501 : Advanced Web Designing

Unit-1: Concepts of NoSQL: MongoDB

1.1 concepts of NoSQL. Advantages and features :

- NoSQL Database is a non-relational Data Management System, that does not require a fixed schema.
- NoSQL Database does not use any kinds of Joins and is easy to Scale Out(Highly Horizontally Scalable).
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL database stands for “Not Only SQL” or “Not SQL.”
- Carlo Strozzi introduced the NoSQL concept in 1998.
- a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data.

Types of NoSQL Databases:

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented

1)KEY-VALUE pair based :

This type of database uses key-value pairs to store data.it is designed in such a way that it handles a lot of heavy load. It stores data in hash tables and each key is unique. The values can be String, JSON, blob etc. It is one of the most basic type of NoSQL database types. They can be used as collections,dictionaries,associative arrays etc.

Eg: Redis,Dynamo,MemCached

2)Column-Oriented Graph:

This type of database work on columns and are based on BIGTable by Google . Here every column is treated separately and each columns’s values are stored in contagious(continuous) fashion.these are very efficient in huge data aggregation such as sum,count,avg etc.

Eg: Cassendra ,Hbase,HyperTable

3)Document-Oriented :

This type of database stores data as a key-value pair but the value is stored in as a document such as JSON or XML.We do not need to define data that makes it more flexible.

Eg: MongoDB,CouchDB,RavenDB

4)Graph-Based:

This Type of Database uses nodes and edges to store the data where entities become nodes and their relation become edges . Every node and edges have unique identifiers. These database are Multi-relational and traversing relations is easier and faster since there is no need to calculate them.

Eg: Neo4J,Infinite Graph,FlockDB

CAP Theorem:

This is also called Brewer's Theorem that states that a NoSQL database can never guarantee All three properties at the same time. i.e it is impossible for a distributed database to provide more than two of the three guarantees :

1) Consistency: The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data.

2) Availability: The database should always be available and responsive. It should not have any downtime.

3) Partition Tolerance: Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable, i.e if there is unavailability in one part , other parts must remain unaffected.

Eventual Consistency: this term refers that there must be replicas created for the data store on multiple machines to get availability and scalability. Changes made on one replica must be propagated to others but it may take time. Since data may be immediately updated on some whereas it may take time to update on others,

making it consistent not immediately but with course of time making it “Eventually Consistent”.

Unlike ACID standards used by Relational Databases , the NoSQL databases use the BASE standards where :

B : Basically

A : Available

S : Soft State

E : Eventually Consistent.

FEATURES OF NoSQL :

1) Non-relational :
   NoSQL databases are non-relational and so there is no flat fixed-column records. They work with self-contained aggregates or BLOBS. Also they do not require complex features like Query languages , Joins , ACID , data-normalization , relation mapping etc.

2) Schema-free :
   They are schema free or have relaxed schemas i.e not fixed . they do not need to define any kind of table schema for the data. Data is usually stored in heterogeneous Structures.

3) Simple API :
   They have easy to use interfaces for storage and Querying results. As well APIs provide low level data manipulation and selection modes. They use HTTP REST with JSON.

4) Distributed :
   Multiple NoSql databases can be executed in distributed fashion. Share Nothing Architecture is used in NoSQL so there is less coordination and more distribution. Provides auto-scaling and fail-over capabilities.

ADVANTAGES OF NoSQL:

1) No Single Point Of Failure:

   This asserts that a single part of program can not effact entire system from working. i.e changes in single part of data store does not affect other parts of data store and ultimately entire database.

2) Easy Replication:

   The data in database can be easily replicated because of the distributed architecture.

3) Flexible Schema Design:

   There is no fixed schema and schema can be altered whenever required without any downtime or service disruption.

4) Supports Unstructured, Semi-Structured and Structured datas:

   Unlike relational databases that support only structured data. NoSQL provides support to all unstructured, semi structured or structured data with the help of flexible schema.

5) Horizontally Scalable:

   NoSQL allows to handle database across several servers rather than single server. Due to this we can add up more inexpensive servers to scale horizontally, rather than upgrading the single server to more powerful one that is very expensive to scale vertically like in RDBMS.

About MongoDB :

mongoDB is an open source , object oriented , cross-platform prominent NoSQL database system that provides high performance, availability and easy scalability.
It is a document-based NoSQL database system , that uses documents like JSON or XML as values.

