Mongo db

Mongo db

- The most popular NoSQL database, is an open-source document-oriented database.
- The term ‘NoSQL’ means ‘non-relational’.
- It means that MongoDB isn’t based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data.
- This format of storage is called BSON (similar to JSON format).
- BSON-Binary encoded JavaScript object notation

Mongo db

- MongoDB is a document database. It stores data in a type of JSON format called BSON.

- MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

MongoDB

- **Windows:**
- Log in Admin
- mongod(minimize)
- Mongo(commands given here)
- **Linux:**
- mongo

Create databse

- MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.
- use DATABASE_NAME
- Eg:
- >use mydb
- switched to db mydb

- To check your currently selected database, use the command **db**
- >db
- Mydb
- If you want to check your databases list, use the command **show dbs**.
- >show dbs
- local 0.78125GB
- test 0.23012GB

CRUD operations:

- CRUD operations *create*, *read*, *update*, and *delete* documents.
- Create Operations
- Create or insert operations add new documents to a collection.
- If the collection does not currently exist, insert operations will create the collection.

Create Collection

- MongoDB commands are **case-sensitive**
- **db.createCollection(name, options)**
- **>db.createCollection(“mycollection”)**
- **To check created collections**
- **>show collections**

Insert document in MongoDB collection

- To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

>db.COLLECTION_NAME.insert(document)

- **The insertOne() method**
- If you need to insert only one document into a collection you can use this method.

>db.COLLECTION_NAME.insertOne(document)

Create or insert operation

- **db .collection.insertOne()** //Inserts a single document into a collection.

- The insertMany() method
- You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Insert many documents

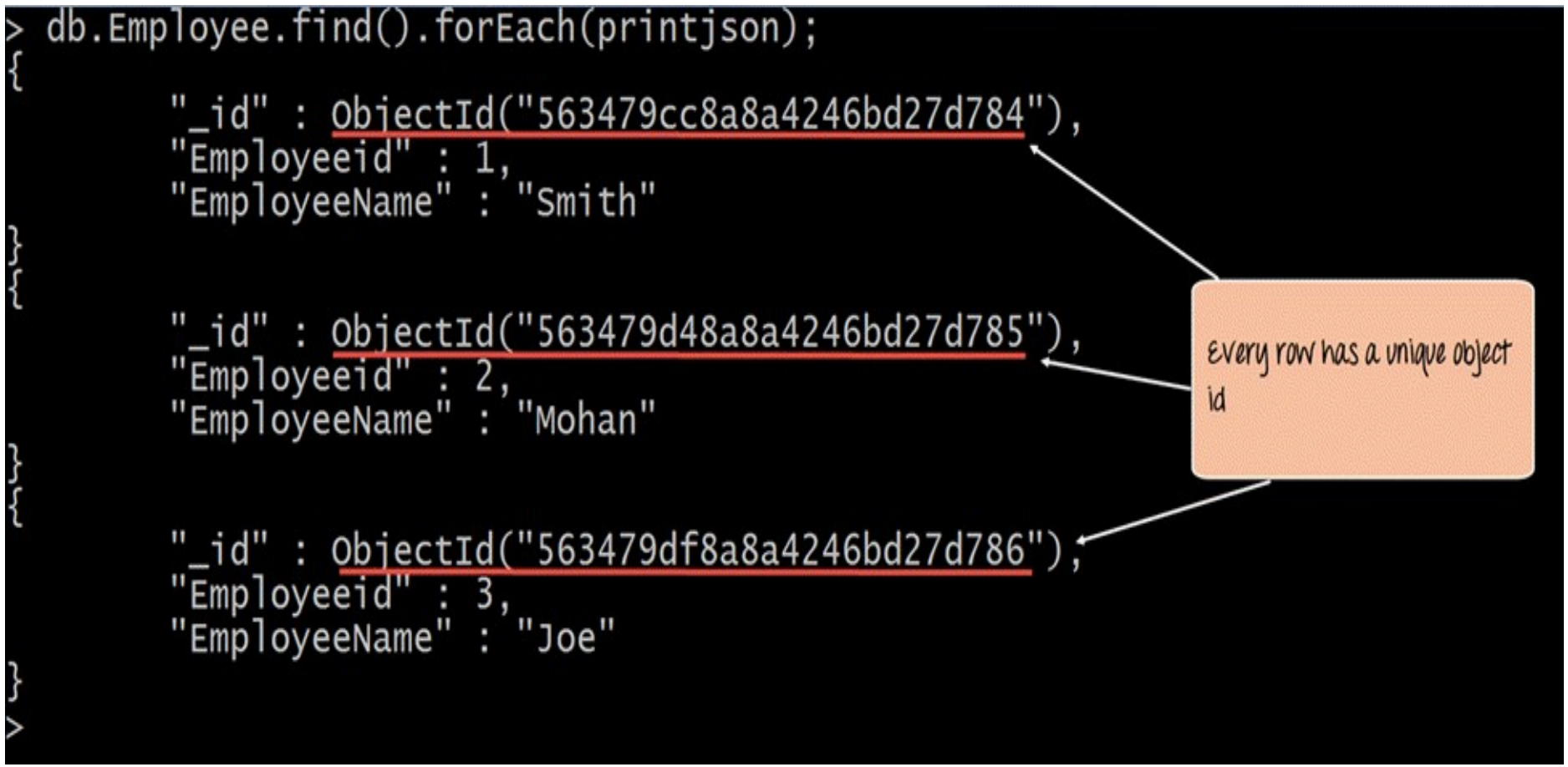
```
db.collection.insertMany(
[ <document 1> , <document 2>, ... ],
{
writeConcern: <document>,
ordered: <boolean>
}
)
```

