. These field:value pairs define the values stored in the document. That means little translation is necessary to convert MongoDB records back into the JavaScript object that you use in your Node.js applications.

MongoDB Work?

Sample Document:

```
{
name: "New Project",
version: 1,
languages: ["JavaScript", "HTML", "CSS"],
admin: {name: "Brad", password: "****"}
}
```

Notice that the document structure contains fields/properties that are strings, integers, arrays, and objects, just like a JavaScript object.

MongoDB Work?

Rules:

- The field names cannot contain null characters, . (dots), or $ (dollar signs).
- Also, the _id field name is reserved for the Object ID.

- The _id field is a unique ID for the system that is made up of the following parts:

1. A 4-byte value representing the seconds since the last epoch
2. A 3-byte machine identifier
3. A 2-byte process ID
4. A 3-byte counter, starting with a random value

- The maximum size of a document in MongoDB is 16MB, which prevents queries that result in an excessive amount of RAM being used or intensive hits to the file system. Although you may never come close, you still need to keep the maximum document size in mind when designing some complex data types that contain file data.

Why MongoDB used?

An organization might want to use MongoDB for the following:

1. **Storage:** MongoDB can store large structured and unstructured data volumes and is scalable vertically and horizontally. Indexes are used to improve search performance. Searches are also done by field, range and expression queries.

2. **Data integration:** This integrates data for applications, including for hybrid and multi-cloud applications.

3. **Complex data structures descriptions:** Document databases enable the embedding of documents to describe nested structures (a structure within a structure) and can tolerate variations in data.

4. **Load balancing:** MongoDB can be used to run over multiple servers.

Features of MongoDB

Features of MongoDB include the following:

1. **Replication:** A replica set is two or more MongoDB instances used to provide high availability. Replica sets are made of primary and secondary servers. The primary MongoDB server performs all the read and write operations, while the secondary replica keeps a copy of the data. If a primary replica fails, the secondary replica is then used.
2. **Scalability:** MongoDB supports vertical and horizontal scaling. Vertical scaling works by adding more power to an existing machine, while horizontal scaling works by adding more machines to a user's resources.
3. **Load balancing:** MongoDB handles load balancing without the need for a separate, dedicated load balancer, through either vertical or horizontal scaling.
4. **Schema-less:** MongoDB is a schema-less database, which means the database can manage data without the need for a blueprint.
5. **Document:** Data in MongoDB is stored in documents with key-value pairs instead of rows and columns, which makes the data more flexible when compared to SQL databases.

Advantages of MongoDB

MongoDB offers several potential benefits:

1. **Schema-less:** Like other NoSQL databases, MongoDB doesn't require predefined schemas. It stores any type of data. This gives users the flexibility to create any number of fields in a document, making it easier to scale MongoDB databases compared to relational databases.

2. **Document-oriented:** One of the advantages of using documents is that these objects map to native data types in several programming languages., Having embedded documents also reduces the need for database joins, which can lower costs.

3. **Scalability:** A core function of MongoDB is its horizontal scalability, which makes it a useful database for companies running big data applications. In addition, sharding lets the database distribute data across a cluster of machines. MongoDB also supports the creation of zones of data based on a shard key.

Disdvantages of MongoDB

Though there are some valuable benefits to MongoDB, there are some downsides to it as well.

1. **Continuity:** With its automatic failover strategy, a user sets up just one master node in a MongoDB cluster. If the master fails, another node will automatically convert to the new master. This switch promises continuity, but it isn't instantaneous -- it can take up to a minute. By comparison, the Cassandra NoSQL database supports multiple master nodes. If one master goes down, another is standing by, creating a highly available database infrastructure.
2. **Write limits:** MongoDB's single master node also limits how fast data can be written to the database. Data writes must be recorded on the master, and writing new information to the database is limited by the capacity of that master node.
3. **Data consistency:** MongoDB doesn't provide full referential integrity through the use of foreign-key constraints, which could affect data consistency.
4. **Security:** In addition, user authentication isn't enabled by default in MongoDB databases. However, malicious hackers have targeted large numbers of unsecured MongoDB systems in attacks, which led to the addition of a default setting that blocks networked connections to databases if they haven't been configured by a database administrator.

