Some term along with their analogues

* Row < -- > Document
* Column < -- > Field
* Table < -- > Collection

MongoDB Command

1. mongo 🡪 Starts MongoDB
2. show dbs 🡪 Show List of databases
3. use < db-name > 🡪 creates a new DB (if it does not exist) and start using DB.
4. db 🡪 shows active/currently used DB.
5. show collections 🡪 show all collections(tables) that exist in selected DB.
6. db.<collection-name>.insertOne({}) 🡪 insert single document(row) in collection(table). {} represents data to be inserted in json format.
7. db.<collection-name>.insertMany([{},{},{},......{}]) 🡪 Insert multiple documents(rows) in a collection(table).

Note: Id will be automatically created.

1. db.<collection-name>.find().pretty() 🡪 shows all the data in a collection.
2. db.<collection-name>.find({<key-value pair>}) 🡪 shows collection which have the provided key-value pair.
3. db.<collection-name>.find({<key-value pair>},{\_id:0}) 🡪 shows collection without ‘\_id’ field.
4. db.<collection-name>.find({<key-value pair>}, {anyKey: 1/0}) 🡪 {} after {<key-value pair>} represents projection, 0 means do not show field/column, 1 means show that field/column.
5. db.<collection-name>.find({<key-value pair>}).pretty().limit(2) 🡪show only first two documents/rows which have that key-value pair. This key-value pair is called **filter**.
6. db.<collection-name>.updateOne(filter, {$set:{<key-value pair that has to be updated>}}) 🡪 updates only one field. $set operator updates the value of a field.
7. db.<collection-name>.updateMany(filter,{$set:{key-value pair that has to be updated}}) 🡪 updates many field for matching filter.
8. db.<collection-name>.deleteMany(filter) 🡪 deletes the documents for which we have matching filter.
9. db.<collection-name>.deleteMany({}) 🡪 removes all documents in a collection.
10. quit() 🡪 Close MongoDB.

Mongoose 🡪 It is npm library that helps us to establish connection between node js and mongoDB. It acts as abstract layer between Node Js & MongoDB.

**Connecting using Mongoose**

const mongoose = require(‘mongoose’);  
mongoose.connect(“mongodb://localhost:27017/<database-name>”,{useNewUrlParser: true, useUnifiedTopology: true}); // It is connection creation  
Note: mongoose.connect() will return promise. If <database-name> is not present then it will be automatically created

**Mongoose Schema**

It defines structure of document (including their default values and the type of fields).  
const profileSchema = new mongoose.schema({  
 name: {type:String,required:true},  
 address: String,  
 dob: Date,  
 gender: {type: String, default: ‘Male’},  
 marksInPercent: Number   
});

**Mongoose Model**

It is an interface to the database for creating, querying, updating, deleting record, etc.  
const ProfileClass = new mongoose.model(“<collection-name>”, “<schema-name>”);  
By doing this we will be able to create Model class.

**Creating & Inserting a document using Mongoose**

const createDocument = async () => {  
 try{  
 const studentProfile = new ProfileClass({  
 name: “Shivam Khandelwal”,  
 address: “Shivaji Nagar, Pune”,  
 dob: 03/03/1995,  
 phoneNumber: 9876543210,  
 gender: “Male”,  
 marksInPercent: 90  
 }) ;  
 const result = await studentProfile.save();   
 } catch(error) {console.log(error);}  
}  
createDocument();

**Steps to insert data**

1. Create a connection
2. Create a Schema
3. From Schema create Model class.
4. Using Model class, create a document and insert it into DB.

**Inserting multiple Document using mongoose**ProfileClass.insertMany([profile1, profile2, profile3......]); // multiple documents in array

**Reading the document**Use any one of query

* Const profile = await ProfileClass.find(); **or**
* Const profile = await ProfileClass.find({gender: “Male”}); **or**
* Const profile = await ProfileClass.find({gender: “Male”}).select({name:1}); **or**
* Const profile = await ProfileClass.find({gender: “Male”}).select({name:1}).limit(10);

Const getDocument = async () => {  
 try {  
 const profile = await ProfileClass.find({gender: “Male”}) .select({name:1}).limit(10)  
 // we can use any one of the query mentioned above  
 } catch(error) {console.log(error)}  
}

**Query Comparison Operator**const profile = await ProfileClass.find({marksInPercent: {**$gt**:80}}).select({name:1}).limit(10);  
Other important query comparison operator are $eq, $gt, $gte, $in, $lt, $lte, $nin, etc.

**Query Logical Operator**$and, $or, $not, $or, etc.

**Sorting and count in Mongoose**For counting use .countDocuments() method.  
For sorting we use .sort({name: 1}) method. // will sort using sort.

**Updating document using mongoose**Use updateOne() & updateMany() to update document.  
There are multiple methods other than mentioned above such as findOne AndUpadte().

**Deleting document using Mongoose**Use deleteOne() or deleteMany() to delete a document.  
Other methods includes findByIdAndDelete(). This method also returns the document(s) that is/are deleted.  
Above mentioned methods return promise.