MongoDB

Introduction

- **MongoDB** is a **No SQL** database. It is an **open-source, cross-platform, document-oriented** database written in C++.
- **MongoDB** uses the concept of the **document** to store data, which is more flexible than the **row concept** in the relational database management system.
- **MongoDB** doesn't **require predefined schemas** that allow you to add to or remove fields from documents more quickly.
- Like any database system, **MongoDB** allows you **to insert, update, and delete, and select data.** In addition, it supports other **features** like **Indexing, Aggregation, Specify collection and index types, File Storage**
- The philosophy of **MongoDB** is to create a full-featured database that is **scalable, flexible, and fast.**

History

- **MongoDB** was developed and is supported by a company named **10gen** which is a **New York based organization.**
- The initial development of **MongoDB** began in **2007** when the company was building a platform as a service like window azure.
- **MongoDB** 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 version of **MongoDB** was released in **August 2009** as **1.0**
- The first ready production of **MongoDB** has been considered from **version 1.6** which was released in **August 2010.**
- **MongoDB 6.0** was the latest and stable version which was released in **July 2022.**
- **MongoDB 8.0 preview** is the latest version as of **2024.**
- Complete **MongoDB Version History** with features: Link

MongoDB Installation Guide

[Installation Guide 1](#)   [Installation Guide 2](#)

MongoDB Server connection using terminal (command prompt)

- Open the bin path of the MongoDB server folder and copy the path.
- Open the terminal and navigate to the bin directory: **cd C:\Program Files\MongoDB\Server\6.0\bin** *-> press enter.*
- Use the command **mongosh** to connect to the MongoDB server: C:\Program Files\MongoDB\Server\6.0\bin>*mongosh -> press enter.*
- You are connected to the MongoDB server and can start writing commands: *test> show databases*

Data Types in MongoDB

Data Types are used to define the type of data stored in each field of a document. Some of the common MongoDB data types are:

- **String:** Used to store textual data. Strings are the most used data type.
➔ Example: {"name": "RVK"}

- **Integer:** Used to store numerical data (whole numbers).
➔ Example: {"age": 21}

- **Double:** Used to store floating-point numbers.
➔ Example: {"cgpa": 9.49}

- **Boolean:** Used to store a Boolean (true/false) value.
➔ Example: {"isPlaced": false}

- **Date:** Used to store dates in ISODate format.
➔ Example: {"joinedAt": ISODate("2023-07-24T00:00:00Z")}

- **Array:** Used to store arrays or lists of values.
➔ Example: {"tags": ["full stack dev", "cloud aspirant", "team player"]}

- **Object:** Used to store embedded documents (sub-documents).
➔ Example: {"address": {"Village": "NKP", "City": "Bargarh", "State": "OD"}}

- **ObjectId:** Used to store unique identifiers for documents.
➔ Example: {"_id": ObjectId("507f191e810c19729de860ea")}
➔ We cannot provide **ObjectId** while inserting a document into collection as it will be provided by default to out document.

- **Null:** Used to store a null value.
➔ Example: {"relationship": null}

Data Modelling in MongoDB

In MongoDB, data has a flexible schema. It is totally different from SQL database where you had to determine and declare a table's schema before inserting data. MongoDB collections do not enforce document structure.

For example:

- Let us take an example of a client who needs a database design for his website. His website has the following requirements:
- Every post is distinct (contains 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.
- Each post can have zero or more comments and the comments must contain username, message, data-time and likes.
- For the above requirement, a minimum of three tables are required in RDBMS.
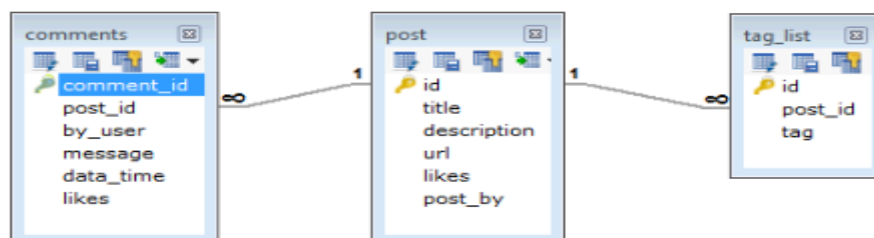
But in **MongoDB**, schema design will have one collection post and has the following **structure:**