MongoDB vs. RDBMS

- A relational database management system (RDBMS) is a collection of programs and capabilities that let IT teams and others create, update, administer and otherwise interact with a relational database. RDBMSes store data in the form of tables and rows. Although it is not necessary, RDBMS most commonly uses SQL.

- One of the main differences between MongoDB and RDBMS is that RDBMS is a relational database while MongoDB is nonrelational. Likewise, while most RDBMS systems use SQL to manage stored data, MongoDB uses BSON for data storage -- a type of NoSQL database.

- While RDBMS uses tables and rows, MongoDB uses documents and collections. In RDBMS a table -- the equivalent to a MongoDB collection -- stores data as columns and rows. Likewise, a row in RDBMS is the equivalent of a MongoDB document but stores data as structured data items in a table. A column denotes sets of data values, which is the equivalent to a field in MongoDB.

MongoDB vs. RDBMS

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

MongoDB vs. RDBMS

- The table below summarizes the main differences between SQL and NoSQL databases.

	SQL Databases	NoSQL Databases
Data Storage Model	Tables with fixed rows and columns	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
Development History	Developed in the 1970s with a focus on reducing data duplication	Developed in the late 2000s with a focus on scaling and allowing for rapid application change driven by agile and DevOps practices.
Examples	Oracle, MySQL, Microsoft SQL Server, and PostgreSQL	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune
Primary Purpose	General purpose	Document: general purpose, Key-value: large amounts of data with simple lookup queries, Wide-column: large amounts of data with predictable query patterns, Graph: analyzing and traversing relationships between connected data
Schemas	Rigid	Flexible
Scaling	Vertical (scale-up with a larger server)	Horizontal (scale-out across commodity servers)
Multi-Record ACID Transactions	Supported	Most do not support multi-record ACID transactions. However, some — like MongoDB — do.
Joins	Typically required	Typically not required
Data to Object Mapping	Requires ORM (object-relational mapping)	Many do not require ORMs. MongoDB documents map directly to data structures in most popular programming languages.

MongoDB platforms

- MongoDB is available in community and commercial versions through vendor MongoDB Inc. MongoDB Community Edition is the open source release, while MongoDB Enterprise Server brings added security features, an in-memory storage engine, administration and authentication features, and monitoring capabilities through Ops Manager.

- A graphical user interface (GUI) named MongoDB Compass gives users a way to work with document structure, conduct queries, index data and more. The MongoDB Connector for BI lets users connect the NoSQL database to their business intelligence tools to visualize data and create reports using SQL queries.

- Following in the footsteps of other NoSQL database providers, MongoDB Inc. launched a cloud database as a service named MongoDB Atlas in 2016. Atlas runs on AWS, Microsoft Azure and Google Cloud Platform. Later, MongoDB released a platform named Stitch for application development on MongoDB Atlas, with plans to extend it to on-premises databases.

MongoDB Data Types

MongoDB Data Types

- The BSON data format provides several different types that are used when storing the JavaScript objects to binary form. These types match the JavaScript type as closely as possible. It is important to understand these types because you can actually query MongoDB to find objects that have a specific property that has a value of a certain type.

- For example, you can look for documents in a database whose timestamp value is a String object or query for ones whose timestamp is a Date object. MongoDB assigns each of the data types an integer ID number from 1 to 255 that is used when querying by type.

MongoDB Data Types

MongoDB data types and corresponding ID number

Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Decimal126	19
Min key	-1
Max key	127

Planning your Data Model

- The key challenge in data modeling is balancing the needs of the application, the performance characteristics of the database engine, and the data retrieval patterns. When designing data models, always consider the application usage of the data (i.e. queries, updates, and processing of the data) as well as the inherent structure of the data itself.

Flexible Schema

- Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema. That is:

1. The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.
2. To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

Document Structure

- The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data. MongoDB allows related data to be embedded within a single document.