Eg:

- db.student.insertMany([{name:"Ajay",age:20},
  {name:"Bina",age:24},
  {name:"Ram",age:23}])

```
> db.empDetails.insertMany(
        [
                {
                        First_Name: "Radhika",
                        Last_Name: "Sharma",
                        Date_Of_Birth: "1995-09-26",
                        e_mail: "radhika_sharma.123@gmail.com",
                        phone: "9000012345"
                },
                {
                        First_Name: "Rachel",
                        Last_Name: "Christopher",
                        Date_Of_Birth: "1990-02-16",
                        e_mail: "Rachel_Christopher.123@gmail.com",
                        phone: "9000054321"
                },
                {
                        First_Name: "Fathima",
                        Last_Name: "Sheik",
                        Date_Of_Birth: "1990-02-16",
                        e_mail: "Fathima_Sheik.123@gmail.com",
                        phone: "9000054321"
                }
```

- In MongoDB, each document stored in a collection requires a unique _id field that acts as a primary key.
- If an inserted document omits the _id field, the MongoDB driver automatically generates an ObjectId for the _id field.
- By default when inserting documents in the collection, if you don’t add a field name with the _id in the field name, then MongoDB will automatically add an Object id field as shown below


> db.Employee.find().forEach(printjson);
{
"_id" : ObjectId("563479cc8a8a4246bd27d784"),
"Employeeid" : 1,
"EmployeeName" : "Smith"
}
{
"_id" : ObjectId("563479d48a8a4246bd27d785"),
"Employeeid" : 2,
"EmployeeName" : "Mohan"
}
{
"_id" : ObjectId("563479df8a8a4246bd27d786"),
"Employeeid" : 3,
"EmployeeName" : "Joe"
}
Every row has a unique object id

- If you want to ensure that MongoDB does not create the _id Field when the collection is created and if you want to specify your own id as the _id of the collection, then you need to explicitly define this while creating the collection.
- When explicitly creating an id field, it needs to be created with _id in its name.

- db.Employee.insert({_id:10, "EmployeeName" : "Smith"})

Read Operations

- Read operations retrieve documents from a collection; i.e. query a collection for documents.
- MongoDB provides the following methods to read documents from a collection:
- The find() method with no parameters returns all documents from a collection and returns all fields for the documents.
- **db.collection.find(query, projection, options)**
- db.collection.find(<query>).pretty()//display the documents in specific format.

