**Mongo DB**

**Database: -** Database is the place where we store our data in organized way to access quickly.

**SQL Database: -** SQL Database is a database in which data is stored in row and columns i.e. table. And relation between tables is established by primary and foreign key.

**NoSQL Database: -** NoSQL Database is a database in which data is not stored in table format. Data stored in different ways like graphs, key value pairs, documents etc.

**MongoDB:** - MongoDB is a document database in documents we have key value pairs. These documents are stored in collections and collections are stored in database. MongoDB is schema less. One document may have 3 columns other may have four. Documents need to similar but not identical.

**MongoDB is case sensitive.**

**Documents:** -Documents are in JSON format [Java Script Object Notation].

Documents are enclosed in curly brackets. Each Key value pairs are separated by commas, key and value are separated by colon [:].

Keys are enclosed in double codes where values like text enclosed in double codes.

We can have sub document means document in a document. And we can have array of value for a single key which are enclosed in square brackets and separated by commas.

For every document must have \_id column which used to uniquely identify the document if we won’t specify \_d then MongoDB assigns some value by default.

Same column in one document may have string and the same column in other document may have numerical .

MongoDB supports Numerical [int, float etc.], Date and Time and string datatypes [char, string].

**Document: -**

{

“\_id”:1203,

“Name”:” Pavan Kumar”,

“Age”:24

}

**Document with Subdocument: -**

{

“\_id”:1203,

“Name”:” Pavan Kumar”,

“Age”:24,

“Adress”:{

“D. No”: -“4-23”,

“Colony”:” ILTD Colony”,

“Pincode”:523157

}

}

**Document with array: -**

{

“\_id”:1203,

“Name”:” Pavan Kumar”,

“Age”:24,

“Ph. No”:[8074170625,9059706348,8096230625]

}

**Replica Sets:** - Replica sets are nothing but whole copy of data and are stored in physically isolated servers which provides redundancy and high availability. If one replica set fails data available from secondary replica sets.

**Advantages of MongoDB: -**

Flexible, more efficient because of schema less

Less complex

Easy to maintain comparatively.

**MongoDB Ecosystem: -** We can use MongoDB on cloud platforms like Azure, GCP and AWS as MongoDB Atlas. We can use MongoDB in Command Line and we can download Compass MongoDB which is an GUI [Graphical User Interface] to access and maintain MongoDB database.

**Connecting to Shell:** - Click on connect in atlas click on connect to shell copy string and paste in command line replace <username> with username and enter password.

**Connecting to Compass: -** Click on connect in atlas click on connect to compass copy string and paste in compass replace <username> and <password> with respective username and password of user.

**Creating and Deleting Database and Loading Sample data using Compass: -**

Go to atlas click on connect to compass and select I have already installed compass copy string and paste in compass and replace username and password then we connect to compass.

Now click on + symbol at database and give name to database and give name to collection.

Now we can see the database with collection now click on collection click on add data. click on import from csv or Jason select the file and select it Json or csv and load data. Now we can see the data.

When we click on database, we can see a delete symbol by click on it then it asks to enter database name to confirmation.

When we click on 3 dots at collection it shows option delete collection and enter name of collection to confirm.

**Note: -** When we click on explore collections, we can see same interface in atlas same as MongoDB Compass.

**Creating and deleting database and collections in Shell:**-

show dbs – to display all databases.

use [database name] to use database.

To create database use [database name] if database is present it moves to that database else it creates that database.

db.createCollection(‘Name of the collection’,Options) – to create a collection

db.[Name\_Of\_the\_Collection].drop() – to delete collection

db.dropDatabase() – to delete database to delete database we must in that database.

**Note:-** when we create database by use command then we won’t create any collection in it then it database won’t saved when we use shoe dbs. So to save database must create one collection that’s why in compass also when we create database it asks for collection so at least one collection has to create.