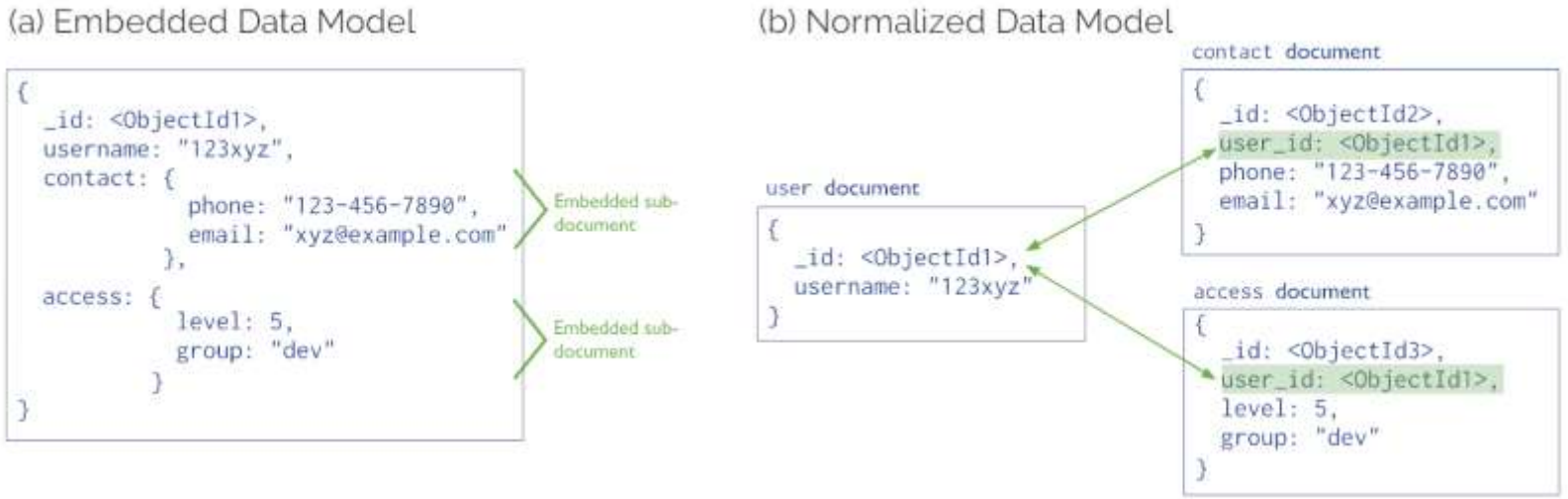
Planning your Data Model

Embedded & Normalized Data Models in MongoDB Data Modeling

- When data professionals start building data models in MongoDB, they fall upon the choice to either embed the information or to have it separately in a collection of documents.

Hence two concepts exist for efficient MongoDB Data Modeling:

1. The Embedded Data Model, and
2. The Normalized Data Model.



Planning your Data Model

Embedded Data Model

- Embedded data modeling in MongoDB Data Modeling — a denormalized data model — is applied when two data sets contain a relationship. Hence an embedded data model sets relationships between data elements, keeping documents in a single document structure. You can save information in an array or in a field, depending upon the requirements.

Normalized Data Model

- In a normalized data model, object references are used to model relationships between data elements/documents. This model reduces duplication of data; hence many-to-many relationships can be documented without the duplication of content fairly easily. Normalized data models are best for modeling large hierarchical datasets, referencing across collections.

Building the MongoDB Environment

Procedure:

- Download MongoDB Community Server (version 5.0.9) from https://www.mongodb.com/try/download/community
- Install **MongoDB** Complete Setup along with MongoDB Compass

  MongoDB Compass is a powerful GUI for querying, aggregating, and analyzing your MongoDB data in a visual environment.

  (https://downloads.mongodb.com/compass/mongodb-compass-1.32.4-win32-x64.exe)
- Working with MongoShell
  - Open Command Prompt window
  - Go to C:\Program Files\MongoDB\Server\5.0\bin
  - Type mongo

Building the MongoDB Environment

Set MongoDB in the windows path environment

Step 1: After successful installation, Right-click on **'This PC' or 'My Computer'.**

Choose properties

Step 2: Choose the `advance system setting` options

Step 3: Click on Environment Variables under Advance section.

Step 4: Choose Path value under system variables and click Edit button

Step 5: Now get your mongo path to your system, where your MongoDB is installed. For example, if you installed MongoDB in C drive, then it your path will be like this: `C:\Program Files\MongoDB\Server\VERSION\bin`

Step 6: Copy this path and enter as a new environment value on Edit environment variables page

Step 7: Now click on OK and close all active dialog box. Your environment is set, restart your terminal and now enter mongo , it will open mongo-shell.