Eg:

- By default, db .collection.find() returns data in a dense format:
- db.books.find()
- { "_id" : ObjectId("54f612b6029b47909a90ce8d"), "title" : "A Tale of Two Cities", "text" : "It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness...", "authorship" : "Charles Dickens" }

Eg:

```
b.books.find().pretty()

{

"_id" : ObjectId("54f612b6029b47909a90ce8d"),

"title" : "A Tale of Two Cities",

"text" : "It was the best of times, it was the worst of times, it was the age of
wisdom, it was the age of foolishness...",


"authorship" : "Charles Dickens"

}
```

The examples in this section use documents from the bios collection where the documents generally have the form: db.bios.find()

```
{
"_id" : <value>,
"name" : { "first" : <string>, "last" : <string> }, // embedded document
"birth" : <ISODate>,
"death" : <ISODate>,
"contribs" : [ <string>, ... ], // Array of Strings
"awards" : [
{ "award" : <string>, year: <number>, by: <string> } // Array of embedded documents
...
]
}
```

Query for Equality

- db.collection name.find({ _id: 5 })
- db. collection name.find({ **"name.last"**: **"Hopper"** })

Query for Equality

- The following operation returns documents in the bios collection where _id equals 5:
- db.bios.find({ _id: 5 })
- The following operation returns documents in the bios collection where the field last in the name embedded document equals "Hopper":
- db.bios.find({ **"name.last"**: **"Hopper"** })

Query Using Operators

- The following operation uses the **$in** operator to return documents in the bios collection where _id equals either 5 or ObjectId("507c35dd8fada716c89d0013"):

```
db.bios.find(

{ _id: { $in: [ 5,
ObjectId("507c35dd8fada716c89d0013") ] } }
)
```

- The following operation uses the $gt operator returns all the documents from the bios collection where birth is greater than new Date('1950-01-01’):
- db.bios.find({ birth: { $gt: new Date('**1950-01-01**') } })
- The following operation uses the **$regex** operator to return do d starts wi

```
db.bios.find(

{ "name.last": { $regex: /^N/ } }

)
```

Query for Ranges

- Combine comparison operators to specify ranges for a field. The following operation returns from the bios collection documents where birth is between new Date('1940-01-01') and new Date('1960-01-01') (exclusive):

  db.bios.find({ birth: { $gt: new Date('1940-01-01'), $lt: new Date('1960-01-01') } })

- Query for Multiple Conditions
- The following operation returns all the documents from the bios collection where birth field is greater than new Date('1950-01-01') and death field does not exists:
- db.bios.find({

    birth: { $gt: new Date('1920-01-01') },

    death: { $exists: false }

- })

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{$eg; <value>}}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt: <value>}}	db.mycol.find({"likes": {$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte: <value>}}	db.mycol.find({"likes": {$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt: <value>}}	db.mycol.find({"likes": {$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte: <value>}}	db.mycol.find({"likes": {$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne: <value>}}	db.mycol.find({"likes": {$ne:50}}).pretty()	where likes != 50
Values in an array	{<key>:{$in: [<value1>, <value2>,..... <valueN>]}}	db.mycol.find({"name":{$in: ["Raj", "Ram", "Raghu"]}}).pretty()	Where name matches any of the value in :["Raj", "Ram", "Raghu"]

References:

- https://www.mongodb.com/docs/manual/reference/method/db.collection.find/#mongodb-method-db.collection.find

- https://www.mongodb.com/docs/manual/introduction/

- Video
- https://www.youtube.com/watch?v=v6Xmydb7u4Y
- https://www.youtube.com/watch?v=eOJeZ4ClINI

Update

MongoDB Update() Method

- The update() method updates the values in the existing document.

Syntax

- The basic syntax of update() method is as follows −

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}})
```

- By default, MongoDB will update only a single document.
- To update multiple documents, you need to set a parameter 'multi' to true.

- >db.mycol.update({'title':'MongoDB Overview'}, {$set: {'title':'New MongoDB Tutorial'}},{multi:true})

- MongoDB findOneAndUpdate() method
- The findOneAndUpdate() method updates the values in the existing document.

- The basic syntax of findOneAndUpdate() method is as follows −

- >db.COLLECTION_NAME.findOneAndUpdate(SELECTIOIN_CRITERIA, UPDATED_DATA)

- update the age and email values of the document with name 'Radhika'.

```
> db.empDetails.findOneAndUpdate(
    {First_Name: 'Radhika'},
    { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
```

Deletion

- To delete multiple documents, use
- db.collection.deleteMany().

- To delete a single document, use
- db.collection.deleteOne().

- db.movies.deleteMany({ title: "Titanic" })
- db.movies.deleteOne({ cast: "Brad Pitt" })