**Find and FindOne: -**

Find and Findone used to find the documents in the collection. Findone find one record in the document if multiple documents satisfy criteria, it returns first one. Find is same as findone but returns multiple documents. We can pass criteria as key value pair in curly brackets we can pass multiple criteria [and operation occurs] which are separated by commas and enclosed in same curly brackets.

db.[name of the collection].findOne({criteria})

db.[name of the collection].find({criteria})

**Note**: - cls command to clear screen

**Using find in compass and atlas: -**

Using find is compass and atlas are same. Click on collection from which we want to query then we have found ribbon below it we have selves to write our filter criteria. Write filter criteria in curly brackets if multiple criteria then sperate them by comma and place them in single curly brackets.

**Note:-** when we execute code 20 records will display c=write ir click enter then other 20 records will display.

**Using $:-** we use $ in front of operators and we use $ to access values in a column check $expr operator to get clarify.

**Comparison Operators: -**

$lt less than

$gt greater then

$lte less than equal

$gte greater then equal

$eq equal to

$ne not equal to

$in to find value in array of values

$nin not in.

Above operators function is same as any other programming language.

{field:{operator:value}} for in and not in we specify array of values in [] separated by comma.

db.trips.find({‘tripduration’:{$gt:500}}) having trip duration greater than 500.

Db.trips.find({‘tripduration’:{$gt:500,$lt:1000}}) multiple conditions on single values specify conditions as a array of values separated by comma.

**Logical Operator: -**

$and, $not, $or, $nor

{Logical Operator:[{condition1},{condition2}…]

For $not we need to used other operator in combination with $not.

$nor is complement of or.

$and, $or same as other programming language.

Db.trips.find({$and:[{‘tripduration’:{$gt:500}},{‘user type’:{$eq:’subscriber’}}])

**$Expr: -** it used to compare the value in a field with value in other field or by hardcoded value.

db.routes.find({$expr:{$eq:[‘$src\_airport’,’$dsc\_airport’]}}

here value in src\_airport comparing with value in dsc\_airport.

**$type: -** it used to find that field in the document is a specified data type or not.

db.trips.find({‘tripduration’:{$type:value}})

1-doble, 2 string, 3-oject, 4-array, 5-binary data, 10 null, 9-date etc.

**$exists: -** it used to check whether the specified field is present it document or not if we give true as value it return documents which have that field if we give false as value it returns documents which does not have that field.

db.routes.find({‘src\_airport’:{$exists:True},’dsc\_airport’:{$exists:False}})

db.routes.find($or:[{‘src\_airport’:{$exists:True}},{’dsc\_airport’:{$exists:False}}])

**Cursor methods: -**

Count () – used to count number of documents in collection.

Size () – used to count the documents used with limit and skip.

Limit(value) – it display specified number of documents.

Skip(value) – used to skip specified number of documents in collection.

Sort({field: values}) – it used to sort the documents in collection if value is 1 ascending order if value is -1 descending order.

**Projection: -** Projection used to select the columns which want to display at output. \_id automatically display to avoid it give 0 to it and specify column name and 1 as key value pair to display.

db.routes.find({},{“\_id”:0,”stops”:1,”src\_station”:1})

**Access sub document [embedded document]:** - we can Access sub documents by dot notations.

For example, we have a address field which has a sub document it has zip code then we access by address.zip

db.inspection.find({“address.zip”:1234})

If we have a sub document in zip then address.zip.field as so on for next level sub document.

**Note**: - Show collections to display all collection in database, show databases to display all databases, use [name of database] to use that database.

**Accessing Array: -**

To find whether array consist of particular value.

db.collection.find({field:value})

to find whether multiple values present in array we can achicve this by $all.

db.collection.find({field:{$all:[value1,value2…]}})

to find exact match of array position must be same.

db.collection.find({field:[value1,value2,…..]})

to find array with particular size.

db.collection.find({field:{$size:value}})

to access element in array of embedded documents we use $elemMatch.

db.collection.find({field:{$elemMatch:{field1:value,field:value}}})

**Inserting documents into collections: -**

We can insert the documents into collections using insert () if collections do not present then it creates new one.

We can pass duplicate values into documents except \_id.

If wont pass \_id internally \_id field creates, and a random value assigned to it.

