

MONGO DB

Welcome to the MongoDB Documentation

Find the guides, samples, and references you need to use the database, visualize data, and build applications on the MongoDB data platform.

MongoDB is a source-available cross-platform **document (JSON) -oriented** database program. Classified as a **NoSQL** database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License which is deemed non-free by several distributions. Wikipedia.

Initial release date: 11 February 2009

Developer: MongoDB Inc.

License: Server Side Public License

Operating system: Windows Vista and later, Linux, OS X 10.7 and later, Solaris, FreeBSD

Stable release: 6.0.1 / 19 August 2022, 7 months ago

Programming languages: JavaScript, Python, Java, PHP, Go, C, C++, Perl, Ruby

introduction to MongoDB -

1. it is an open-source non relational(non Traditional/no SQL) database that stores the data in the form of collection and documents.
2. it preserves most of the functionality while offering horizontal scalability.
3. it stores the json documents in the form of a collection having dynamic schemas.
4. it stores all the related information together which enhances the speed of query processing.

FEATURES OF MongoDB



MongoDB is a popular NoSQL database management system known for its flexibility and scalability. Here are some of the key features of MongoDB:

1. ****Schema-less****: MongoDB is a document-oriented database, which means it doesn't require a predefined schema. You can store documents with different structures in the same collection.
2. ****JSON-like Documents****: Data is stored in BSON (Binary JSON) format, which is a binary representation of JSON-like documents. This makes it easy to work with data in a format similar to native data structures in programming languages.
3. ****Flexible Data Model****: MongoDB supports complex, nested data structures, arrays, and embedded documents. This flexibility is particularly useful for applications with evolving data requirements.
4. ****Highly Scalable****: MongoDB is designed for horizontal scaling, allowing you to distribute data across multiple servers or clusters to handle large amounts of data and high traffic loads.
5. ****Automatic Sharding****: MongoDB offers automatic data sharding, which divides a collection into smaller chunks and distributes them across multiple servers. This enables horizontal scaling and better performance.
6. ****Replication****: MongoDB supports automatic failover and data redundancy through replica sets. It ensures data availability and

reliability by maintaining multiple copies of data across different servers.

7. **Query Language:** MongoDB provides a powerful and expressive query language that supports a wide range of query operations, including filtering, sorting, and aggregation.

8. **Indexes:** MongoDB supports various types of indexes, including compound indexes and geospatial indexes, to improve query performance.

9. **Geospatial Capabilities:** MongoDB has built-in geospatial features that allow you to perform location-based queries and store geospatial data efficiently.

10. **Aggregation Framework:** MongoDB's aggregation framework provides powerful data transformation and manipulation capabilities, enabling complex data analysis and reporting.

11. **Full-Text Search:** MongoDB offers full-text search capabilities, making it suitable for applications that require text-based search functionality.

12. **Security:** MongoDB provides robust security features, including authentication, authorization, encryption at rest, and auditing, to protect your data.

14. **Community and Enterprise Editions:** MongoDB comes in both community and enterprise editions, with additional features and support available in the enterprise version.

15. **Drivers and Ecosystem:** MongoDB provides official drivers for various programming languages, and it has a rich ecosystem of tools and libraries that make it easier to work with MongoDB in different environments.

These features make MongoDB a versatile and powerful database solution for a wide range of applications, including web and mobile applications, IoT, and big data projects.

Official Site Doc Link:

<https://docs.mongodb.com/manual/crud/>

<https://docs.mongodb.com/manual/tutorial/query-documents/>

MongoDB

It is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on the concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents(JSON). It is the equivalent of an RDBMS(Relational Database Management System) table.

A collection exists within a single database.

Collections do not enforce a schema.

Documents within a collection can have different fields.

Typically, all documents in a collection are of similar or related purpose.

