MongoDB

MongoDb is database server environment which allows to interact with the database , in the MYSQL we work with tables but in mongodb we work with collections.

-inside these collection we have documents and they look like javascript object and in this way we store the data.

In mongodb we interact with the mongo server , which intern interacts with the wired tiger storage engine and which interacts with the file system .in mongodb the queries are written in jason format and intern converted into bson format by the mongo server

-The advantage of mongodb is the flexibility of data storage which it gives.we use a Json format to store the data.

In mongodb there is no command to create a new database , but when inserting the data into the database, the data base will be created on the fly.Another feature of mongodb is that it creates a new id , whenever a new document is inserted to the collection.we can also explicitly configure the id

-show dbs/show databases- This command is used to see the existing databases.

-Use.databasename – This command is used to switch to that particular database

**CRUD operations**

-db.collectionname .insertOne({key name: value , keyname2: value})—This command is used to insert data into the database through creating a collection as well.(insertone is used to insert one document to the collections)

-db.collectionsname.insertMany([{document},{document}]): this is used to insert more than one document to the collections, but here we insert the data in the array format

-db.collectionsname.find(): This is used to display the data in the database. We can also find the data based on the filter i.e through addressing some key.

We can aslo do findOne to print the first value from the filter.

If the number of data is more, then find command does not show all the documents but it provides what is known as curser object and displays only first 20 documents, which can be used to cycle through the documents.

We can use find().toArray() to get all the documents.

Find.forEach((variable) => {printjson(samevariable}) –this command also helps to print out all the documents.

Find({},{document wanted})--There is projection concept inorder to filter the data

-\_id : id name – this command helps to create a id in the mongodb

-db.collectionname.delelteOne(key to be deleted with some condition):This command deletes one element that it finds first in the document.

There is also deletion of multiple data , db.collectionname.deleltMany({}) this deletes all the documents.

-The term filter in the syntax is that , to find the value to work with and make operations on that data.

db.collectionname.deleteMany({filter})-this deletes all the filtered data in all the other documents.

-db.collectionname.updateOne(filter, document updating): this is the method use to update the mongodb collections.we must insert $set in between document updating in the syntax to update the data due to the atomic error that it provides , this helps in updating the document rather than replacing it completely

There is also updation of multiple data , db.collectionname.updateMany({}, {$set:{to be updated}})

We use $set here too

-db.collectionname.update({filter},{ updating the values})—this command updates all the fields in the document. And all the existing enteries are lost.

-db.collection.replaceOne({filter},{document}): this keeps the id and replaces the values in the document

-embeded document: this is a document having one document inside another document.

-we can use ‘.’ Inorder to go deep into the subdocuments to filter or access those data.

Db.dropDatabase()-this command is used to drop the database.

**Schemas**

Mongo dB does not have any schemas, we can create documents in however format we want , and it may not even have the same format.

Datatypes

There are different datatypes in mongodb

Integers(32bit) , text , objecte id, arrays, double(64bit), isodate, embedded document, timestamp, Boolean.

Relations

One to one relation : we use nested embedded document when we are having strong one to one relation . rather declaring the data in two different document we can declare all the required fields in the same document , so that when we query for the display, we can display everything.

-We can also use different collections inorder to store data depending upon the application which we are using.

One to many relations: In the many to one relationship we use different collections and then reference them for relations . But we can also use embedded document to provide the relation for one to many relationship

Many to many to relationship : we can use referencing to find the relationship between them , we can also use embedded documents for building the relationship

So we have to look into it inorder to provide the relationship. For one to one and for one to many we use embedded document and for many to many it is good to use referencing .

-db.anycollection.aggregate([{$lookup : {from : ‘other collection to relate to ‘ , localField : ‘the relation in the current collection which is referencing the other collection object id , foreignField: the relation in the other collection , as alias}}])

This lookup combines the other collection documents to the current document.but does not alter the current collection.

Schema Validation

This schema validation is used to maintain a strict schema format in the collection documents, when a collection is created using schema validation , the enteries into the schema should be in the specified format else it will provide an error.

