MONGODB

NoSQL Database

What is MongoDB?

- Developed by 10gen
- It is a NoSQL database
- A document-oriented database
- It uses BSON format
- Dynamic schemas

Features of MongoDB

- MongoDB provides **high performance**. Most of the operations in the MongoDB are faster compared to relational databases.
- MongoDB provides **auto replication** feature that allows you to quickly recover data in case of a failure.
- **Horizontal scaling** is possible in MongoDB because of **sharding**. **Sharding** is partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved.

Features of MongoDB

Horizontal scaling vs vertical scaling:

- **Vertical scaling** means adding more resources to the existing machine while **horizontal scaling** means adding more machines to handle the data.
- Vertical scaling is not that easy to implement, on the other hand horizontal scaling is easy to implement.
- Horizontal scaling database examples: **MongoDB**, **Cassandra** etc.

Features of MongoDB

- **Load balancing**: Horizontal scaling allows MongoDB to balanace the load.
- **High Availabilty**: Auto Replication improves the availability of MongoDB database.
- **Indexing:** Index is a single field within the document. Indexes are used to quickly locate data without having to search every document in a MongoDB database. This improves the performance of operations performed on the MongoDB database.

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document
column	field
index	index
table joins (e.g. select queries)	embedded documents and linking
Primary keys	_id field is always the primary key
Aggregation (e.g. group by)	aggregation pipeline

MAPPING RELATIONAL DATABASES TO MONGODB

Create a Database

- To start MongoDB
  - type Mongo at bin folder of mongo installation folder
- Once we are in the MongoDB shell, create the database in MongoDB by typing this command:
  - **use database_name**
  - Ex: > **use myDB**
- To check the currently connected database just type the command **db**.
  - > **db**

Create a Database

- To list all the databases , use the command **show dbs**
  - **> show dbs**

- To delete (drop) the database
  - > **db.dropDatabase()**
    - this command deletes the currently selected database.

Collections

- **Collections** in MongoDB is equivalent to the tables in RDBMS.
- The data in MongoDB is stored in the form of documents.
- These documents are stored in Collection and Collection is stored in Database.
- The collection in MongoDb can be created in two ways:
  - **Method1** : Creating the Collection in MongoDB on the fly
  - **Method2** : Creating collection with options before inserting the documents

Collection –Method1

- No need to create collection before you insert document in it.
- With a single command we can insert a document in the collection and the MongoDB creates that collection on the fly.
- Syntax:

  db.collection_name.insertOne({key:value, key:value…})

- Example:

  db.student.insertOne ({"name":"Nag","rno":111});

Collection –Method2

- We can also create collection before we actually insert data in it.
- This method provides you the options that you can set while creating a collection.
- **Syntax**:
  - db.createCollection(name, options)
    - **name** is the collection name
    - **options** is an optional field

Collection –Method2

- The options that we can provide while creating a collection:
  - **capped**: type: boolean.
    The default value of this parameter is false. If you set it true then it enables a capped collection. If you specify true, you need to specify size parameter also.
  - **size**: type: number.
    This specifies the max size of collection (capped collection) in bytes.
  - **max**: type: number.
    This specifies the max number of documents a collection can hold.
  - **autoIndexId**: type: boolean
    The default value of this parameter is false. If you set it true then it automatically creates index field _id for each document.

Collection –Method2

- creating collection without any parameters:
  - > db.createCollection("students")
  - { "ok" : 1 }

- Creating collection with options:
  - db.createCollection(“students", { capped : true, size : 200})
  - { "ok" : 1 }

Dropping a collection

- To drop a collection , first connect to the database in which you want to delete collection and then type the following command to delete the collection:
- Syntax:
  - >db.collection_name.drop()
- Ex:
  - > db.student.drop()

Insert Document

- Syntax to insert a document into the collection:
  - > db.collection_name.insertOne()
  - > db.students.insertOne({_id:1,name:"kalyan",dept:"cse"})

Insert Multiple Documents in collection

- To insert multiple documents in collection, we define an array of documents and later we use the insert() method on the array variable.

```
var students=[
        { "name": "Anil", "rno":11
        }
        { "name" :"Bharath", "rno":12
        }]
```

- >db.student.insert(students)

