NoSQL (Any one from MongoDB/CASSANDRA)

By Mrs. Ankita S. Joshi

NoSQL Databases

- NoSQL Database is used to refer a non-SQL or non relational database.
- It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

History

- In the early 1970, Flat File Systems are used. Data were stored in flat files and the biggest problems with flat files are each company implement their own flat files and there are no standards.
- It is very difficult to store data in the files, retrieve data from files because there is no standard way to store data.
- Then the relational database was created by E.F. Codd and these databases answered the question of having no standard way to store data.
- But later relational database also get a problem that it could not handle big data, due to this problem there was a need of database which can handle every types of problems then NoSQL database was developed.

MongoDB

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
- In simple words, you can say that - Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.
- "MongoDB is a scalable, open source, high performance, document-oriented database." - 10gen
- MongoDB was designed to work with commodity servers. Now it is used by the company of all sizes, across all industry.

History

- The initial development of MongoDB began in 2007 when the company was building a platform as a service similar to window azure.
- MongoDB was developed by a NewYork based organization named 10gen which is now known as MongoDB Inc. It was initially developed as a PAAS (Platform as a Service). Later in 2009, it is introduced in the market as an open source database server that was maintained and supported by MongoDB Inc.
- The first ready production of MongoDB has been considered from version 1.4 which was released in March 2010.

MongoDB advantages over RDBMS

- In recent days, MongoDB is a new and popularly used database. It is a document based, non relational database provider.
- Although it is 100 times faster than the traditional database but it is early to say that it will broadly replace the traditional RDBMS. But it may be very useful in term to gain performance and scalability.
- A Relational database has a typical schema design that shows number of tables and the relationship between these tables, while in MongoDB there is no concept of relationship.