{ _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 }] }

MongoDB Database commands

- There is no create database command in MongoDB. MongoDB do not provide any command to create databases.
- If we want a new database, we require **use** command, the use command in MongoDB creates a database if the database with the specified name doesn't exist. If a database with the same name already exists then it switches to that database.
- Syntax: use Database_Name
- Example: use klu
  - ➢ After executing above command if a database named as klu already exists then it switches to klu or else creates a database named as klu
- To check the **currently selected database** we can use **db** command.
- To get the **list of all available databases,** we can use below commands:
  - ➢ Show dbs
  - ➢ Show databases
- To **drop or to delete** an existing database, we can use the below command:
  - ➢ **db.dropDatabase()**
  - ➢ The above command will delete the current database that you are using; to delete other databases, you need to switch to them through **use db** command.

Example:

```
test> use klu
switched to db klu
klu> show dbs
Student    72.00 KiB
admin      40.00 KiB
config     72.00 KiB
klu-v      72.00 KiB
local      96.00 KiB
stu       144.00 KiB
klu> db
klu
klu> db.dropDatabase()
{ ok: 1, dropped: 'klu' }
```

- In the example a database named as klu is created and switched with **use klu**
- **show dbs** command is used to display the list of all available databases
- **db** command is used to check the currently selected database
- **db.dropDatabase()** command is used to delete the database klu.

MongoDB Collection Commands

- A **collection** is a **group of MongoDB documents.** Documents within a collection can have different fields. A **collection** is the equivalent of a **table** in a relational database system.
- **Collections** can be created by **db.createCollection("collectionName", Options)** command, **Options** is a document type, specifies the memory size and indexing of the collection. It is an optional parameter.
- There are **four types of options** which can be used while creating collections and they are **Capped, AutoIndexID, Size, Max.**
  - **Capped:** Enables a capped collection. Capped collections are fixed-size collections that automatically overwrite their oldest entries when they reach their maximum size.
  - **AutoIndexID:** Automatically creates an index on the **_id** field. Its default value is **false.**
  - **Size**: Specifies a maximum size in bytes for a capped collection. If **capped** is **true**, then this field must be specified.
  - **Max**: Specifies the maximum number of documents allowed in the capped collection.
- To **list out** all the collections we can use **show collections** or **db.getCollectionNames()** commands.
- To **drop** an existing collection we can use **db.collectionName.drop()** command, this command will delete the entire collection.
- If we want to **delete only the documents** inside the collection but not the entire collection then we can use **db.collectionName.deleteMany({}).**
- To **find all documents** available inside a collection we can use the following command: **db.collectionName.find().**

```
test> use klu
switched to db klu
klu> db.createCollection("CSE",{capped:true,size:5242880,max:3000});
{ ok: 1 }
klu> show collections
CSE
```

```
klu> db.CSE.deleteMany({});
{ acknowledged: true, deletedCount: 2 }
klu> show collections
CSE
```

<mark>MongoDB Document Commands</mark>

- **Documents** in MongoDB are JSON-like objects (BSON) that store data as field-value pairs.

MongoDB Insert

- **Documents** can be **inserted** into MongoDB using the **insertOne()** and **insertMany()** commands.
- The **db.collectionName.insertOne()** is used to insert a single document.
- The **db.collectionName.insertMany()** is for inserting multiple documents.

Examples

```
klu> db.CSE.insertOne({id:30959,name:"RVK",cgpa:9.49});
{
  acknowledged: true,
  insertedId: ObjectId("66a2001f20ac7139c55c2e89")
}
```

```
klu> db.CSE.insertMany([{id:30965,name:"Satya",cgpa:9.29},{id:30976,name:"Faizaan",cgpa:9.52},
{id:31090,name:"Sai",cgpa:9.35}]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("66a2216d20ac7139c55c2e8d"),
    '1': ObjectId("66a2216d20ac7139c55c2e8e"),
    '2': ObjectId("66a2216d20ac7139c55c2e8f")
  }
}
```

MongoDB Find

- **Documents** can be **selected** using **find()** and **findOne()** commands.
- The **find()** method accepts a query object. If left empty, all documents will be returned.
- The **findOne()** method is used to select only one document. This method accepts a query object. If left empty, it will return the first document it finds.