Administering User Accounts

- In MongoDB, we are allowed to create new users for the database. Every MongoDB user only accesses the data that is required for their role.
- A role in MongoDB grants privileges to perform some set of operations on a given resource. In MongoDB, users are created using createUser() method.
- This method creates a new user for the database, if the specified user is already present in the database then this method will return an error.

Syntax:

db.createUser(user, writeConcern)

Administering User Accounts

Parameters:

user: It contains authentication and access information about the user to create. It is a document.

1. **user:** Name of the user
2. **pwd:** User password. This field is not required if you use this method on $external database to create a user whose credentials are stored externally. The value of this field can be of string type or passwordPrompt().
3. **customData:** User Associative Information. It is an optional field.
4. **roles:** Access Level or Privilege of a user. You can also create a user without roles by passing an empty array[]. In this field, you use built-in roles or you can create you own role using db.createRole(role, writeConcern) method.

To specify the roles you can use any of the following syntax:, Simply specify the role name: **"read"**

Or

you can specify a document that contains the role and db fields. It is generally used when the role is specified in a different database.

{role:<role>, db: <database>}

Administering User Accounts

How to create an administrative user?

- In MongoDB, you can create an administrative user using the createUser() method. In this method, we can create the name, password, and roles of an administrative user. Let us discuss this concept with the help of an example:

Example:

In this example, we are going to create an administrative user in the admin database and gives the user readWrite access to the config database which lets the user change certain settings for sharded clusters.

```
db.createUser(
{
 user: "hello_admin",
 pwd:  "hello123",
 roles:
 [ { role:"readWrite",db:"config"},
"clusterAdmin"
 ] } );
```

Administering User Accounts

How to create a normal user without any roles?

- In MongoDB, we can create a user without any roles by specifying an empty array[] in the role field in createUser() method.

Syntax:

```
db.createUser({ user:”User_Name”, pwd:”Your_Password”, roles:[]});
```

Administering User Accounts

How to create a normal user with some specifying roles?

- In MongoDB, we can create a user with some specified roles using the createUser() method. In this method, we can specify the roles that the user will do after creating.

Example:

- In this example, we are going to create a user with some specified roles.

```
db.createUser(
...{
...user: "new_one_role",
...pwd: with_roles",s
...roles:["readWrite", "dbAdmin"]
...}...);
```

Administering User Accounts

Here, we create a user whose name is "new_one_role", password is "with_roles" and the specified roles are:

1. **readWrite Role:** This role provides all the privileges of the read role plus the ability to modify data on all non-system collections.
2. **dbAdmin Role:** This role gives the ability to the user to perform administrative tasks such as schema-related tasks, indexing. It does not grant privileges for the User and Role Management.

MongoDB – Field Update Operators

- MongoDB provides different types of field update operators to update the values of the fields of the documents that matches the specified condition.

The following table contains the field update operators:

Operator	Description
$currentDate	This operator is used to set the value of a field to current date, either as a Date or a Timestamp.
$inc	This operator is used to increment the value of the field by the specified amount.
$min	This operator is used only to update the field if the specified value is less than the existing field value
$max	This operator is used only to update the field if the specified value is greater than the existing field value.
$mul	This operator is used to multiply the value of the field by the specified amount.
$rename	This operator is used to rename a field.
$setOnInsert	This operator is used to set the value of a field if an update results in an insert of a document. It has no effect on update operations that modify existing documents.

MongoDB – Field Update Operators

- $currentDate – Set Current Date to LoggedIn Field

```
db.emp.updateOne( { "name" : "ram"}, {$currentDate: {loggedIn:true}})
```

- $inc – to add 5000 to Salary as Annual Increment (5000 will be added to exist. Salary)

```
db.emp.updateOne({ "name":"ram"}, {$inc:{salary:5000}})
```

- $min – to set salary = 9000 if it is less than 500

```
db.emp.updateOne({ "name":"ram"}, {$min:{"salary" : 9000}})
```

- $max – to set salary = 10000 if it is less than 10000

```
db.emp.updateOne({ "name":"ram"}, {$max:{"salary" : 10000}})
```