If we pass duplicate to \_id it gives error.

We can insert array, embedded document, array of embedded documents and embedded documents with array.

**Delete documents in collection: -**

We can delete documents by deleteOne and deleteMany.

deleteOne delete first matching document where deleteMany deletes all matching documents.

We need to specify the filter condition based on that condition it delete the documents in collection.

db.collection.deleteOne({field:value})

db.collection.deleteMany({field:value}) here field:value is nothing but a filter condition if we place it in find it returns records matching this criteria where in deleteMany it deletes documents which are matching to criteria for deleteOne it delete first document which matches that criteria.

To delete embedded and array documents we can write same as in find but replace find with deleteMany or DeleteOne based on requirement.

If we didn’t pass criteria it deletes all documents in collection for deleteMany and first record for deleteOne.

**Update Documents in Collection: -**

We can update documents in collection updateMany and updateOne both are same updateMany updates multiple documents based on matching criteria where updateOne updates first document based on matching criteria.

If we won’t mention any filter criteria it updates all documents.

We update documents through $set operator. We can update multiple columns at same time, and we can update embedded documents.

If we given new filed without criteria than a new field added to the documents.

**$set** used to create new column or to modify value to existing column.

db.updateOne({filter},{$set{field:value, field.field1:value,….}})

db.updateMany({filter},{$set{field:value, field.field1:value,….}})

$unset used to remove a field from document.

db.updateOne({filter},{$unset{field:””, field.field1:””,….}})

db.updateMany({filter},{$unset{field:””, field.field1:””,….}})

**$push** used to add element or elements to existing array if array not found it creates new.

db.updateMany({filter},{$push:{“field”:value/array of values}})

db.updateOne({filter},{$push:{“field”:value/array of values}})

**$rename** it used to rename the field if field name does not present no new field is added.

db.updateMany({filter},{$rename:{“old field name”:”new field name”}})

db.updateOne({filter},{$rename:{“old field name”:”new field name”}})

**$inc** used to add value to increment the existing value of a field if not found it created new field with value specified.

db.updateMany({filter},{$inc:{“field”:value}})

db.updateOne({filter},{$inc:{“field”:value}})

lot more opeartors we can use in update. And we can use multiple operators at same time in update separated by comma.

**Upsert**:- Upsert insert record into collection if it is no record is matched with filter criteria. By default, upsert in false make it true to insert document if it not matched to any existing record.

db.collection.updateMany({filter},{operators:{“field”;”value”}},{upsert:True})

**Note**:-we can update documents in atlas by click on pencil icon but we can modify one record at a time to modify multiple records use shell.

**Aggregation pipeline: -**

Aggregation pipeline has several stages one stage outputs connected to other stage. We can use n of stages n number of times. It has stages like $match, $project, $group etc.

**$match: -**

$match used to match the documents in the collection usually it used as first stage in aggregation but we can use at any place and any number of times.

db.collection.aggregation([{$match:{“field”:value…..}}])

**$project: -**

It same as projection parameter in find but here we can assign one field values to other by using $ and we can derive values from existing fields and assign to new fields. Here just display assigned value but underlying data does not change.

db. collection.aggregation([{$project{“field”:value,”New\_field”:”$field”….])

**Arithmetic expression operators: -**

$add used to add the values {$add: [value1, value2,…]} we can use $add for dates also.

$sub used to subtract values {$sub: [value1,value2,….]} we can use $sub for dates also

$multiply used for multiplication {$multiply: {value1, value2..]}

$divide used for division {$divide: {numerator, denominator]}

$round used to round the values {$round” {value, no.of.decimals}

We are unable to use the arithmetic expression operators directly in match use them with $expr.

**String expression operators: -**

$concat used to combine two or more strings {$concat:[string1,string2,…]}

$toLpper used to change lower case string to upper.{$toUpper:[string1,string2,….]}

$toLower used to change upper case string to lower case string.{$toLower[string1,string2,….]}

$regexMatch used to find field which have a specified string.

{$regexMatch{input:field, regex:”string to search”}}