Examples

```
klu> db.CSE.findOne()
{
  _id: ObjectId("66a2007a20ac7139c55c2e8a"),
  id: 30959,
  name: 'RVK',
  cgpa: 9.49
}
```

```
klu> db.CSE.find()
[
  {
    _id: ObjectId("66a2007a20ac7139c55c2e8a"),
    id: 30959,
    name: 'RVK',
    cgpa: 9.49
  },
  {
    _id: ObjectId("66a2216d20ac7139c55c2e8d"),
    id: 30965,
    name: 'Satya',
    cgpa: 9.29
  },
  {
    _id: ObjectId("66a2216d20ac7139c55c2e8e"),
    id: 30976,
    name: 'Faizaan',
    cgpa: 9.52
  },
  {
    _id: ObjectId("66a2216d20ac7139c55c2e8f"),
    id: 31090,
    name: 'Sai',
    cgpa: 9.35
  }
]
```

```
klu> db.CSE.find({name:"RVK"})
[
  {
    _id: ObjectId("66a2007a20ac7139c55c2e8a"),
    id: 30959,
    name: 'RVK',
    cgpa: 9.49
  }
]
```

```
klu> db.CSE.find({id: {$gt: 30976}})
[
  {
    _id: ObjectId("66a2216d20ac7139c55c2e8f"),
    id: 31090,
    name: 'Sai',
    cgpa: 9.35
  }
]
```

```
klu> db.CSE.find({}, {_id: 0, name: 1, id: 1, cgpa: 1})
[
  { id: 30959, name: 'RVK', cgpa: 9.49 },
  { id: 30965, name: 'Satya', cgpa: 9.29 },
  { id: 30976, name: 'Faizaan', cgpa: 9.52 },
  { id: 31090, name: 'Sai', cgpa: 9.35 }
]
```

The above shown 5 examples explanations:

- The **first example** returned the first document because we used **findOne()**.
- The **second example** returned all the documents because we used **find()**.
- The **third example** returned a single document with name as **rvk** because we used a **query in find()**.
- The **fourth example** returned the documents with id greater than 30976 because we used the **$gt query** in **find()**.
- The **fifth example** returned all the documents but excluded the **_id** field because we specified **_id: 0** in **find()** and set other **fields to 1**.

MongoDB Update

- To **update** an existing document in MongoDB we can use **updateOne()** and **updateMany()** methods.

- These methods accept **two parameters** out of which the first parameter is a query object to define which document, or documents should be updated, and the second parameter is an object defining the updated data.
- The **updateOne()** method will update the first document that is found matching the provided query.
- The **updateMany()** method will update all documents that match the provided query.

Examples

```
klu> db.CSE.updateOne({id:30959},{$set: {name: "kalyan"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Here we update name to "kalyan" for student with id 30959 using updateOne()

```
klu> db.CSE.updateMany({}, {$inc: {id: 1}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

➢ Here we have updated the ids of all students by incrementing them with 1.

```
klu> db.CSE.updateOne({id:2100030959}, {$set: {id:2100030959, name: "RVK",
cgpa: 9.49}},{upsert: true})
{
  acknowledged: true,
  insertedId: ObjectId("66a2575a8102ecb76cd39fab"),
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
```

➢ In the above example, we tried to update a student with id 2100030959, and since they were not available in our records, a new document was created because we used **upsert.**
➢ **Upsert** is an option in MongoDB update operations. When set to true, it ensures that if the document specified in the filter criteria does not exist, a new document will be created with the specified update values.

MongoDB Delete

- In MongoDB, We can delete documents by using this methods **deleteOne()** or **deleteMany().**
- These methods accept a query object. The matching documents will be deleted.
- The **deleteOne()** method will delete the first document that matches the query provided.
- The **deleteMany()** method will delete all documents that match the query provided.
- If no query is provided in **deleteOne(),** then it deletes the first document it encounters in the collection.
- If no query is provided in **deleteMany(),** then it deletes all documents in the collection.