Query Document using find() method

- find() method is used to query all the documents from a collection.
  - > db.collection_name.find()
  - > db.student.find()

- To print the data in JSON format run the command
  - db.collection_name.find().forEach(printjson)

Comparison Operators

- Query Document based on the criteria
  - Instead of fetching all the documents from collection, we can fetch selected documents based on a criteria.
  - Equality Criteria: $eq
    - db.student.find({"name":{$eq:"Ashwin"}})

Comparison Operators

- Greater Than Criteria:
  - db.collection_name.find({"field_name":{$gt:criteria_value}})
  - db.students.find({“rno":{$gt:10}})
- Less than Criteria: $lt
- Not Equals Criteria: $ne
- Greater than equals Criteria: $gte
- Less than equals Criteria: $lte

Comparison Operators

- $in
  - The $in operator choose the documents where the value of a field equals any value in the specified array.
  - db.student.find({ field: { $in: [<value1>, <value2>, ...] } })
  - db.student.find({"rollno":{$in:[10,12]}})
- $nin
  - The $nin operator chooses the documents where the field value is not in the specified array or does not exist

Comparison Operators

- The **$and** operator works as a logical AND operation on an array. The array should be of one or more expressions and chooses the documents that satisfy all the expressions in the array.
- **Syntax:**
  - { $and: [{ <exp1> }, { <exp2> }, ...]}
- **Example**:
  - db.student.find({ $and: [{ name: { $eq: "Anil" } }, { rollno: { $eq: 12 } }] })

Comparison Operators

- $or
  - It works as a logical OR operation on an array of two or more expressions and chooses documents that meet the expectation at least one of the expressions.
  - { $or: [{ <exp_1> }, { <exp_2> }, ... , { <exp_n> }] }
  - Example:
  - db.products.find({ $or: [{ name: { $eq: "Pencil" } }, { price: { $eq: 4 } }], });

Comparison Operators

- $not
  - The $not operator works as a logical NOT on the specified expression and chooses the documents that are not related to the expression.
  - { field: { $not: { <operator-expression> } } }
  - Example:
  - db.products.find({ price: { $not: { $gt: 5 } } });

Comparison Operators

- $nor
  - The $nor operator works as logical NOR on an array of one or more query expression and chooses the documents that fail all the query expression in the array.
  - { $nor: [{ <expression1> } , { <expresion2> } , ...] }
  - Example:
  - db.products.find({ $nor: [{ name: { $eq: "Pen" } }, { price: { $eq: 10 } }] });

Update Document

- Two ways to update a document in a collection.
  1. updateOne() method
  2. save() method
- The updateOne() method is used when we need to update the values of an existing document.
- The save() method is used to replace the existing document with the document that has been passed in it.

Update Document

- Updating Document using updateOne() method
- Syntax:
  - db.collection_name.updateOne(criteria, update_data)
- Example:
  - db.student.updateOne({"name":“Anil"},{$set:{"name":“Anil Kumar"}})

Update Document

- Updating Document using save() method
- **Syntax**:
  - db.collection_name.save({_id:ObjectId(), new_document})
- **Example**:

```
db.student.save({
        _id: ObjectId("606aab7a8f714b1761930662"),
        name: "John",
        rollno: 20,
})
```

Note: To work with save() method we should know the unique _id field of that document.

Delete Document

- The remove() method is used for removing the documents from a collection in MongoDB.
- Syntax:
  - db.collection_name.remove(delete_criteria)
  - >db.student.remove({“rno”:100})

Delete Document

- Remove all Documents
  - To remove all the documents from a collection but does not want to remove the collection itself then use remove() method like this:
    - db.collection_name.remove({})

MongoDB Projection

- MongoDB Projection is used when we want to get the selected fields of the documents rather than all fields.
  - db.collection_name.find({},{field_key:1 or 0})
  - db.students.find({name:"kalyan"},{name:1})
  - need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.
  - When we set a field to 1 in Projection other fields are automatically set to 0
  - The vice versa is also true when we set few fields to 0, other fields set to 1 automatically

Sort()

- Using sort() method, you can sort the documents in ascending or descending order based on a particular field of document.
- Syntax of sort() method:
  - db.collection_name.find().sort({field_key:1 or -1})
  - 1 is for ascending order and -1 is for descending order.
  - db.student.find().sort({“name”:1})