- $mul – to add 20% of Salary as Increment i.e., 1.2 times x Salary

```
db.emp.updateOne( {"name" : "ram"}, { $mul: {salary : 1.2}})
```

- $rename – The rename operator changes the name of a field.

```
db.emp.updateOne({}, {$rename: { "name" : "Name"}})
```

Administering Databases

Creating a Database

- Before we create our own database on Mongo, let's have a look at the existing ones that were created through the installation by using the following command in the mongo shell:

```
show dbs
```

- As we can see, three databases named admin, config, and local were created by default. Now, let's create our own database to store the documents and collections.
- It should be interesting not that there is no explicit create command in the mongo shell to create a database. Instead, when we switch context through the **"use"** keyword, a new database will be created if it doesn't exist already:

```
use students
```

- Thus we have created a new database named students and have switched to it as well using the mongo shell.

Administering Databases

Create a Collection

- Let's create a collection named `Batch2021` to store our students' documents. We could do this by using the `createCollection` command:

```
db.createCollection("Batch2021")
```

We could list the collections in a database using the following query:

```
show collections
```

Administering Databases

MongoDB Insert Document

Inserting a Single Document:

- To insert a single document into our collection we could use the method `insertOne` on our newly created collection. It is also important to note that if we don't have a collection with a specified name mongo will create a new collection for us and insert the document into it. The collection names are case-sensitive as well.

Let's insert a document with the data of a student named "John" and his test scores as follows:

```
db.Batch2021.insertOne({name: "John", score1: 80, score2: 75})
```

Administering Databases

- our single document is inserted into the collection successfully. We could also query all the documents in the collection in the following way:

```
db.Batch2021.find({})
```

- We could see a strange field named _id that we did not add in our query to insert this document. This is a special field whose value should be unique for a document in a collection. In some ways, we could think of them as primary keys in a table from a relational database paradigm.

- If we don't pass a value (ObjectId) to this field, mongo will auto-generate this for every document as it did for us in our example. These ObjectIds have a specific length and a format that makes them valid.

Administering Databases

Inserting Multiple Documents

We could insert multiple documents into a collection in one go using the `insertMany` operation in the mongo shell.

Let's insert the data of two other students into the document with inserMany:

```
db.Batch2021.insertMany([{name:"Jane",score1:92,score2:87,score3:77},{name:"Adam"
,score1:99,score2:88}])
```

- By default, mongo performs an ordered insert on the documents inserted using insertMany operation. This means that the documents are inserted into the document in the specific order in which we supply them to the operation. If the order doesn't matter, we could also perform unordered insertions passing the ordered option as false.

```
db.Batch2021.insertMany([{name:"Jane",score1:92,score2:87,score3:77},{name:"Adam"
,score1:99,score2:88}], {ordered: false})
```

Administering Databases

1. Displaying a List of Databases
   - show dbs
2. Changing the Current Database
   - use mru
   - db – To check current Database

3. Creating Databases

- Your created database (mru) is not present in list. To display database, you need create collection and need to insert at least one document into it.
  - db.createCollection("student")
  - show collections – To see existing Collections

4. Deleting Databases
   - db.student.drop() – To drop collection
   - db.dropDatabase() – to delete Default Database

Administering Databases

- 5. Copying Databases
  - db.copyDatabase('mru','mru1')

- 6. Managing Collections
  a) Displaying a List of Collections in a Database
     - show collections
  b) Creating Collections
     - db.createCollection("student")
  c) Deleting Collections
     - db.student.drop() – To drop collection

- 7. Adding Documents to a Collection
  - db.student.insertOne({'name':'raju'}) – To add single document
  - db.student.insertMany([{"name":"raju"},{"name":"rao"}]) – To add Many documents

Administering Databases

- 8. Finding Documents in a Collection
  - db.student.find() – To view documents in student collection
  - db.student.findOne() – To view First Document in Student Collection
  - db.student.findOne({'name':'rao'}) – To view selected Record
- 9. Deleting Documents in a Collection
  - db.student.remove({'name':'rao'}) – To remove a Single Document
  - db.student.remove({}) – To remove a All Documents
- 10. Updating Documents in a Collection
  - db.student.update({'name':'raju'},{$set:{'name':'ram'}}) – To update first matched document
- 11. Quit the mongo shell
  - exit

End Of UNIT-5