Examples

```
klu> db.CSE.deleteOne({id:2100030959})
{ acknowledged: true, deletedCount: 1 }
```
This will delete the record having id as 2100030959.

```
klu> db.CSE.deleteMany({id:31090})
{ acknowledged: true, deletedCount: 1 }
```
This will delete all the records having id as 31090.

```
klu> db.CSE.deleteOne({id:210030959})
{ acknowledged: true, deletedCount: 0 }
```
This will not delete any records as no records found with that id.

MongoDB Query Operators

Equality ($eq)

- It matches documents where the field value equals the specified value.
- **Example**:
  Find students with cgpa equal to 9.5

```
db.CSE.find({ cgpa: { $eq: 9.5 } });
```

Greater Than ($gt)

- It matches documents where the field value is greater than the specified value.
- **Example:** Find students with cgpa greater than 9.0

```
db.CSE.find({ cgpa: { $gt: 9.0 } });
```

Less Than ($lt)

- It matches documents where the field value is less than the specified value.
- **Example**: Find students with cgpa less than 8.0

```
db.CSE.find({ cgpa: { $lt: 8.0 } });
```

Greater Than or Equal ($gte)

- It matches documents where the field value is greater than or equal to the specified value.
- **Example**: Find students with cgpa greater than or equal to 8.5

```
db.CSE.find({ cgpa: { $gte: 8.5 } });
```

Less Than or Equal ($lte)

- It matches documents where the field value is less than or equal to the specified value.
- **Example**: Find students with cgpa less than or equal to 7.5

```
db.CSE.find({ cgpa: { $lte: 7.5 } });
```

Not Equal ($ne)

- It matches documents where the field value is not equal to the specified value.
- **Example**:
  Find students with cgpa not equal to 8.0

```
db.CSE.find({ cgpa: { $ne: 8.0 } });
```

In ($in)

- It matches documents where the field value is in an array of values.
- **Example**: Find students with cgpa of 8.5, 9.0, or 9.5

```
db.CSE.find({ cgpa: { $in: [8.5, 9.0, 9.5] } });
```

Not In ($nin)

- It matches documents where the field value is not in an array of specified values.
- **Example**: Find students with cgpa not equal to 8.5, 9.0, or 9.5

```
db.CSE.find({ cgpa: { $nin: [8.5, 9.0, 9.5] } });
```

And ($and)

- It matches documents that satisfy all the specified conditions.
- **Example**: Find students with id 2100030959 and cgpa 9.5

```
db.CSE.find({ $and: [{ id: 2100030959 }, { cgpa: 9.5 }] });
```

Or ($or)

- It matches documents that satisfy at least one of the specified conditions.
- **Example**: Find students with cgpa 9.5 or cgpa 8.5

```
db.CSE.find({ $or: [{ cgpa: 9.5 }, { cgpa: 8.5 }] });
```

Not ($not)

- It Inverts the effect of a query expression.
- **Example**: Find students with cgpa not greater than 9.0

```
db.CSE.find({ cgpa: { $not: { $gt: 9.0 } } });
```

Nor ($nor)

- It returns documents where both queries fail to match
- **Example**: Find students whose id and cgpa are not 2100030959, 9.5.

```
db.CSE.find({ $nor: [{ id: 2100030959 }, { cgpa: 9.5 }] });
```

MongoDB Update Operators

Set ($set)

- It updates the value of a field.
- **Example:** Set cgpa to 9.0 for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $set: { cgpa: 9.0 } });
```

Unset ($unset)

- It removes the specified field from a document.
- **Example:** Remove the cgpa field from the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $unset: { cgpa: "" } });
```

Inc ($inc)

- It increments the value of a field by a specified amount.
- **Example:** Increment cgpa by 0.1 for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $inc: { cgpa: 0.1 } });
```

Mul ($mul)

- It multiplies the value of a field by a specified amount.
- **Example:** Multiply cgpa by 1.1 for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $mul: { cgpa: 1.1 } });
```

Rename ($rename)

- It is used to rename a field.
- **Example:** Rename the cgpa field to grade for the student 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $rename: { cgpa: "grade" } });
```

Min ($min)

- It updates the field to a specified value if the specified value is less than the current value.

- **Example:** Set cgpa to 8.5 only if it is less than the current value for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $min: { cgpa: 8.5 } });
```