Syntax; db.createCollection(collection name , {

Validator : {$jsonschema : {

Bsontype : object,

Required : [document field],

Properties: {

Docment field : data type to receive,Description : ‘error descrption ‘ }}});



We can also set the validation level to warning or error , when there is no correct entry made into the data base. By adding validationAction : ‘warn’ or ‘error’ after the closing parenthesis of validator and also changing the syntax as db.runCommand{collMod: followed by the same declaration done in the above screenshot for posts till the end.

By default it is error but we can change it to warn.

Deep insert concepts:

The insert in the mongodb is ordered and if we write a data to the data base that is already present in the data base then it will throw an error and it will not proceed to enter the other documents that is next to it , so by using another document with ordered and making it as false inserts the new documents to the database and neglectst the duplicate docume nts.This happens when we set the object id to the custom id and try to enter the data using the same object id.

Syntax: db.collections name .insert({},{}, {ordered : false})

-write concern : in the mongodb storage engine there are journals ,which is a todo list and this is helpful inorder to store the to do list as the storage engine has to perform to write the data to the disk. We can aslo use a acknowledge command to let us know whether the entry was made in the server or not

Syntax : db.collection name.insert({}{},{writeConcern : {w : 1 , j: true/undefined , wtimeout: time value}})

W indidcate whether the entry is made to the data base or not

J : this indicates whether the entry is made to the journals , which acts as a backup when the server is down , to start the process again.

Wtimeout: this is the time limit for the storage engine to write into the database.

-Atomictiy:this is the feature of the mongodb to insert the data into the database, either all the documents are inserted into the database or nothing is inserted , it doesnot insert some fields in the documents and neglect others, but in insert many command it is subjected to one document at a time , so it is required to use ordered command as if one document fails to get inserted the other documents next to it will get inserted.

-importing files into the mongo db database:

Move to the path where the files are located. Then use the below syntax

Mongoimport filename –d databasename -c collection name --jsonArray(to indicate more files are present) –drop (this is to indicate ,if the data base is already present then drop it and recreate a new database , if not create a new database)

Deep read concept:

There are two types of values in read i.e find method , filter and operators

Different types of operators are query and projection operator

Query selectors and projection operators.

Query selector Types are : comparison , logical , element , evaluation , array , comments, geospacial

-comparison : the operators are $eq $ne ,$gt,$lt,$lte,$gte

Sytax ; db.collection name. find(value :{comparison : condition}})

Eg: db.collectionname.find({value: {$gte: 1}})

We can query subdocuments by using ‘ .’ and for arrays we can use [] to get the equality.

$in and $nin : this comparison helps in getting the values more than one since the $in and $nin takes the array and prints out that values that are equal to the values in the array.

Db.collectionname.find({value: {$in : [value 1, value2]}})

Logical operator: $or , $and , $not ,$nor

Syntax: db.collection name .find({$or : [{values :{comaparison: condition}}, {values : {comparison:codition}}]})

Mongodb defaults the and operation

i.e db.collection name .find({value :{comparison : condition} , {values: {comparison : condition }) so here it default does and operation

Element operator :

$exists operator : this operator is used to provide the results which contain a certain type in the document.we can also use comparison operator with the element operator by inserting them after comma.

syntax ; db.collectionname.find({value : {$exists: true}})

Type operator: Type operator is another type of filter which is used to filter the values by data type .

Syntax: db.colletionname: find({value : {$type : datatype}})

Evaluation operator:

$regex : regular expression , this is used to find the texts in the values.it has the same syntax as the other operator except the {$regex : /text/}

$expression operator : this is used to compare two fields in the same document and to compare both of them

Syntax: db.collectionname.find({$expr: {comparison operator : [$value 1 , $value2]}})

We can use them to write complex queries.below is the example.

db.sales .find({$expr: {$gt:[{$cond:{if :{$gte: ['$volume' , 190]},then : {$subtract:["$volume" ,30]}, else: "$volume"}},"$target"]}}).pretty()

Array quering:

$size operator : this is used to find the total number of array elements . its syntax is similar to comparison operator.