Terminologies :

Database : database is a container for collections of documents in mongoDB. Therefore it can be said as collection of collections. Each database in mongo gets its own set of files in filesystem and a single mongo server can have multiple databases.

Collection : a collection to a MongoDB is what a table is to RDBMS . the collection is a group of related or similar MongoDB documents. Collections do not enforce schemas and exist within single database.

Document: a document is a set of key value pairs. They have dynamic schemas i.e documents in same collection do not need to have same set of fields or structure.

Advantages of mongoDB:

- There is no fixed schema. Any document of same collection need not require to contain same fields or contents as another.
- Structure of single object is clear.
- There are no Complex joins
- Mongo supports dynamic Queries using document based query language that is as efficient as SQL.
- It is easy to scale out(horizontal scaling of NoSQL).
- There is no need of conversion or mapping of objects to database objects.
- They use internal memory for storing the working set, enabling faster access of data.

1.1.1 MongoDB Datatypes (String, Integer, Boolean, Double, Arrays, Objects):

Following are a few dataypes that mongoDB supports :

1) String: The most common datatype is string. Mongodb accepts UTF-8 strings
   Eg: db.mydb.insert({“String example”: ”this is a string”})
2) Integer: This type is used to store numerical data. It can be 32 or 64 bit depending on server.
   Eg : db.mydb.insert({"Integer example": 62})
3) Boolean: This datatype is used to insert True/False values.
   Eg : db.mydb.insert({“Boolean example”:True})
4) Double: This datatype is used to insert Floating Point/decimal values.
   Eg : db.mydb.insert({“Double example”:123.5462})
5) Arrays: This type is used to store arrays, lists or multiple values into one key
   Eg : db.mydb.insert({“Array example”:[“HELLO”,”CIAO”,”BONJOUR”]})
6) Objects: Object datatype is used to store Embedded Documents in MongoDB.
   Eg : db.mydb.insert({“Object example”:{“name”:”Rahil”,”age”:20,”percentage”:91.1231,”Extras”:[“Taekwondo”,”Cricket”,”Badminton”]}})

There are numerous more datatypes available in mongoDB like Date,Timestamp,Binary,Regular Expression etc.

Note: If a document contains another document in the form of the key-value pair then such type of document is known as an embedded document.

Note:BSON stands for “Binary encoded JSON” or “Binary JSON”

1.1.2 Database creation and dropping database:

1) Creating a database in MongoDB:

   To create a database in the mongoDB we need to use the “use” command.

   Syntax:

   use database_name

   example:

   use mydb

2) Dropping a database in MongoDB:

   To Drop a database in mongoDB we use the dropDatabase() method.

   Syntax:

   db.dropDatabase()

   example:

   use mydb
   db.dropDatabase()

   Note : Other commands for Databases :

   1) To show database list : “show dbs” {provided there is atleast one record in database}
   2) To show currently selected database :”db”

1.2 create and Drop collections:

1)Creating a collection:

To create a collection in mongoDB we use the createCollection() method.

Syntax : db.createCollection(name,[options])

Here , there is one Compulsory argument name and various other optional arguments that can be passed.

- name : It is string value used to pass the name of collection to be created.

Optional arguments :

- capped : it is a Boolean value stating whether or not collection to be capped. A capped collection is a collection of fixed size that re-writes oldest record when max size is reached. If set true, size parameter is also needed to be specified.

- autoIndexId : if true , creates index automatically on id fields. By default false

- size : it is used to specify the size of capped collection and compulsory if capped is true.

- max : it is used to specify maximum number of documents allowed in capped collection.

Eg1 : db.createCollection("Mycollection")

Eg2:
db.createCollection("Mycollection",{capped:true,autoIndexID:true,size:31696,max:1000})

mongoDB also creates collection on its own if we use insert command to insert some document.

2)Drop a collection:

The drop() method is used to drop a collection from database in mongoDB.

Syntax : db.collection_name.drop()

Eg: use mydb

db.mycollection.drop()

1.3 CRUD operations (Insert, update, delete, find, Query and Projection operators):

1) Insert Operation :

To insert data into mongoDB collection , we can use several insert() methods or save() method.