Max ($max)

- It updates the field to a specified value if the specified value is greater than the current value.
- **Example:** Set cgpa to 9.5 only if it is greater than the current value for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $max: { cgpa: 9.5 } });
```

Current Date ($currentDate)

- It sets the value of a field to the current date.
- **Example:** Set the lastModified field to the current date for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $currentDate: { lastModified: true } });
```

Pop ($pop)

- It removes the first or last element of an array.
- **Example:** Remove the last subject from the subject's array for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $pop: { subjects: 1 } });
```

Pull ($pull)

- It removes all array elements that match a specified query.
- **Example:** Remove "Database" from the subject's array for the student with id 2100030959.

```
db.CSE.updateOne({ id: 2100030959 }, { $pull: { subjects: "Database" } });
```

➢ The above listed operators are commonly used operators, to know more about all operators, visit the following link: Operators

Aggregate Operators in MongoDB

Match ($match)

- It filters the documents to pass only the documents that match the specified condition.
- **Example:** Find students with cgpa greater than 8.5.

```
db.CSE.aggregate([{ $match: { cgpa: { $gt: 8.5 } } }]);
```

Project ($project)

- It reshapes each document in the stream, optionally adding new fields or removing existing ones.
- **Example:** Include only id and name fields, exclude _id.

```
db.CSE.aggregate([{ $project: { _id: 0, id: 1, name: 1 } }]);
```

Group ($group)

- It groups input documents by the specified _id expression and applies the accumulator expressions.
- **Example:** Group students by cgpa and count the number of students in each group.

```
db.CSE.aggregate([
  { $group: { _id: "$cgpa", count: { $sum: 1 } } }
]);
```

Sort ($sort)

- It sorts the documents in the stream according to the specified fields.
- **Example:** Sort students by cgpa in descending order.

```
db.CSE.aggregate([{ $sort: { cgpa: -1 } }]);
```

Limit ($limit)

- It limits the number of documents passed to the next stage in the pipeline.
- **Example:** Limit the result to the first 5 students.

```
db.CSE.aggregate([{ $limit: 5 }]);
```

Skip ($skip)

- It skips a specified number of documents in the stream.
- **Example:** Skip the first 5 students.

```
db.CSE.aggregate([{ $skip: 5 }]);
```

Add Fields ($addFields)

- It adds new fields to the documents.
- **Example:** Add a field status with value "active" to all documents.

```
db.CSE.aggregate([{ $addFields: { status: "active" } }]);
```

Sum ($sum)

- It computes the sum of the values of a numeric field.
- **Example:** Calculate the total cgpa of all students.

```
db.CSE.aggregate([{ $group: { _id: null, totalCgpa: { $sum: "$cgpa" } } }]);
```

Average ($avg)

- It computes the average value of a numeric field.
- **Example:** Calculate the average cgpa.

```
db.CSE.aggregate([{ $group: { _id: null, averageCgpa: { $avg: "$cgpa" } } }]);
```

Min ($min)

- It returns the minimum value of a numeric field.
- **Example:** Find the minimum cgpa.

```
db.CSE.aggregate([{ $group: { _id: null, minCgpa: { $min: "$cgpa" } } }]);
```

Max ($max)

- It returns the maximum value of a numeric field.
- **Example:** Find the maximum cgpa.

```
db.CSE.aggregate([{ $group: { _id: null, maxCgpa: { $max: "$cgpa" } } }]);
```

Disclaimer

➢ The following notes are general summaries and overviews of the topics discussed.

➢ These notes are not exhaustive and do not cover all aspects of the subject matter.

➢ The information provided herein is intended for educational purposes only and should not be used as a substitute for professional advice, detailed study, or official course materials.

References

For more detailed information, please refer to the following resources:

Reference 1: Complete MongoDB Tutorial

Reference 2: Complete MongoDB Tutorial

Reference 3: MongoDB Cheat Sheet

Reference 4: MongoDB Most Asked Questions

Reference 5: Top MongoDB Interview Questions

*Prepared By: **Reddy Venkat Kalyan***