Db.collectionname.find({value: {$size: no of array element to be filtered for}})

$all operator : this is used to find all the values without any order .

Db.collectionname.find(value: {$all: [the values to filter for ]})

Projection ;$ , $elemMatch , $meta , $slice.

$elemMatch : this method is used to filter the values of subdocument and within the same document and not in the other subdocument.

Syntax: db.users.find({document: {$elemMatch: {subdocument : condition, subdocument:condititon}}}).pretty()

Eg : db.users.find({hobbies: {$elemMatch: {title : 'sport' , frequency: {$gte:2}}}}).pretty()

-sort ; this method is used to sort the documents in ascending or in descending order , 1 is used for ascending and -1 is used for descending the documents.

Db.collectionname.find().sort({value1 : condition , value2: condition})

-skip : this method is used to skip a certain number of documents.

Db.collectionname.find().skip(no of documents to skip)

-limit: this allows to get a certain amount of document and to neglect other .

Db.collectionname.find().limit(value to print out)

-Slice : this is another function which is used to get the documents but in the subdocument only the specific amount of it and it takes two arguments , in the first one it specifies how many values to skip and second element indicates the number of values to be printed.

Db.collectionname.find({value : {$slice:[1,2]}))

1 – indicates to skip first element

2- indicates to print two element after the first element.

Update deeper concept

$set method ; this is used to update the values in the existing data base

Syntax: db.collectionname.updateOne{{filter}, {$set: {fields to update}})

$set method can be used to update more fields also

Incrementing and decrementing the value: by setting the value to 1 we can increment and by setting the value to -1 and so on we can decrement the values. The syntax is same as the $set method.

Min, Max and mul: min updates the value only if the value to be set is lower than the existing value.

Max updateds the values only if the values to be set is higher than the existing value.

Mul is used to multiply the existing value with the some number.

The syntax of all three is similar to the set method

-unset operator is used to drop a field in the document

The syntax : db.colectionsname.updateMany({filter},{$unset: {fieldname:””}})

Rename : this is used to rename the field name

db.users.updateMany({},{$rename: {field: ‘tobe renamed to’}})

db.users.updateMany({}, {$rename : {name : ‘total age]}})

Upsert : this is used to update the existing field if present or if the fields are not present then it goes ahead and creates a new field.

Syntax: db.users.updateOne({name: 'maria'}, {$set: {age: 20, hobbies :[{title: 'good food' ,frequency: 3}],issporty:true}}, {upsert: true})

We have certain function to get access to the current document where we are example is below and marked in bold

db.users.updateMany({hobbies: {$elemMatch: {title : 'sport' , frequency: {$gte: 3}}}}, **{$set:{'hobbies.$.highfrequency': true}})**

we can update the all the fields inside the array documents if a certain conditions are met find the eg: db.users.updateMany({totalAge: 30},{$inc:{'hobbies.$[].frequency' : -1}})

we can update only the specific fields in the subdocuments . by using array filters find the below example; db.users.updateMany({'hobbies.frequency': {$gt : 3}}, {$set:{'hobbies.$[el].goodfrequency': true}}, {arrayFilters: [{'el.frequency':{$gt: 3}}]})

$push: this works similar to the $set , inorder to add an extra array into the existing fields.

We can use $push to add more than one fields also into the array. And we can use $each method to go into that array so that we can make an entry into that array.

Eg: db.cname.updateOne({condition},{$push: {document: {$each: {condition}}})

We can use pull operator to pull or remove the fields from the array, but the push command will enter the duplicate values.we can aslo use addToSet command to enter the values into the array documents but it wont add the duplicates.

db.users.updateOne({name: 'maria'},{$pull : {hobbies: {title: 'hiking'}}})

we can use pop command inorder to remove the first or the last element in the array.find the example below

db.users.updateMany({name: 'maria'},{$pop: {hobbies: -1}})

delete deep concepts

db.users.delete({filter}): to delete a specific fields from the database

db.users.delete({}) .this is used to delete the entire documents in the database

db.collectionname.drop(): this deletes the entire collections

db.dropDatabase() : this deletes the entire database

indexes:

an index may speed up our update , find , delete quires.