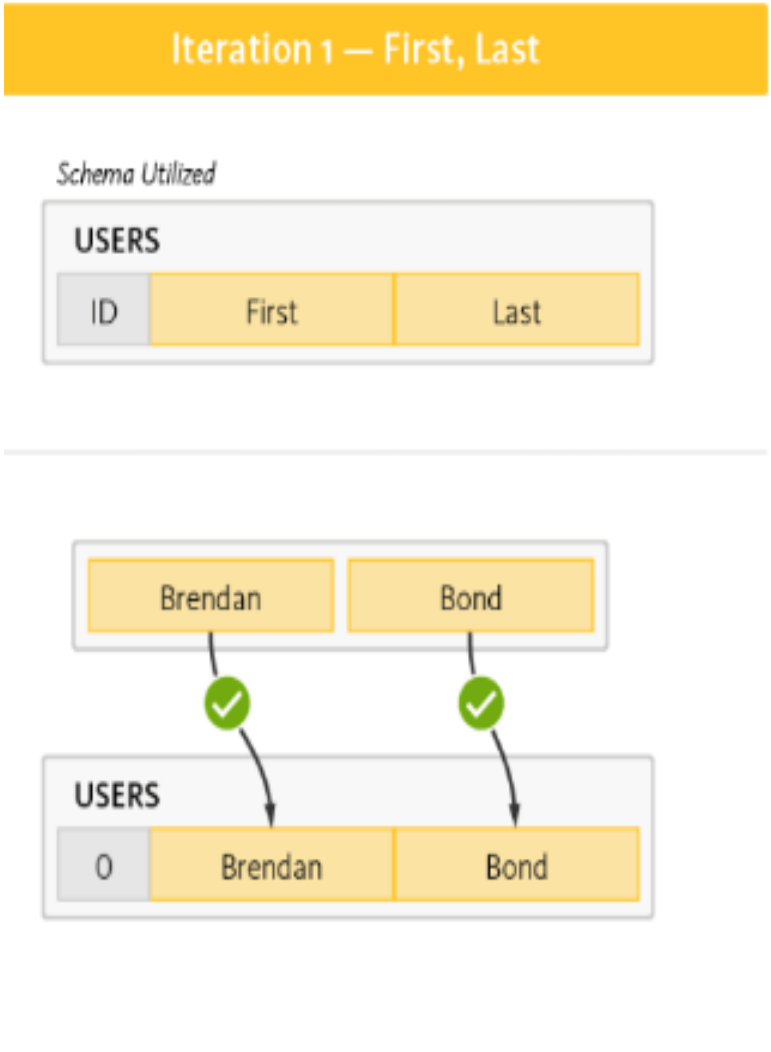
MongoDB Advantages

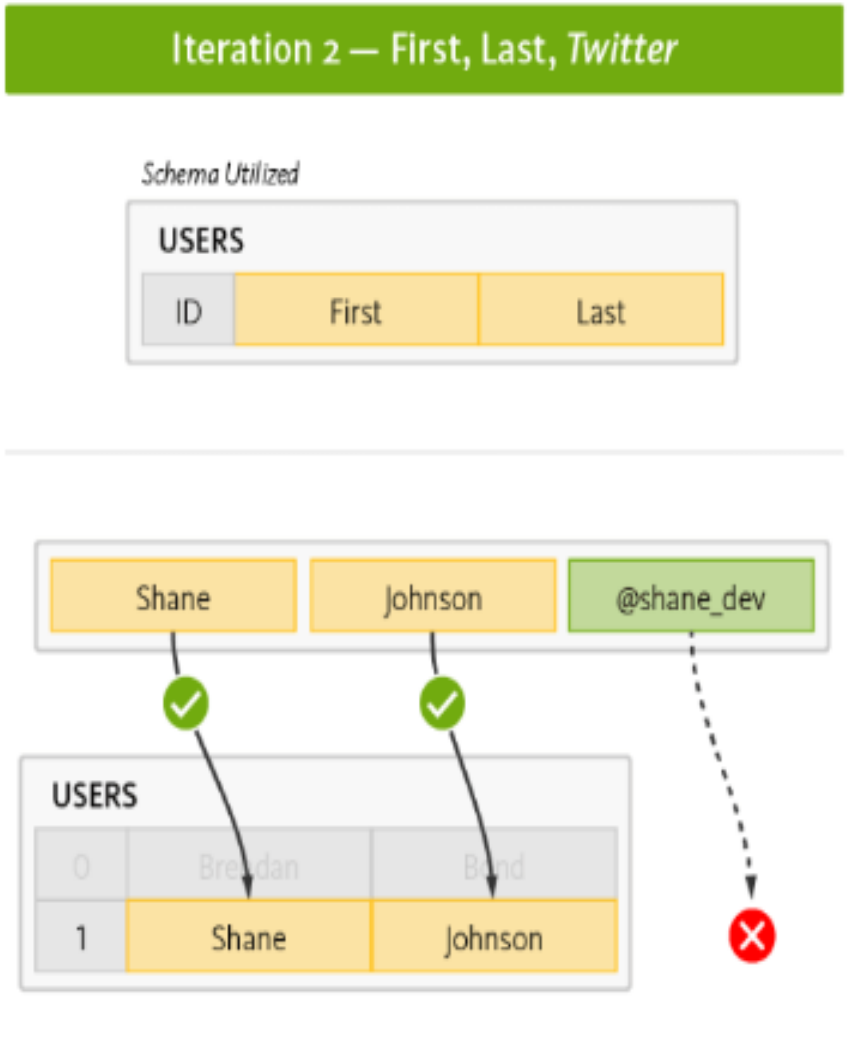
- MongoDB is schema less. It is a document database in which one collection holds different documents.
- There may be difference between number of fields, content and size of the document from one to other.
- Structure of a single object is clear in MongoDB.
- There are no complex joins in MongoDB.
- MongoDB provides the facility of deep query because it supports a powerful dynamic query on documents.
- It is very easy to scale.
- It uses internal memory for storing working sets and this is the reason of its fast access.

