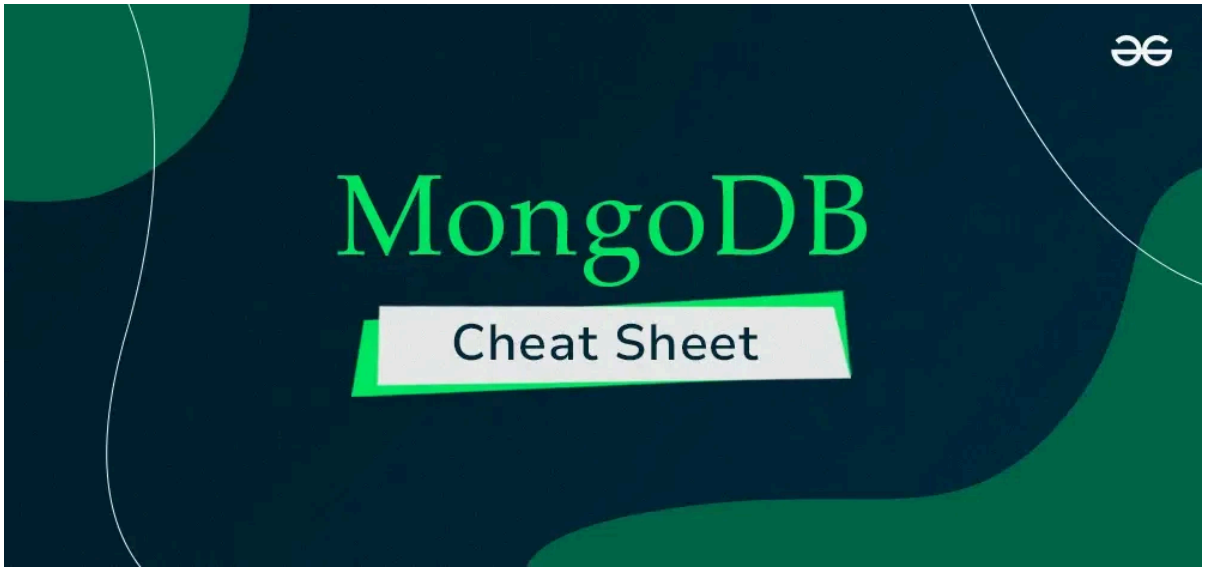
MongoDB Cheat Sheet (Basic to Advanced)

Last Updated : 25 Oct, 2024

MongoDB is a powerful **NoSQL database** known for its flexible, **document-oriented storage** that is ideal for handling **large-scale**, **complex data**. **MongoDB Atlas** (a cloud-based solution), **MongoDB Compass** (a GUI for data visualization) and the **MongoDB Shell** for command-line operations, users can efficiently perform **CRUD operations**.

The **MongoDB Aggregation Framework** enables advanced data analysis with **grouping**, **filtering** and **joining** capabilities. Perfect for scalable applications, MongoDB supports diverse **operators** and is optimized for modern data needs, making it a top choice for **dynamic**, **high-performance database management**.



Basics of MongoDB

Before proceeding towards the MongoDB cheat sheet let's have a quick look on MongoDB basic.

What is	MongoDB is a document-oriented **NoSQL database** that

DataTypes in MongoDB	MongoDB supports various data types including string, integer, double, boolean, arrays, objects, dates, and null.
What is ObjectId in MongoDB	ObjectId is a 12-byte **hexadecimal** number that uniquely identifies documents in a collection.
MongoDB Atlas	MongoDB Atlas is a fully managed cloud database service. It provides automated backups, monitoring, and security features.
MongoDB Compass	MongoDB Compass is a graphical user interface for **MongoDB**. It allows users to visualize data, run queries, and analyze performance.
What is MongoDB Shell	MongoDB Shell is a **command-line interface** for interacting with MongoDB instances. It allows users to execute queries, perform administrative tasks, and manage databases.