Schema - A schema is a JSON object that defines the structure and contents of your data.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema

means that documents in the same collection do not need to have the same set of fields or

structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB

Example : -

```
{
  "Name": "Sushant",
  "Qualification": "BTech",
  "Subjects": ["c", "java", "Python"],
  "yearOfPassing": 2020,
  "address": [{
    "current": {
      "addressLine1": "Filmcity Mumbai",
      "city": "Mumbai"
    },
  },
```

```
    "permanent":{
      "addressLine1":"Gandhi Nagar",
      "city":"Patliputra"
    }
  }
]
```

Check and validate your JSON

<https://jsonformatter.curiousconcept.com/>

or

<https://www.toptal.com/developers/json-formatter>

Any relational database has a typical schema design that shows the number of tables and the relationship between these tables (one to one , one to many etc). While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS (Relational Database Management System)

Schema less: MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.

1. Structure of a single object is clear.
2. No complex joins.
3. Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
4. Ease of scale-out: MongoDB is easy to scale.
5. Conversion/mapping of application objects to database objects is not needed.
6. Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

Document Oriented Storage: Data is stored in the form of JSON style documents.

- Index on any attribute
- Replication and high availability

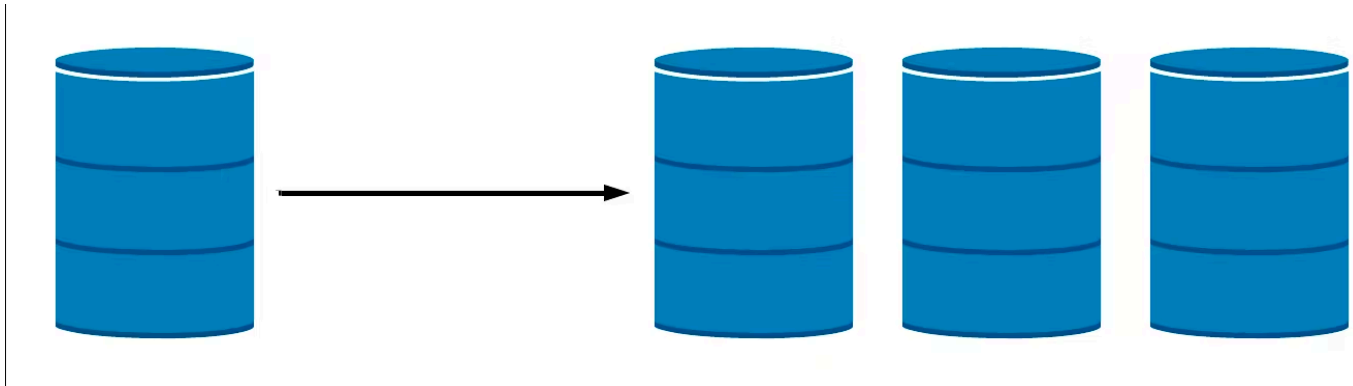
Auto-sharding
Rich queries
Fast in-place updates
Professional support by MongoDB

Where to Use MongoDB?

1. Big Data
2. Content Management and Delivery
3. Mobile and Social Infrastructure
4. User Data Management
5. Data Hub

Horizontal scaling or scalability in MongoDB

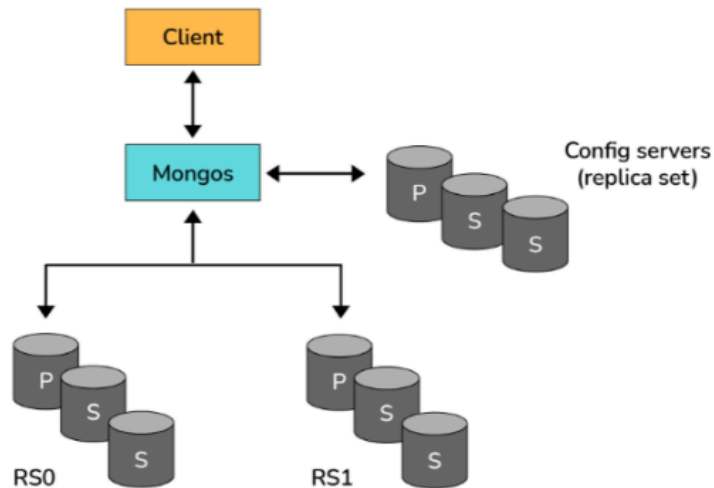
Horizontal scaling, also known as scale-out, refers to bringing on additional nodes to share the load. Scaling MongoDB horizontally is achieved primarily through sharding.