Performance analysis of MongoDB and RDBMS

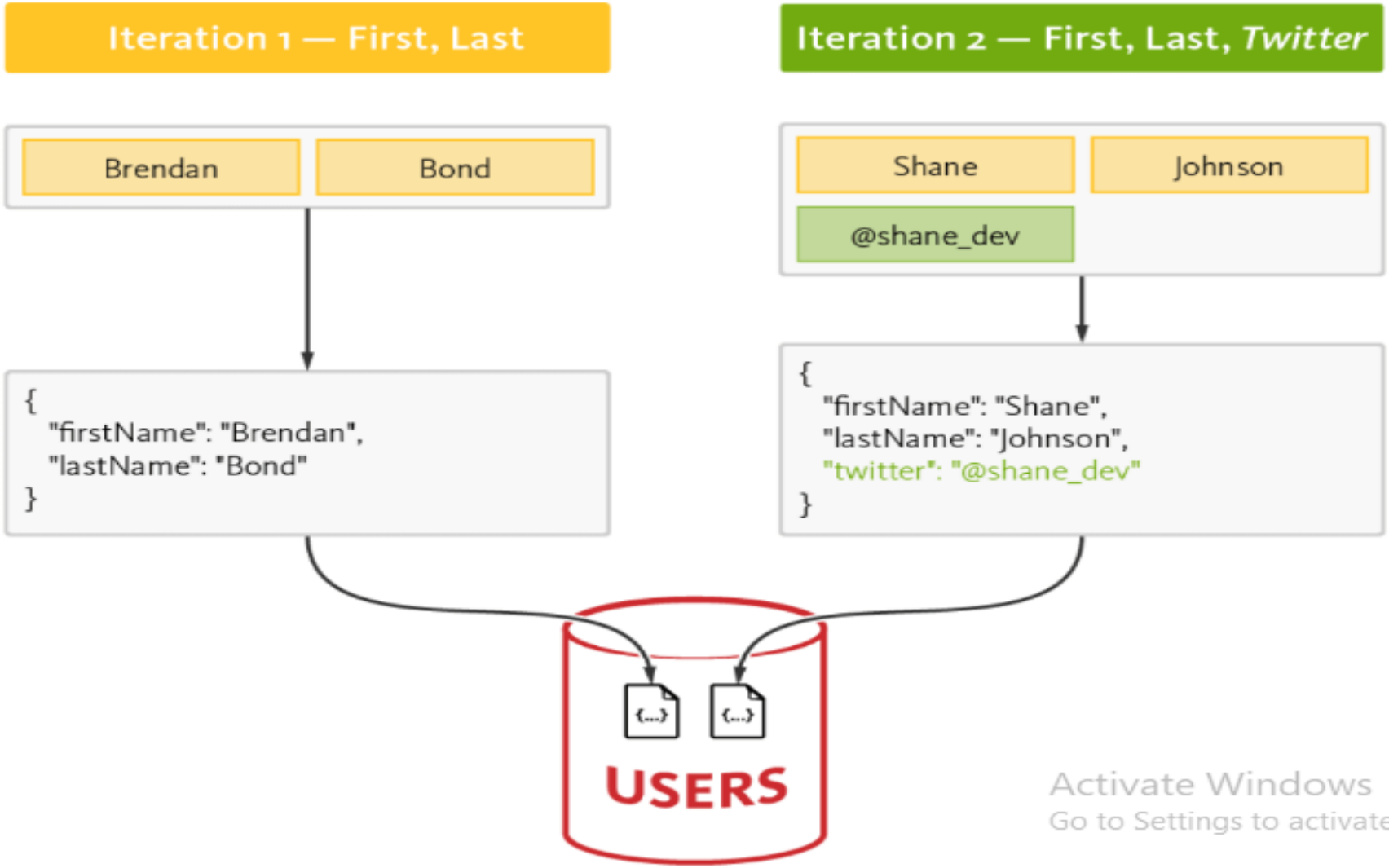
- In relational database (RDBMS) tables are using as storing elements, while in MongoDB collection is used.
- In the RDBMS, we have multiple schema and in each schema we create tables to store data while, MongoDB is a document oriented database in which data is written in BSON format which is a JSON like format.
- MongoDB is almost 100 times faster than traditional database systems.

RDBMS


Iteration 1 — First, Last
Schema Utilized
USERS
ID
First
Last
Brendan
Bond
USERS
0
Brendan
Bond


Iteration 2 — First, Last, Twitter
Schema Utilized
USERS
ID
First
Last
Shane
Johnson
@shane_dev
USERS
1
Shane
Johnson

MongoDB



MongoDB - Data Modelling

- Data in MongoDB has a flexible schema.documents in the same collection. They do not need to have the same set of fields or structure Common fields in a collection's documents may hold different types of data.
- MongoDB provides two types of data models: — Embedded data model and Normalized data model. Based on the requirement, you can use either of the models while preparing your document.