CRUD Operations in MongoDB

This section covers the basics of working with your MongoDB database using CRUD operations. You'll learn how to **Create**, **Read**, **Update**, and **Delete** documents which allowing us to efficiently manage your data. Get ready to add, retrieve, modify, and remove information easily

Connect to MongoDB

```
mongo
```

Open a terminal and start the **MongoDB shell** by typing mongo.

```
use blog
```

Create (if not exists) and use the '**blog**' database

Create Collections

```
// Create a 'posts' collection
db.createCollection("posts")

// Create a 'users' collection
db.createCollection("users")
```

Create two collections: posts for storing blog posts and users for storing user information.

Insert Operations

Insert a single document into '**posts**' collection

```
db.posts.insertOne({
    title: "Introduction to MongoDB",
    content: "MongoDB is a NoSQL database.",
    author: "John Doe",
    tags: ["mongodb", "nosql", "database"]
})
```

Insert multiple documents into '**users**' collection

```
db.users.insertMany([
    {
        username: "johndoe",
        email: "johndoe@example.com",
        age: 30
    },
    {
```

```
    }
])
```

Update Operations

Update a document in '**users**' collection

```
db.users.updateOne(
    { username: "johndoe" },
    { $set: { age: 31 } }
)
```

Update multiple documents in 'posts' collection

```
db.posts.updateMany(
    { tags: "mongodb" },
    { $addToSet: { tags: "database" } }
)
```

Delete Operations

Delete a document from '**users**' collection

```
db.users.deleteOne({ username: "janedoe" })
```

Delete multiple documents from '**posts**' collection

```
db.posts.deleteMany({ author: "John Doe" })
```

Drop the entire 'users' collection

```
db.users.drop()
```

Query Operations

Find all documents in '**posts**' collection

Find one document in '**posts**' collection

```
db.posts.findOne({ title: "Introduction to MongoDB" })
```

Find and modify a document in 'posts' collection

```
db.posts.findOneAndUpdate(
    { title: "Introduction to MongoDB" },
    { $set: { content: "MongoDB is a flexible and scalable NoSQL
database." } }
)
```

Find one and delete a document in '**posts**' collection

```
db.posts.findOneAndDelete({ author: "John Doe" })
```

Find one and replace a document in '**posts**' collection

```
db.posts.findOneAndReplace(
    { title: "Introduction to MongoDB" },
    { title: "MongoDB Overview", content: "A detailed guide to MongoDB."
}
)
```

Query with Projections

Find documents with projection (only return 'title' and 'author' fields)

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

Query nested documents (e.g., find users with email ending in '.com')

```
db.users.find({ "email": /.*\.com$/ })
```

Query documents with null or missing fields

```
db.users.find({ email: null })
```

```
// Show available databases
show dbs

// Show collections in the current database
show collections
```

To see a list of available databases and their collections

MongoDB Operators

MongoDB operators are like tools that help you work with your data effectively. They allow you to find specific information, make changes and analyze your data in MongoDB.

Mastering these operators gives you the ability to manage and explore your data more efficiently, uncovering valuable insights along the way.

1. Comparison Operators

Find documents where age is greater than 30 in 'users' collection

```
db.users.find({ age: { $gt: 30 } })
```

Find documents where age is less than or equal to 28 in 'users' collection

```
db.users.find({ age: { $lte: 28 } })
```

Find documents where title is equal to "MongoDB Overview" in 'posts' collection

```
db.posts.find({ title: { $eq: "MongoDB Overview" } })
```

Find documents where age is not equal to 30 in 'users' collection

```
db.users.find({ age: { $ne: 30 } })
```

In these queries, we utilize the **$gt** (**greater than**), **$lt** (**less than**), and **$eq** (**equality**) comparison operators to filter documents based on specific criteria.

2. Logical Operators

Find documents where age is greater than 25 AND less than 35 in 'users' collection

```
db.users.find({ $and: [ { age: { $gt: 25 } }, { age: { $lt: 35 } } ] })
```

Find documents where username is "johndoe" OR email is "janedoe@example.com" in 'users' collection

```
db.users.find({ $or: [ { username: "johndoe" }, { email:
"janedoe@example.com" } ] })
```

Find documents where age is NOT equal to 30 in 'users' collection

```
db.users.find({ age: { $not: { $eq: 30 } } })
```

Find documents where age is neither 30 nor 31 in 'users' collection

```
db.users.find({ age: { $nor: [ { $eq: 30 }, { $eq: 31 } ] } })
```

We use the $and operator to find documents where multiple conditions must be satisfied simultaneously. The $or operator is utilized to find documents where at least one of the specified conditions is met. Using the $not operator, we exclude documents where a specific condition is true. The $nor operator is used to find documents where none of the specified conditions are met.