- Insert() method :
  This is used to insert a document into a collection.
  Syntax : db.collection_name.insert(document)
  Example : db.users.insert({_id:1,name:"rahil",age:"20"})

- Save() method:
  This method can be used to insert a document if the _id is not specified. Else it would replace data of document containing _id specified with that specified in the method.
  Syntax : db.collection_name.save({New document})
  Example : db.students.save({"email":"rahilp33@gmail.com"})

- insertOne() method:
  This method is used when only one document is to be inserted into collection.
  Syntax : db.collection_name.insertOne(document)
  Example : db.students.insertOne({ Name:"Rahil" , class:"TYBCA" ,age:20})

- insertMany() method:

  This method is used to insert multiple documents into collection. You need to pass an array of documents into this method.
  Syntax :
  db.collection_name.insertMany([doc1,doc2,doc3…docN])
  Example:db.students.insertMany([{ Name:"Rahil" , class:"TYBCA" ,age:20},{ Name:"Vivek" , class:"TYBTech" ,age:20}])

2) Update Operation:

   To update the document in mongoDB collection we have several methods enlisted below:

- Update() method:

  This method updates the values in existing document.
  Syntax : db.collection_name.update({selection criteria},{$set:{new data to be updated}})
  Example:db.students.update({'name':'rahil'},{$set:{'age':'21'}})

- Save() method:

  This method can be used to update records as well as insert records.This method replaces the existing document with new document passed as the argument.
  Syntax : db.collection_name.save({_id:id,New document})
  Example :
  db.students.save({_id:1,"email":"rahilp33@gmail.com"})

- findOneAndUpdate() method:

  It updates data in the first matched document that matches selection criteria with the new data provided in parameter. Even if multiple documents match only first match is updated.
  Syntax : db.collection_name.findOneAndUpdate({selection criteria},{data to be updated},[optionals])
  Example :
  db.students.findOneAndUpdate({name:"rahil"},{$set:{"phone":" 9054124121"}})

- updateOne() method:

This method is used to update the first matched document with the collection based on the given query. This method updates only one document at a time.
Syntax : db.collection.updateOne({selection criteria},{$set:{updation data}})
Example :
db.employees.updateOne({“name”:”vivek”},{$set:{“post”:”manager”}})

- updateMany() method:
  This method is used to update all the matches with new data that match the given selection criteria.
  Syntax : db.collection.updateMany({selection criteria},{$opt:{updation data}})
  Example:db.mycollection.updateMany({“post”:”manager”},{$inc:{salary:1000}})

3) Delete Operation :

The remove() method is used to delete documents in the Collection of mongoDB.
It takes two parameters , deletion criteria that intakes the query to remove document and other one is JustOne that states whether only one document is to be removed or multiple by switching it to true or 1 or keeping blank i.e false.

Syntax : db.collection.remove({deletion criteria}[,JustOne])

Example :
1)Removing all (unconditioned) : db.collection1.remove({})
2)Removing all (matching condition) :
db.collection1.remove({“post”:”HR”})
3)Remove one (matching condition) :
db.collection1.remove({“post”:”HR”},1)

4) Find/Query Operation :

In MongoDB the find() method is used to query out results/documents from a collection. This method returns a cursor to the retrieved documents.

Syntax : db.collection.find({query})

Examples :
1)All documents : db.collection2.find()
2)All matching the condition : db.collection2.find({“name”:”Rahil”})

There is also a findOne() method that returns only one document from collection.
Syntax : db.collection.findOne({})
Example :
1)unconditioned: db.collection3.findOne()
2)conditioned: db.collection3.findOne({“post”:”manager”})

1.4 Operators (Projection, update, limit (), sort ()) and Aggregation commands :

Query Operator	Syntactic Format	Description	RDBMS equivalent
Equality	db.mycol.find({“key”:”value”}) or db.mycol.find({“key”:{$eq:”value”}})	It is used to represent equality condition and checks if field is equal to value	Where col = value
Less Then	db.mycol.find({“key”:{$lt:”value”}})	It is used to represent less than condition and checks if field is less then value	Where col < value
Less then Equal to	db.mycol.find({“key”:{$lte:”value”}})	It is used to represent less than or equal to condition and checks if field is less or equal then value	Where col <= value
Greater then	db.mycol.find({“key”:{$gt:”value”}})	It is used to represent greater than condition and checks if field is greater then value	Where col > value

