Complete notes on

MongoDB

Written By:-
Dipti Gaydhane
(Software Engineer)

INDEX

MongoDB Tutorial

11) MongoDB Aggregation $group

12) MongoDB Aggregation $Match

13) MongoDB Mongosh Insert

14) MongoDB Mongosh Find

15) MongoDB Mongosh Update

16) MongoDB Query Operators

17) MongoDB Aggregation pipelines

18) MongoDB Data Api

19) MongoDB Aggregation $Lookup

20) MongoDB Aggregation $project

21) MongoDB Aggregation $add

22) MongoDB Drivers

Mongo-DB

MongoDB Getting Started :

MongoDB

MongoDB is a document database and can be installed locally or hosted in the Cloud.

SQL vs Document Database

SQL databases are considered relational databases. They store related data in seperate tables. They when data is queried from multiple tables to join the data back together.

MongoDB is a document database which is often referred to as a non-relational database.

This does not mean that relational data cannot be stored in document databases. It means that relational data is stored differently. A better way to refer to it is as a non-tabular database.

You still have multiple groups of data too. In MongoDB, instead of tables these are called Collections.

Local vs Cloud Database

MongoDB can be installed locally, which will allow you to host your own MongoDB server on your hardware.

This requires you to manage your server, upgrades, and any other maintenance.

Creating a Cluster

After you have created your account, set up a free "Shared Cluster" then choose your preferred cloud provider and region.

By default, MongoDB Atlas is completely

locked down and has no external access.

You will need to set up a user and add your Ip address to the list of allowed Ip addresses.

Under "Database Access", create a new user and keep track of the username and password.

Next, under "Network Access", add your current Ip address To allow access from your computer.

Install MongoDB shell (mongosh):

There are many ways to connect to your MongoDB database.

We will start by using the MongoDB shell, Mongosh.

Use the official instructions to install Mongosh on your Operating System.

To verify that it has been installed properly

Open your terminal and type.

```
Mongosh -- version
```

You should see that the latest version is installed.

The version used in this tutorial is v1.3.1.

Connect to the database

To connect to your database, you will need your database specific connection string.

In the MongoDB Atlas dashboard, under "Databases", click the "Connect" button for your cluster.

Example ↓

your connection string should look similar to this

```
Mongosh
"Mongodb+srv://cluster0.ex4ht.mongodb.
```

```
net/myFirstDatabase" --apiversion
1 --username YOUR_USER_NAME
```

MongoDB Query API :

The MongoDB Query API is the way you will interact with your data.

The MongoDB Query API can be used two ways :

- CRUD operations
- Aggregation pipelines

MongoDB Query API Uses

You can use the MongoDB Query API to perform.

- Adhoc queries with Mongosh, Compass, VS Code, or a MongoDB Driver for the programming language you use.

- Data transformations using aggregation pipelines.
- Document join support to combine data from different collections.
- Graph and geospatial queries.
- Full-text search
- Indexing to improve MongoDB query performance.
- Time Series analysis.

MongoDB mongosh Create Database :

After connecting to your database using Mongosh, you can see which database you are using by typing `db` in your terminal.

If you have used the connection string provided from the MongoDB Atlas dashboard, you should be connected to the `MyFirstDatabase` database.

Show all databases

To see all available databases, in your terminal type `show dbs`.

Change or Create a Database

Example,

Create a new database called "blog":

```
Use blog
```

MongoDB Mongosh Create Collection:

Create collection using Mongosh

There are 2 ways to create collection

Method 1

You can create a collection using the `CreateCollection()` database Method.

Example:

```
db.createCollection("posts")
```

Method 2

You can also create a collection during the insert process.

Example:

We are here assuming Object is a valid Javascript Object containing post data.

```
db.posts.insertOne(Object)
```

This will create the "posts" collection if it does not already exist

MongoDB Update Operators :

There are many update operators that can be used during document updates.

Fields

The following operators can be used during document updates.

- $currentDate : Sets the field value of the current date
- $inc : increments the field value
- $rename : Renames the field.
- $set : Sets the value of a field
- $unset : Removes the field from the document.

Array

The following operators assists with updating arrays.

- $addToSet : Adds distinct elements to an array
- $pop : Removes the first of last elements of array.
- $pull : Removes all elements from an array that match the query.
- $push : Adds an element to an array

MongoDB Mongosh Delete :

Delete Documents

We can delete documents by using the

Methods deleteOne() or deleteMany().