3. Arithmetic Operators

Let's Add 5 to the age of all users in 'users' collection

```
db.users.updateMany({}, { $add: { age: 5 } })
```

Let's Subtract 2 from the age of users aged 30 in 'users' collection

```
db.users.updateMany({ age: 30 }, { $subtract: { age: 2 } })
```

```
db.users.updateMany({}, { $multiply: { age: 2 } })
```

Let's Divide the age of all users by 2 in 'users' collection

```
db.users.updateMany({}, { $divide: { age: 2 } })
```

Let's Calculate the absolute value of the age of all users in 'users' collection

```
db.users.updateMany({}, { $abs: { age: true } })
```

We use the $add, $subtract, $multiply, and $divide operators to perform addition, subtraction, multiplication, and division respectively on numeric fields. The $abs operator calculates the absolute value of numeric fields.

4. Field Update Operators

Let's Update the age of users to the maximum value of 40 in 'users' collection

```
db.users.updateMany({}, { $max: { age: 40 } })
```

Let's Update the age of users to the minimum value of 20 in 'users' collection

```
db.users.updateMany({}, { $min: { age: 20 } })
```

Let's Increment the age of users by 1 in 'users' collection

```
db.users.updateMany({}, { $inc: { age: 1 } })
```

Let's Multiply the age of users by 1.1 in 'users' collection

```
db.users.updateMany({}, { $mul: { age: 1.1 } })
```

We use the $max and $min operators to update fields to the maximum or minimum value respectively. The $inc operator increments numeric fields by a specified value. The $mul operator multiplies numeric fields by a specified value.

Let's Find documents where 'tags' field is an array in 'posts' collection

```
db.posts.find({ tags: { $isArray: true } })
```

Let's Find documents in 'posts' collection where the size of the 'tags' array is 3

```
db.posts.find({ $expr: { $eq: [{ $size: "$tags" }, 3] } })
```

Let's Find the first element of the 'tags' array in each document of 'posts' collection

```
db.posts.aggregate([
    { $project: { firstTag: { $arrayElemAt: ["$tags", 0] } } }
])
```

Let's Concatenate the 'tags' arrays of all documents in 'posts' collection

```
db.posts.aggregate([
    { $group: { _id: null, allTags: { $concatArrays: "$tags" } } }
])
```

Let's Reverse the 'tags' array in all documents of 'posts' collection

```
db.posts.updateMany({}, { $reverseArray: "$tags" })
```

We use the $isArray operator to find documents where a field is an array. The $size operator is used to find documents based on the size of an array field. With $arrayElemAt, we retrieve a specific element from an array field. The $concatArrays operator concatenates arrays. Finally, $reverseArray reverses the elements of an array.

6. Array Update Operators

Let's Remove all occurrences of "**mongodb**" from the 'tags' array in 'posts' collection

```
db.posts.updateMany({}, { $pull: { tags: "mongodb" } })
```

Let's Remove the last element from the '**tags**' array in all documents of '**posts**' collection

```
db.posts.updateMany({}, { $pop: { tags: 1 } })
```

Let's Remove all occurrences of "**nosql**" and "database" from the '**tags**' array in '**posts**' collection

```
db.posts.updateMany({}, { $pullAll: { tags: ["nosql", "database"] } })
```

Let's Add "**newtag**" to the end of the '**tags**' array in a specific document in '**posts**' collection

```
db.posts.updateOne({ title: "Introduction to MongoDB" }, { $push: {
tags: "newtag" } })
```

Let's Update the '**tags**' array in all documents where "**mongodb**" is present with "**updatedtag**"

```
db.posts.updateMany({ tags: "mongodb" }, { $set: { "tags.$":
"updatedtag" } })
```

7. String Expression Operators

Concatenate the 'title' and 'content' fields into a new field '**fullText**' in 'posts' collection

```
db.posts.aggregate([
    {
        $project: {
            fullText: { $concat: ["$title", " ", "$content"] }
        }
    }
])
```