Greater then Equal to	db.mycol.find({“key”:{$gte:”value”}})	It is used to represent greater than or equal to condition and checks if field is greater or equal then value	Where col >= value
Not Equal to	db.mycol.find({“key”:{$ne:”value”}})	It is used to represent not equal condition and checks if field is not equal to value	Where col <> value
In	db.mycol.find({“key”:{$in:[”v1”,”v2”,”v3”,…”vn”]}})	It is used to check if field is present in array	Where col in(v1,v2...)
Not In	db.mycol.find({“key”:{$nin:[”v1”,”v2”,”v3”,…”vn”]}})	It is used to check if field is not present in array	Where col not in(v1,v2…)
And	db.mycol.find({$and:[{key1:val1},{key2:val2}]})	It is used for logical “And” to apply multiple conditions and check if all are satisfied	Where col1 = value and col1 = value
Or	db.mycol.find({$or:[{key1:val1},{key2:val2}]})	It is used for logical “Or” to apply multiple conditions and check if one of those are satisfied	Where col1 = value or col1 = value
Nor	db.mycol.find({$nor:[{key1:val1},{key2:val2}]})	It is used for logical “Nor” to apply multiple conditions and check if none are satisfied	-
Not	db.mycol.find({key:{$not:{value}})	It is used to negate condition results	not
Projection Operator	**Syntactic Format**	**Description**	**RDBMS equivalent**
$	db.collection.find({array:condition},{array.$:1})	Used to return the first element of an array in “Query” condition.	-
$elemMatch	db.collection.find({field:{$elemMatch:condition}})	Used to return first element in array that matches the “elemmatch” condition	-
$meta	db.collection.find({query},{field:{$meta:keyword}})	The meta operator returns the result for each matching document where the metadata associated with the query.	-

$slice	db.collection.find({field:value},{array:{$slice:count}})	It controls the number of values in an array that a query returns	-
Update Operator	Syntactic Format	Description	RDBMS equivalent
$currentDate	db.collection.updateOne({field:value}, {$currentdate:{field:type}})	It is used to set value of a field to current date	
$inc	db.collection.updateOne({field:value}, {$inc:{field:count}})	It is used to increment value of a field by given count	
$min	db.collection.updateOne({field:value}, {$min:{field:count}})	It is used to update value if specified value is less then current value	
$max	db.collection.updateOne({field:value}, {$max:{field:count}})	It is used to update value if specified value is greater then current value	
$mul	db.collection.updateOne({field:value}, {$mul:{field:num}})	It is used to multiply the value of a field to a number	
$rename	db.collection.updateOne({field:value}, {$rename:{field:newname}})	It is used to update the field name to a new name	
$set	db.collection.updateOne({field:value}, {$set:{field:value}})	It is used to set/update the value of a field to that provided.	
$setOnInsert	db.collection.updateOne({field:value}, {$setOnInsert:{field:value}},{upsert:true})	It is used to asign value of a field if upsert is true and updation takes place	
$unset	db.collection.updateOne({field:value}, {$unset:{field:””}})	It is used to delete a particular field from document	
$	db.collection.updateOne({field:value}, {update optr:{array.$: value}})	It is used as placeholder to update only first element in array.	
$[]	db.collection.updateMany({},{Update optr:{array.$[] : value}})	It is used to update all elements in a given array.	
$[identifier]	db.collection.updateMany({}, {Update optr:{array.$[identifier] : value}} ,{arrayFilters:[{field:value}],upsert:true})	It is used to identify and update the value of array element that satisfies in array filter condition.	

$addToSet	db.collection.updateOne({field:value}, {$addToSet:{array:value}})	It is used to add element to array if it is not already there.	
$pop	db.collection.updateOne({field:value}, {$pop:{array:-1|1}})	It is used to remove first or last element from the array, -1 for first , 1 for last.	
$pull	db.collection.updateMany({},{$pull:{array:value,…})	It is used to remove all elements that match the value of specified condition.	
$push	db.collection.updateOne({field:value}, {$push:{array:value}})	It is used to append specified value into an array.	
$pullAll	db.collection.updateMany({id:value}, {pullAll{field:[value1,value2…],..}})	It is used to remove all instances of the specified values from an array. pullAll doesn't need condition , it removes listed values instead	
Modifiers	Syntactic Format	Description	-
$each	db.collection.updateOne({field:value}, {$addToSet:{array:{$each:[value1,value2…]}}}) Or db.collection.updateOne({field:value}, {$push:{array:{$each:[value1,value2…]}}})	Its used with addToSet operator to add multiple values to an array if not exist. It is used with push to append multiple values to array.	
$position	db.collection.updateOne({field:value}, {$push:{array:{$each:[set of values],$position:pos}}})	It is used with push to input values into array at certain position. $each is required for position to be used.	
$slice	db.collection.updateOne({field:value}, {$push:{array:{$each:[set of values],$slice:num}}})	It is used to limit number of elements of array during push. It must appear with $each.	
$sort	db.collection.updateOne({field:value}, {$push:{array:{$each:[set of values],$sort:1|-1}}})	It is used to sort the array elements during the push operation using 1 as value for Asc and -1 for Desc. it also must appear with $each.	