These Methods accept a query object. The Matching documents will be deleted.

deleteOne()

The deleteOne() Method will delete the first document that Matches the query provided.

Example↓

```
db.posts.deleteOne({ title: "post Title 5" })
```

deleteMany()

The deleteMany() Method will delete all documents that Match the query provided.

Example↓

```
db.posts.deleteMany {{ category: "Technology" }}
```

MongoDB Aggregation $count :

This aggregation stage counts the total amount of document passed from the previous stage.

Example :-

In this example we are using 'sample_restaurants' database loaded from our sample data in the Intro to Aggregation

```
db.restaurants.aggregate
([
  {
    $match: { cuisine: "Chinese" }
  },
  {
    $count: "totalchinese"
  }
])
```

MongoDB Aggregation $sort :

This aggregation stage group sorts all documents in the specified sort order.

ocean group
PAGE:
DATE: / /

Example:

```
db.listingsAndReviews.aggregate
([
  {
    $sort  { "accommodates": -1 }
  },
  {
    $project:
    {
      "name": 1,
      "accommodates": 1
    }
  },
  {
    $limit: 5
  }
])
```

InsertMany()

To insert multiple documents at once, use the insertMany() Method.
This Method inserts an array of Objects into the database.

Example:

```
db.posts.insertMany([
  {
    title: "post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    body: "Body of post."
    likes: 4,
    tags: ["news", "events"],
    date: Date()
  }
])
```

MongoDB Aggregation $limit :

This Aggregation Stage limits the number of documents passed to the next Stage.

Example → `db.movies.aggregate([{ $limit: 1 }])`

MongoDB Aggregation $group :

This aggregation stage group documents by the unique _id expression provided.

Example↓

```
db.listingsAndReviews.aggregate(
    [{$group: { _id: "$property_type" }}]
)
```

UpdateMany()

The UpdateMany() Method will update all documents that Match the provided query.

Example↓

Update likes on all documents by 1. for this we will use the $inc (increment) Operator.

```
db.posts.updateMany({}, { $inc: { likes: 1 } })
```

MongoDB Aggregation $match :

This aggregation stage behaves like a find. It will filter documents that match the query provided.

Example ↓

In this example, we are using the "sample_airbnb" database loaded from our sample data

```
db.listingsAndReviews.aggregate([
  { $match : { property_type : "House" } },
  { $limit : 2 },
  { $project : {
    "name" : 1,
    "bedrooms" : 1,
    "price" : 1
  }}
])
```

MongoDB mongosh Insert :

Insert Documents

There are 2 Methods to insert document

insertOne()

To insert a single document, use the insertOne() Method. This Method inserts a single object into the database.

Example ↓

```
db.posts.insertOne({
  title: "post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

insertMany()

To insert multiple documents at once, use the insertMany() Method. This Method inserts an array of objects into the database.

Example ↓

```
db.posts.insertMany([
```

```
    {
  title: "post Title 2",
  body: "Body of post.",
  category: "Event",
  likes: 2,
  tags: ["news", "events"],
  date: Date()
 },
 {
  title: "post Title 4",
  body: "Body of post.",
  category: "Event",
  likes: 4,
  tags: ["news", "events"],
  date: Date()
 }
])
```

MongoDB Mongosh Find :

Find Data

There are two Methods to find and select data from a MongoDB Collection, find() and findOne()

Example ↓

```
db.posts.find()
```

Findone()

To select only one document, we can use the findone() Method.
This Method accepts a query object. If left empty, it will return the first document it finds.

Example ↓

```
db.posts.findOne()
```

Querying Data

To query, or filter, data we can include a query in our find() or findOne() Methods.

Example ↓

```
db.posts.find ({ category : "News"})
```

Projection

Both find Methods accept a second parameter called projection.

This parameter is an object that describes which fields to include in the results.

Example:

This example will only display the title and date fields in the results.

```
db.posts.find({}, {title: 1, date: 1})
```

MongoDB Mongosh Update :

Update Document

To update an existing document we can use the updateOne() or updateMany() Methods.
The first parameter is a query object to define which documents or documents should updated.
The second parameter is an object defining the updated data.

updateOne()

updateOne() Method will update the first document that is found matching the provided query Let's see what the 'like' count for the post with the title of 'post' Title 1:

Example:

```
db.posts.find( { title : "post Title 1" } )
```

Example:

```
db.posts.updateone ( { title : "post Title 1" } , { $set :
                                { likes : 2 } } )
```

Example:

```
db.posts.find ( { title : "post Title 1" } )
```

updateMany()

updateMany() Method will update all documents that match the provided query

Example:

Update likes on all documents by 1 for this we will use the $inc (increment) operator.