Let's Compare the 'title' field case insensitively to "**MongoDB**" in 'posts'

```
db.posts.find({ $expr: { $eq: [{ $strcasecmp: ["$title", "MongoDB"] },
0] } })
```

Let's Convert the '**title**' field to **uppercase** in 'posts' collection

```
db.posts.updateMany({}, { $set: { title: { $toUpper: "$title" } } })
```

Let's Convert the '**title**' field to **lowercase** in 'posts' collection

```
db.posts.updateMany({}, { $set: { title: { $toLower: "$title" } } })
```

Let's Extract the first 5 characters from the '**title**' field in 'posts' collection

```
db.posts.aggregate([
    { $project: { firstFiveChars: { $substrCP: ["$title", 0, 5] } } }
])
```

We use the **$concat** operator to concatenate fields or strings. **$strcasecmp** compares strings case insensitive. The **$toUpper** operator converts a string to uppercase. $toLower converts a string to lowercase. **$substrCP** extracts a substring from a string based on code points.

MongoDB Aggregation Framework

We'll perform various **aggregation** operations using MongoDB's aggregation framework

Let's Update documents with aggregation pipeline: multiply '**age**' field by 2 and store in '**doubleAge**' field

```
db.users.aggregate([
    { $addFields: { doubleAge: { $multiply: ["$age", 2] } } },
    { $out: "users" }
])
```

Let's Count the number of documents in 'users' collection

```
db.users.aggregate([
```

Let's Group documents in 'users' collection by 'age' and calculate the count in each group

```
db.users.aggregate([
    { $group: { _id: "$age", count: { $sum: 1 } } }
])
```

Let's Perform a left outer join between 'posts' and 'users' collections based on 'author' field

```
db.posts.aggregate([
    {
        $lookup: {
            from: "users",
            localField: "author",
            foreignField: "username",
            as: "author_info"
        }
    }
])
```

Let's Get the first document in each group sorted by 'age' in descending order in 'users' collection

```
db.users.aggregate([
    { $sort: { age: -1 } },
    { $group: { _id: null, oldestUser: { $first: "$$ROOT" } } }
])
```

Let's Perform map-reduce operation to calculate the total age of all users

```
var mapFunction = function () {
    emit("totalAge", this.age);
};

var reduceFunction = function (key, values) {
    return Array.sum(values);
```

```
    mapFunction,
    reduceFunction,
    { out: { inline: 1 } }
);
```

We use various stages such as **$addFields**, **$out**, **$count**, **$group**, **$lookup**, **$first**, and map-reduce for different aggregation operations.

Aggregation framework allows us to perform complex computations, transformations, and data analysis on MongoDB collections efficiently.

MongoDB Indexing

Indexing enhances query performance and allows for efficient data retrieval in MongoDB

Let's Create a single field index on the '**username**' field in the '**users**' collection

```
db.users.createIndex({ username: 1 })
```

Let's Get the list of indexes on the 'users' collection

```
db.users.getIndexes()
```

Let's Drop the index on the 'username' field in the 'users' collection

```
db.users.dropIndex("username_1")
```

Let's Create a compound index on the 'title' and 'content' fields in the 'posts' collection

```
db.posts.createIndex({ title: 1, content: 1 })
```

Let's Create a multikey index on the 'tags' array field in the 'posts' collection

```
db.posts.createIndex({ tags: 1 })
```

Let's Create a text index on the 'content' field in the 'posts' collection

```
db.users.createIndex({ email: 1 }, { unique: true })
```

We use createIndex() to create various types of indexes, such as single field, compound, multikey, text, and unique indexes. getIndexes() retrieves the list of indexes on a collection. dropIndex() drops an index by its name.