- ## limit() method :

  In mongodb limit() method is used to limit the number of field of documents to be displayed. It can be handy when document has tons of field but you only need to display a few. limit() is used with find() method.

  Syntax : db.collection.find().limit(n) or : db.collection.find().limit(n).skip(ns)

  Where n is number of fields required to display (similar to limit in rdbms) and ns is number of docs to skip(similar to offset in rdbms)

  Example : db.mycol.find().limit(2) gives top two records of the retrieved results.

  db.mycol.find().limit(2).skip(1) gives top two records of the retrieved results after skipping 1st one.

- ## sort() method:

  sort() method is used to sort the documents in a collection. This methods accept documents containing list of fields along with their sorting order.1=ASC|-1=DESC.

  Syntax : db.mycol.find().sort({key:1|-1})

  Example : db.mycol.find().sort({“field1”:1}) orders records by ascending order by the values of field1 say name or id or so.

  db.mycol.find().sort({“field1”:-1}) orders records by descending order by the values of field1 say name or id or so.

- ## Aggregate() method:

  aggregate() is used to club multiple documents into one and perform aggregation operations such as sum,max,min over them . in simple words, we can group documents records by some field and then Query some operation like finding max out of those documents, or sum of those documents.

Syntax : db.collection.aggregate([{aggregation pipeline}]);

Or
db.collection.aggregate([{stage:{expression:value,newfield:{accumulator:value}}}]);

- Here stages can be :

  1)$projection - to select specific fields from collection

  2)$match - to filter results using conditions

  3)$group – to group the documents using a field to aggregate results

  4)$sort – to sort results into asc/desc order

  5)$skip – to Skip forward some amounts of documents like offset In SQL used with limit.

  6)$limit – to limit the number of results

- Here accumulators can be:

  1)$sum - to sum values of a field in multiple documents

  2)$min - to find min values out of all documents

  3)$max – to find max values out of all documents

  4)$avg – to calculate average of values of a field in multiple documents

  5)$count – to count number of documents

  Others are $push,$addToSet,$first,$last;

- Aggregation Commands:

1. aggregate:

In simpler sense, aggregation commands are those that are used to group values and perform operations on them together. For reference we can say this as for example a combination of count() or sum() with the group by clause in SQL.

Syntax:
db.runCommand({aggregate:collection,pipeline:[{operations:values}]})

Example: db.runCommand({aggregate:"mycol",pipeline:[{$project:{tags:1},{$unwind:"$tags"},{$group:{_id:"tags",count:{$sum:1}}}],cursor:{}})

2. count:

count command is used to count the number of docs in a collection or view. It returns a doc that contains count and value.

Syntax : db.runCommand({count:collection,[opt]})

Example : db.runCommand({count:'mycol'})

3. distinct:

this is used to find distinct values for given field in a single collection . it returns a doc with array of distinct values.

Syntax : db.runCommand({distinct:collection,key:field})

Example : db.runCommand({distinct:"cars",key:"models"})

4. mapReduce:

the map reduce function is an alternative to the aggregate function, where it uses a mapFunction to map the required key-value pairs and a reduce function to reduce and return the output results according to aggregation operation taking place and field.results are stored in collection name provided in {out:}

Syntax:
db.collection.mapReduce(mapFunction,reduceFunction,{out:"newcollection"})

Example:

var mapFunction1=function(){emit(this.field1,this.field2)

var reduceFunction1=function(key,valueArr){return Array.sum(valueArr)}

db.collection.mapReduce(mapFunction1,reduceFunction1,{out:"results"}).

This function is used to map field2 to field1 for all records and then perform sum of field2 values and return the results into a collection named return.

Works similar to :
db.collection.aggregate([
{$group:{_id:"$field1",value:{$sum:"$field2}}},
{$out:"results"}
])