Data Model types

- Embedded Data Model
  - In this model, you can have (embed) all the related data in a single document, it is also known as de-normalized data model.
- Normalized Data Model
  - In this model, you can refer the sub documents in the original document, using references.

Embedded Data Model

```
{
_id: , Emp_ID: "10025AE336"
Personal_details:
{
First_Name: "Radhika",
Last_Name: "Sharma",
Date_Of_Birth: "1995-09-26"
},
Contact:
{
 e-mail:
"radhika_sharma.123@gmail.com
",
phone: "9848022338"
},
Address:
{
city: "Hyderabad",
Area: "Madapur",
State: "Telangana"
}
}
```

Normalized Data Model

Employee:

{

_id: <ObjectId101>,

Emp_ID: "10025AE336"

}

Personal_details:

{

_id: <ObjectId102>,

empDocID: "

ObjectId101",

First_Name: "Radhika",

Last_Name: "Sharma",

Date_Of_Birth: "1995-09-26"

}

Normalized Data Model

Contact:

{

_id: <ObjectId103>,
empDocID: "
ObjectId101",

e-mail:
"radhika_sharma.123@g
mail.com",

phone: "9848022338"

}

Address:

{

_id: <ObjectId104>,
empDocID: "
ObjectId101",

city: "Hyderabad",

Area: "Madapur",

State: "Telangana"

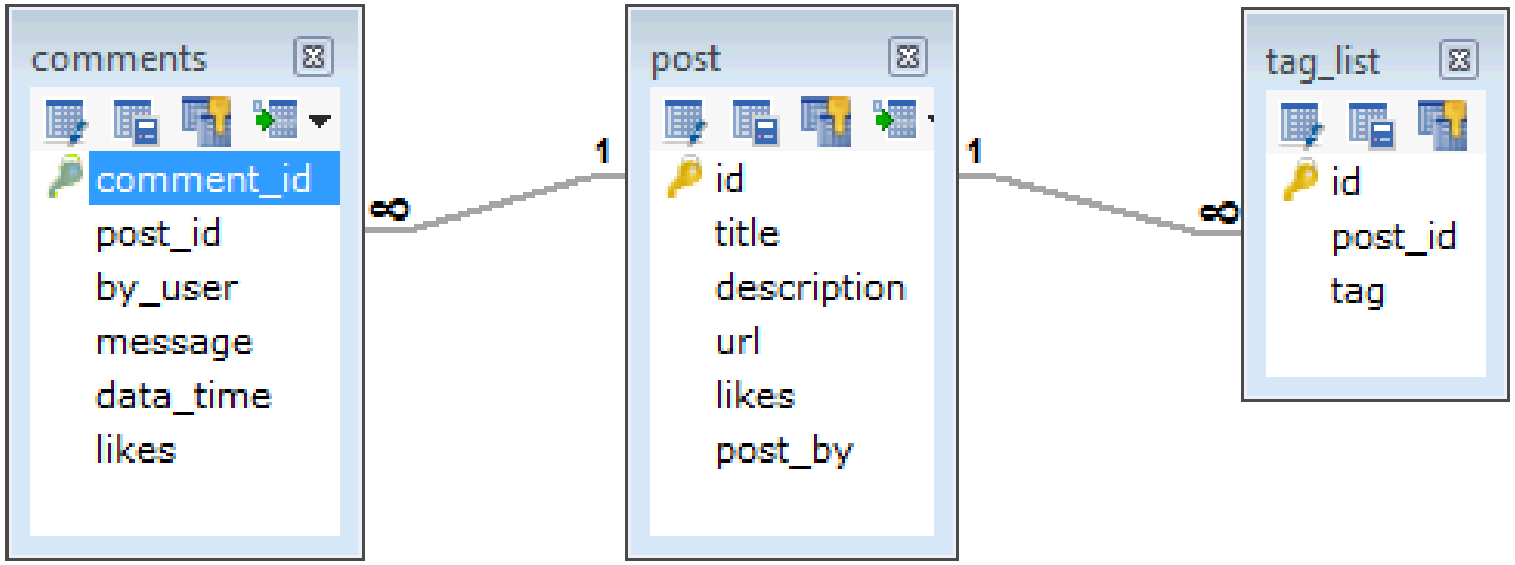
}

Example

- Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.
- Every post has the unique title, description and url.
- Every post can have one or more tags.
- Every post has the name of its publisher and total number of likes.
- Every post has comments given by users along with their name, message, data-time and likes.
- On each post, there can be zero or more comments.