Transactions in MongoDB

MongoDB supports multi-document ACID transactions, allowing for **atomicity, consistency, isolation**, and durability.

```
// Start a session
session = db.getMongo().startSession()

// Start a transaction
session.startTransaction()

try {
    // Perform operations within the transaction
    db.collection1.insertOne({ field1: "value1" }, { session: session })
    db.collection2.updateOne({ field2: "value2" }, { $set: { field3:
"value3" } }, { session: session })

    // Commit the transaction
    session.commitTransaction()
} catch (error) {
    // Abort the transaction on error
    session.abortTransaction()
}
```

Data Modeling in MongoDB

Data modeling in MongoDB involves designing schemas and relationships between documents.

```
// Relationship: Embedding data in documents
db.users.insertOne({
    username: "john_doe",
```

```
        { title: "Post 2", content: "Content 2" }
    ]
})

// Relationship: Referencing documents
db.comments.insertOne({
    user_id: ObjectId("user_id_here"),
    post_id: ObjectId("post_id_here"),
    content: "Comment content"
})

// Specify JSON schema validation
db.createCollection("collection_name", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["field1", "field2"],
            properties: {
                field1: {
                    bsonType: "string"
                },
                field2: {
                    bsonType: "number"
                }
            }
        }
    }
})

// Scaling in MongoDB involves sharding, replication, and proper index
usage to distribute data across multiple servers.
```

We demonstrate embedding data in documents and referencing documents to model relationships between collections. **JSON** schema validation ensures data integrity by enforcing structure and data types. Scaling in **MongoDB** involves strategies like sharding and replication to handle large volumes of data.

as **MongoDB Atlas**, **MongoDB Compass** and the **MongoDB Shell**, simplify data management while the **Aggregation Framework** enables advanced querying and **data analysis**.

With its focus on scalability, performance, and ease of use, MongoDB is an ideal choice for modern applications. It's a preferred solution for developers and organizations seeking efficient, high-performance **database systems** in today's **data-driven landscape**.

Next Article

PostgreSQL - Cheat Sheet : Basic to Advanced

Similar Reads

MongoDB Cheat Sheet (Basic to Advanced)

MongoDB is a powerful NoSQL database known for its flexible, document-oriented storage that is ideal for handling large-scale, complex data. MongoDB Atlas (a cloud-based solution), MongoDB Compass (a GUI for...

12 min read

PostgreSQL - Cheat Sheet : Basic to Advanced

PostgreSQL is a powerful, open-source object-relational database management system (ORDBMS). It is designed to help developers build robust applications and allow administrators to maintain data integrity...

5 min read

How to Back Up and Restore a MongoDB Database?

MongoDB is considered one of the classic examples of NoSQL systems. Its documents are made up of key-value pairs, which are the basic unit of data in MongoDB. Whether we're dealing with accidental data loss,...

5 min read

30 Days of SQL - From Basic to Advanced Level

This basic to advanced SQL tutorial covers the entire SQL syllabus in a structured way and provides the best learning material and strategies to master complete SQL in 30 Days. We have laid out the complete SQL...

8 min read

MongoDB, a leading NoSQL database, is known for its flexibility, scalability, and ease of use. However, like any database, MongoDB is susceptible to data loss due to hardware failures, software issues, human errors, ...

7 min read

How to Connect Node to a MongoDB Database ?

Connecting Node.js to MongoDB is a common task for backend developers working with NoSQL databases. MongoDB is a powerful, flexible, and scalable database that stores data in a JSON-like format. In this step-b...

6 min read

How to List all Databases in the Mongo Shell?

Knowing how to list databases in MongoDB is an important part of managing your data effectively. By using basic MongoDB shell commands, you can easily see what databases you have and understand their sizes. By...

4 min read

How to Integrate MongoDB in Next.js ?

A developer considers various options carefully for his or her tech stack before writing code. The primary objective is choosing a tech stack that aligns with the project requirements. Therefore, each tool within the...

10 min read

How to Set a Primary Key in MongoDB?

In MongoDB, primary keys play an important role in uniquely identifying documents within a collection. While MongoDB doesn't have a concept of primary keys in the same way as relational databases but it allows for t...

3 min read

How to Get String Length in MongoDB

In MongoDB, determining the length of a string stored in a document can be useful for various data processing tasks and queries. MongoDB provides several operators and methods that allow us to calculate the length of...

5 min read

Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:
K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

Company
About Us
Legal
Privacy Policy
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML
Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies
HTML
CSS
JavaScript

Python Tutorial
Python Programming Examples
Python Projects
Python Tkinter

Bootstrap
Web Design
Django

Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

DevOps
Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design
High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation
Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects
Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos
DSA
Python
Java
C++
Web Development
Data Science
CS Subjects