```
db.posts.updateMany({}, { likes: 1 } })
```

MongoDB Query Operators

Comparison

- $eq : value are equal
- $ne : values are not equal
- $gt : value is greater than another value
- $gte : value is greater than or equal to another value.
- $lt : value is less than another value.
- $lte : value is less than or equal to another value.
- $in : value is matched within an array.

Logical

- $and : Returns documents where both queries match
- $or : Returns documents where either query matches
- $nor : Returns documents where both queries fail to match.
- $not : Returns documents where the query does not match.

Evalution

- $regex : Allows use of regular expressions when evaluting field values.
- $where : Uses a Javascript expression to Match documents.
- $text : performs a text search.

MongoDB Aggregation pipelines :

Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more.

Aggregation pipelines can have one or more stages are important. Each stage acts upon the results of the previous stage.

Example :

```
db.posts.aggregate([
  // Stage 1: only find documents that have more
     than 1 like
  {
    $match: { likes: { $et: 1 } }
  }
```

```
// Stage 2: Group documents by category and sum
   each Categories likes
{
  $group: { _id: "$category", totallikes : {$sum:"$likes"}}
}
})
```

MongoDB Data API :

Read & Write With the MongoDB Data API

The MongoDB Data API is a pre-configured set of HTTPS endpoints that can be used to read and write data to a MongoDB Atlas database.

With the MongoDB data API, you can create, read, update, delete, or aggregate documents in a MongoDB Atlas database.

Cluster Configuration

In order to use the Data API, you must first enable the functionality from the Atlas UI.

From the MongoDB Atlas dashboard navigate to Data API in the left menu.

Select the data source you would like to enable the API on and click Enable the Data API.

Access Level

By default, no access is granted select the access level you'd like to grant the Data API. The choices are. No accesses, Read Only, Read and Write, or Custom Access.

Data API key

In order to authenticate with the data API, you must first Create a data API key.

Click Create API key. enter a name for the key then Click Generate API key.

Be Sure to Copy the API key and save it Some-Where Safe.

Sending a Data API Request

Now use the Data API to send a request to the database.

MongoDB Aggregation $lookup :

This aggregation stage performs a left outer join to a collection in the same database.

There are four required fields.

- **from** : The collection to use for lookup in the same database.
- **localfield** : The field in the primary collection that can be used as a unique identifier in the from collection.
- **foreignfield** : The field in the from collection that can be used as a unique identifier in the primary collection.
- **as** : The name of the new field that will contain the matching documents from the from collection.

Example ↓

```
db.comments.aggregate([
{
```

```
        $lookup: {
            from : "movies",
            localField : "movie_id",
            foreignfield : "_id"
            as : "movie_details",
        }
    }
        {
        $limit :1
        }
])
```

MongoDB Aggregation $project :

This is the same projection that is used with the find() Method.

Example:

In this example, we are using the "sample-restaraants" database loaded from our sample data in the Intro Aggregation.

```
db.restaurants.aggregate([
{
```

```
$project {
    "name": 1,
    "cuisine": 1,
    "address": 1
  }
},
{
  $limit: 5
}
])
```

MongoDB Schema Validation

By Default MongoDB has a flexible schema. This means that there is no strict schema validation set up initially.

Schema validation rules can be created in order to insure that all documents a collection share a similar structure.

Example:

```
db.createCollection("posts", {
  validator: [
    $jsonSchema: {
```

```
bsonType : " Object ",
required : [ " title ", " body " ],
properties : {
  title : {
  bsonType : " String ",
  description : " Title of post - Required."
  },
  date : {
    bsonType : " date ",
    description : " Must be a date - Optional."
    }
  }
}
]}
```

MongoDB Aggregation $add :

Aggregation $addFields

Example:

```
db.restaurants.aggregate ([
{
  $addFields : {
```

```
        avgGrade : { $avg: "$grades.score" }
    }
},
{
    $project: {
        "name": 1,
        "avgGrade": 1
    }
},
{
    $limit: 5
}
])
```

MongoDB Drivers :

The MongoDB shell (mongosh) is great, but generally you will need to use MongoDB in your application. To do this, MongoDB has many language drivers.

These are the current officially supported drivers :

- C
- C++
- C#

- Go
- Java
- Node.js
- PHP
- Python
- Ruby
- Rust
- Scala
- Swift

Indexing & Search

MongoDB Atlas comes with a full-text search engine that can be used to search for documents in a collection.
Atlas Search is powered by Apache Lucene.

Running Query

To use our search index, we will use the $search operator in our aggregation pipeline.

Example → db.movies.aggregate([

```
{
    $search: {
        index: "default",
```

```
        text : {
            query : "Star Wars",
            path : "title",
        }
    },
},
{
    $project : {
        title : 1,
        year : 1,
    }
}
])
```

MongoDB Charts

MongoDB charts lets you visualize your data in a simple, intuitive way.

MongoDB charts Setup

From the MongoDB Atlas dashboard, go to the charts tab.
If you've never used charts before, click the Activate Now. button. This will take about 1 Minute to complete.

you'll see new dashboard. click the dashboard name to open it.

Creating a chart

Create a new chart by clicking the Add chart button.
Visually creating a chart is initiative. Select the data sources that you want to use.

MongoDB Node.js Database Interaction

Node.js Database Interaction

The 'sample_infix' database loaded from our sample data in the intro a aggregations.

MongoDB Node.js Driver Installation

```
npm install mongodb
```

We can now use this package to connect to MongoDB database.

Create an index.js file in your project dictionary

index.js

```
const { MongoClient } = require ('mongodb');
```

Data API Endpoints

We used previous the findone endpoint in our URL.

There are several endpoints available for use with the Data API

All endpoints start with the Base URL: https://data.mongodb-api.com/app/<Data API App ID>/endpoint/data/v1/action/

Find a single Document

Endpoint

POST Base_URL / findOne

The findone endpoint is used to find a single document in a collection.

Request Body

Example ↓

```
{
  "datasource" : "<data source name>",
  "database" : "<database name>",
  "collection" : "<collection name>",
  "filter "   : <query filter>
  "projection" : <projection>
}
```

Find Multiple Documents

Endpoint

POST. Base_URL/find

The find endpoint is used to find Multiple documents in a collection.

Request Body

Example ↓

```
{
```

```
"datasource" : "< data source name >",
"database"   : "< database name >",
"collection" : "< collection name >",
"Filter"     : <query Filter>,
"projection" : <projection>,
"Sort"       : <Sort expression>,
"Limit"      : <number>,
"Skip"       : <number>
}
```

Insert Multiple Document

Endpoint

POST Base-URL / insertMany

The insertMany endpoint is used to insert Multiple documents into a collection.

Request Body

Example ↓

```
{
"datasource" : "< data source name >",
"database"   : "< database name >",
```

```
"collection": "<collection name>",
"documents": [<document>, <document> ...]
}
```

Update a Single Document

Endpoint

POST Base_URL / updateone

Request Body

Example ↓

```
"datasource": "<data source name",
"database": "<database name>",
"collection": "<collection name>",
"filter": <query filter>
"update": <update expression>,
"upsert": true | false
}
```

Update Multiple Documents

Endpoint

POST Base_URL/ updateMany

Request Body

Example ↓

```
{
  "datasource" : "<data source name>",
  "database"   : "<database name>",
  "collection" : "<collection>",
  "Filter"     : <query filter>
  "Update"     : <update expression>
  "upsert" : true | false
}
```

Delete a Single Document

Endpoint

POST Base_URL/ deleteone

Request Body

Example → {

```
  "datasource" : "<data source name>",
  "database"   : "<database name>",
```

```
"collection" : "<collection name>",
"Filter    " : <query filter>
}
```

Delete Multiple Documents

Endpoint

POST Base_URL/deletemany

Request Body

Example

```
{
    "datasource" : "<data source name>",
    "database" : "<database name>",
    "collection" : "<collection name>",
    "Filter    " : <query filter>
}
```

Aggregate Documents

Endpoint

POST Base_URL/aggregate

Request Body

Example ↓

```
{
    "datasource" : "<data source name>",
    "database" : "<database name>",
    "collection" : "<collection name>",
    "pipeline" : [<pipeline expression>, ....]
}
```