RDBMS

- In RDBMS schema, design for above requirements will have minimum three tables.



mongodb

```
{
_id: POST_ID
title: TITLE_OF_POST,
description:
POST_DESCRIPTION,
by: POST_BY,
url: URL_OF_POST,
tags: [TAG1, TAG2, TAG3],
likes: TOTAL_LIKES,
```

```
comments:
    [
    {
    user:'COMMENT_BY',
    message: TEXT,
    dateCreated: DATE_TIME,
    like: LIKES
    },
    {
    user:'COMMENT_BY',
    message: TEXT,
    dateCreated: DATE_TIME,
    like: LIKES
    }
     ]
    }
```

JSON Object Literals

- This is a JSON string:
  - {"name":"John", "age":30, "car":null}
- Inside the JSON string there is a JSON object literal:
  - {"name":"John", "age":30, "car":null}
- JSON object literals are surrounded by curly braces {}.
- JSON object literals contains key/value pairs.
- Keys and values are separated by a colon.

JSON Object Literals

- Keys must be strings, and values must be a valid JSON data type:
  - string
  - number
  - Object
  - array
  - boolean
  - null
- Each key/value pair is separated by a comma.

MongoDB Create Database

- **Use Database method:**
  - There is no create database command in MongoDB. Actually, MongoDB do not provide any command to create database.
  - Here, in MongoDB you don't need to create a database manually because MongoDB will create it automatically when you save the value into the defined collection at first time.
- **Syntax:**
  - use DATABASE_NAME

MongoDB Drop Database

- The dropDatabase command is used to drop a database. It also deletes the associated data files. It operates on the current database.
- **Syntax:**
  - db.dropDatabase()
- This syntax will delete the selected database. In the case you have not selected any database, it will delete default "test" database.

MongoDB Create Collection

- In MongoDB, db.createCollection(name, options) is used to create collection. But usually you don't need to create collection. MongoDB creates collection automatically when you insert some documents.
- **Syntax:**
  - db.createCollection(**name**, options)
- **Name:** is a string type, specifies the name of the collection to be created.
- **Options:** is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.

Options

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

MongoDB Create Collection

- The following example shows the syntax of **createCollection()** method with few important options −
- > db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 })

createCollection()

- Basic syntax of **createCollection()** method without options is as follows −
- >use test
- switched to db test
  >db.createCollection("mycollection")
- { "ok" : 1 }
- You can check the created collection by using the command **show collections**.
- >show collections
- mycollection
- system.indexes

MongoDB - Drop Collection

- The drop() Method
  - MongoDB's **db.collection.drop()** is used to drop a collection from the database.
- Syntax
  - Basic syntax of **drop()** command is as follows –
  - db.COLLECTION_NAME.drop()
- Example
  - db.mycollection.drop()
  - true

Insert data

- The insertOne() method
  - If you need to insert only one document into a collection you can use this method.
- Syntax
  - The basic syntax of insert() command is as follows –
  - >db.COLLECTION_NAME.insertOne(document)

Example

- db.createCollection("empDetails")
- { "ok" : 1 }
- > db.empDetails.insertOne({ First_Name: "Radhika", Last_Name: "Sharma", Date_Of_Birth: "1995-09-26", e_mail: "radhika_sharma.123@gmail.com", phone: "9848022338" })
- { "acknowledged" : true, "insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
- }

Insert Data

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

```
> 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"
                }
        ]
)
```

The find() Method

- To query data from MongoDB collection, you need to use MongoDB's **find()** method.
- Syntax
  - The basic syntax of **find()** method is as follows −
  - >db.COLLECTION_NAME.find()
  - **find()** method will display all the documents in a non-structured way.
- The pretty() Method
  - To display the results in a formatted way, you can use pretty() method.
- Syntax
  - >db.COLLECTION_NAME.find().pretty()

The find() Method

- The findOne() method
  - Apart from the find() method, there is **findOne()** method, that returns only one document.
- Syntax
  - >db.COLLECTIONNAME.findOne()