Index scan , this sorts the all the data so that while finding the data the mongodb just jumps directly to that value and provides the keys which is the pointer to the document and fetch inside the index scan then points to the collections and pulls the data.

Syntax: db.collectionsname.createIndex({“field name “: 1{ascending} -1{descending})

Indexes should be used only there are limited values to query , since it slows down the process if there are more values to return.

Deleting the index:Db.collectionname.dropIndex({“field name “: 1{ascending} -1{descending})

We use explain({executionStats}) to get the stats results for the scan. And other stats are query planner , all plan execution stats

We can also use complex index scan for scanning more than one conditions

Syntax: db.collectionsname.createindex.({condition1}, {condition2}})

Here we can use to output the results for condition 1 only but we cannot use to output the results for condtiton2 only.so if we use to find the results of condition2 only then it runs full scan.

-db.collectionname.getIndexex() – this command is used to show the indexes

Mongodb creates a default index by id .

-we can also create a unique index when we want to create a indexes to be unique

Syntax; db.collectionname.createIndex({index condtion},{unique: true})

-partial index: This is used to create a index document with only required fields

Syntax: db.collectionname.createIndex({index condition}, {partialFilterExpression : conditionto create it})

-there is a concept called time to live , this destroys the data after the specified amount of time ;

Syntax: db.collectionname.createIndex({indexcondition}, {expireAfterSeconds:value})

Covered queries.

This is to provide the optimal solution while finding the solution , not only the pointer but also the data value will be in the index so we can use some covered query method to find the data without going into the document.

Syntax: db.collectionname.find({filter},{\_id : 0 , name: 1})

Rejection plan: this concept is that , if we have similar two or more indexes then mongo db runs both of them and then checks for certain threshold condition if any one it succeeds then it implements that as the best indexing , and caches that solution until some condition are met inorder to remove it

Multikey index: when we create index , mongodb stores the values of the array inside the index, so that the query may be fast.multikey comes into picture when indexing with arrays or subdocument.

Textindex: this is used to find the text values and this stores the text as separate items in the index array so that It can query them faster.

Syntax; db.collectioname.createIndex({value : ‘text’})

Db.collection.find($text: {$search: ‘text value’}})

Setting the default languages: default\_language: ‘language type ‘

This is used to provide the more weight to the some values during the mongodb scoring : Weigths : field: value , field: value

To find the score value : $meta : ‘textScore’

The concept for indexing is foreground and background , by default we create index in the foreground , this locks the collection while until the index is created, so it takes time to find or insert command is executed in parallel with the index creation command.

But we can fix this by using background indexing which helps to not to lock the collection during the index creation

Syntax : db.collectionname.createIndex({filter}, {background : true})

Geospatial data

Here we are used for the location to query. We save in the GeoJason object format , we have type and followed by the coordinates for that location and hence that becomes a Geojason data

Once the location is set in any document we can find the near by locations using the $near and followed by the $geometry and then mentioning the type and coordinates of the location that we want to check.Inorder to locate them we should create a index

Db.collectionname.createIndex({indexname: ‘2dsphere’})

Syntax: db.collectionname.findOne({location: {$near:{$geometry: {type: ‘point’ , coordinates: [longitude , latitiude]}}}})

We use $maxDistance and $minDistanace to find the location that is near to the location that is specified in during declaration. i.e we can enter the location and near by and find whether it near to the location that is declared or not.

We can configure 4 points and take in the longitude and latitude values and then we can insert the coordinate values inside a collection and we can find whether it is inside the 4 points or not .

Syntax: db.collectionname.findone({$geoWithin : {$geometry: {type : ‘polygon’ , coordinates: [[the constant that contain the points]]}}})

We can find whether the location is within the specified points by using goeIntesects object and the syntax I similar to the above geojason creation type.