The find() Method

- Syntax
  - The basic syntax of **find()** method with projection is as follows −
  - >db.COLLECTION_NAME.find({},{KEY:1})
- Example:
  - >db.mycol.find({},{"title":1,_id:0})
    {"title":"MongoDB Overview"}
    {"title":"NoSQL Overview"}
    {"title":"Tutorials Point Overview"}

Limit Records

- To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed.
- Syntax
  - The basic syntax of **limit()** method is as follows −
- >db.COLLECTION_NAME.find().limit(NUMBER)

- Apart from limit() method, there is one more method **skip()** which also accepts number type argument and is used to skip the number of documents.
- Syntax
  - The basic syntax of **skip()** method is as follows −
  - >db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)

The sort() Method

- To sort documents in MongoDB, you need to use **sort()** method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.
- Syntax
  - The basic syntax of **sort()** method is as follows −
  - >db.COLLECTION_NAME.find().sort({KEY:1})

  Example:
  db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})

RDBMS Where Clause Equivalents in MongoDB

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{$eg;<value>}}	db.mycol.find({"by":{$eg:" 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

RDBMS Where Clause Equivalents in MongoDB

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"]
Values not in an array	{<key>:{$nin:<value>}}	db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty()	Where name values is not in the array :["Ramu", "Raghav"] or,

AND and OR in MongoDB

- Syntax
  - To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND –
  - >db.mycol.find({ $and: [{<key1>:<value1>}, { <key2>:<value2>}] })

- OR in MongoDB
- Syntax
  - To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** −
  - >db.mycol.find({ $or: [{key1: value1}, {key2:value2}] }).pretty()

NOR keyword

- Syntax
  - To query documents based on the NOR condition, you need to use $nor keyword. Following is the basic syntax of **NOR** –
  - >db.COLLECTION_NAME.find({ $nor: [{key1: value1}, {key2:value2}] })

NOT in MongoDB

- Syntax
  - To query documents based on the NOT condition, you need to use $not keyword following is the basic syntax of **NOT** −
  - >db.COLLECTION_NAME.find({ $NOT: [{key1: value1}, {key2:value2}] }).pretty()

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)
- 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 Save() Method

- The **save()** method replaces the existing document with the new document passed in the save() method.
- Syntax
  - The basic syntax of MongoDB **save()** method is shown below −
  - >db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})

MongoDB findOneAndUpdate() method

- The **findOneAndUpdate()** method updates the values in the existing document.
- Syntax
  - The basic syntax of **findOneAndUpdate()** method is as follows −
  - >db.COLLECTION_NAME.findOneAndUpdate(SELECTIOIN_CRITERIA, UPDATED_DATA)
  - > db.empDetails.findOneAndUpdate({First_Name: 'Radhika'}, { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}

MongoDB updateMany() method

- The updateMany() method updates all the documents that matches the given filter.
- Syntax
  - The basic syntax of updateMany() method is as follows −
  - >db.COLLECTION_NAME.updateMany(<filter>, <update>)
  - Example
  - > db.empDetails.updateMany({Age:{ $gt: "25" }}, { $set: { Age: '00'}})

The remove() Method

- MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.
  - **deletion criteria** − (Optional) deletion criteria according to documents will be removed.
  - **justOne** − (Optional) if set to true or 1, then remove only one document.
- Syntax
  - Basic syntax of **remove()** method is as follows −
  - >db.COLLECTION_NAME.remove(DELLETION_CRITTERIA,justOne)

Remove method

- Remove Only One
- If there are multiple records and you want to delete only the first record, then set **justOne** parameter in **remove()** method.
  - >db.COLLECTION_NAME.remove(DELETION_CRITERIA, 1)
- Remove All Documents
- If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**
  - > db.mycol.remove({})

Aggregation

- Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- In SQL count(*) and with group by is an equivalent of MongoDB aggregation.

The aggregate() Method

- For the aggregation in MongoDB, you should use **aggregate()** method.
- Syntax
  - Basic syntax of **aggregate()** method is as follows −
  - >db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
	Gets the maximum of the corresponding	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max :

References

- https://www.couchbase.com/resources/why-nosql
- https://www.javatpoint.com/mongodb-create-database