Aggregation Frame work

Aggregation frame work is the series of pipe line method which is used to retrieve the data as how we want.

First step to retrieve the data is match function and this works similar to find and all the filter operation holds good hear too

Db.collectionname.aggregate([{$match :{fiter}}])

The next step to analyse the data is group stage and this is similar to the group by in sql this groups the similar type and sums up the values.

Db.collectionname.aggregates({group: {name: {‘$the documentlocation’} , {name : {$sum : 1})}})

Next we can sort the data that is grouped

Db.collectionname.aggregate({$sort : {name: -1/1])

Next phase is project phase and this is transformation phase and this works similar to the projection phase.

Db.collection.aggregate([{$project: {condition fields})

We have a stage known as $unwind stage which ungroups the elements inside the arrays

Db.collection.aggregate([{$unwind: field which has that}])

$slice is used to pull the required number of array values from the array document

$slice: [“array on which to work with” , from the element to start with, the index number to print ]

$size is used to get the length of the array

$size : [‘array to work with ‘ ]

$filter method is used to print out the filter results

$filter: {input : ‘location of the variable ‘ , as : ‘any variable name’ , cond : {condition operator to filter the result(here we use $$any variable name to work on the conditions .}}

$bucket field: This field is used to put the variables into certain categories so that we can distribute the data in different categories.

$bucket: {$groupBy : ‘name of the field’ , boundaries: the categories , output: the fields that to be outputted.}

Bucket auto is the feature of mongodb to create its boundaries bucket. The extra field in the above syntax is buckets: no of buckets

$out this stage is used to create a new collection for the staged output values.

Syntax: $out : collectionname.

Numeric types in Mongo DB

We have four different types of data types they are int34 , int64 , double 64 and high precisiondouble128bit, by default the shell uses 64bit double.

The command for the number int is numberInt(value ) for 32digit integers

For the double : numberDouble(value) for 64digit integers

For the highprecision double we use : numberDecimal(value)

Since the 64digit double does not provide the great precession so we have to go with 128 bit high precision if we need more precission for some task to work on with the digits.

Security

Mongo db provides role based authentication and authorization

We should open the server using mongo –auth

We use createUser({user: “user name’ , pwd : ‘password’ role : [“userAdminAnyDatabase”})

This role has the access to create more users and has the access to all the databases.

The built in different roles of the mongodb

User ; read , readWrite

Admin : dbAdmin , userAdmin , dbOwner

Special roles: readAnyDatabase, readWriteAnyDatabase, userAdminAnyDatabase, dbAdminAnyDatabase

Clusteradminroles: clusterManager, clusterMonitor , hostManger, clusterAdmin

Superuser: dbOwner(admin), userAdmin(user),userAdminAnyDatabaseroot

We can provide the user the access to a specified database.

We can add extra roles to the users through db.updateUser(username , {role :[roles , {role , db : database}]})

Db.getUser() to check the users on the database

Mongodb uses secure socket layer(ssl ) technique to transfer the data through encryption , we need to configure the ssl in the shell.

Fault tolerance

This is the method used to create a good queries and to write all the commands properly so that the sever runs efficiently . There are multiple methods to restrict the fault tolerance i)sharding ii)replica sets

Capped collections: this is new type of collections where we limit the amount of data and the old documents will be deleted when the size exceeds the specified limit.

Db.createCollection(collectionname , {capped : true , size : size for the collection , max : max number of collection to hold })

Replica sets: we by default connect to the primary node and when we insert the data to the primary node , if we have secondary node then the it will automatically be written to the secondary , and this is done asynchronously ,this means we are replicating it , so when the primary node is offline we can use secondary to get the data.

Sharding: this process is to divide the workload between many mongo db servers so that the application can have more requests and the servers can provide assistance easily.there is a middle man mongos which is used to route the requests to different mongodb servers.

Transactions

-This is to inform the mongodb server to roll back the server to that point where the documents were effected.

Stitch

This is a feature that the mongodb company provides where we can configure the application without worrying about the backend logic i.e to start the server and manage the server