Sharding-

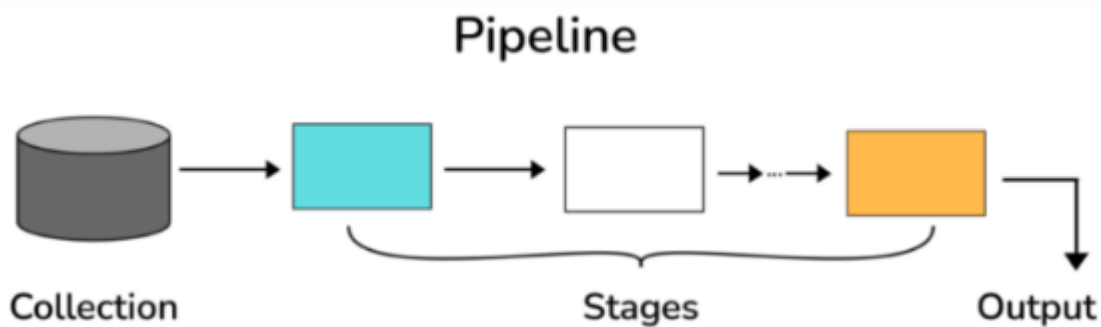
Sharding is the process of splitting data up across machines. We also use the term "partitioning" sometimes to describe this concept. We can store more data and handle more load without requiring larger or more powerful machines, by putting a subset of data on each machine.

In the figure below, RS0 and RS1 are shards. MongoDB's sharding allows you to create a cluster of many machines (shards) and break up a collection across them, putting a subset of data on each shard. This allows your application to grow beyond the resource limits of a standalone server or replica set.



Aggregation Framework in MongoDB?

- The aggregation framework is a set of analytics tools within MongoDB that allow you to do analytics on documents in one or more collections.
- The aggregation framework is based on the concept of a pipeline. With an aggregation pipeline, we take input from a MongoDB collection and pass the documents from that collection through one or more stages, each of which performs a different operation on its inputs (See figure below). Each stage takes as input whatever the stage before it is produced as output. The inputs and outputs for all stages are documents—a stream of documents.



Replica Set in MongoDB?

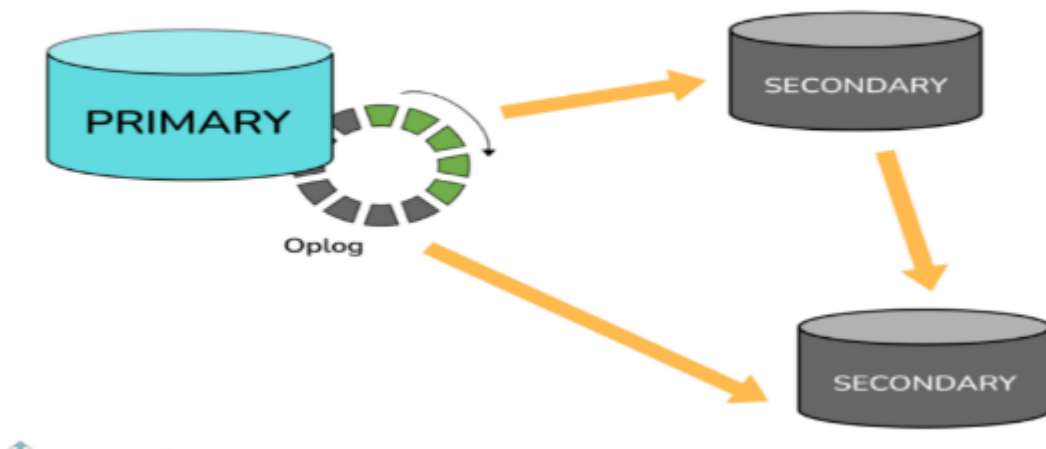
To keep identical copies of your data on multiple servers, we use replication. It is recommended for all production deployments. Generally in any development company projects are deployed in below environments ■

1. Development Environment
2. Test Environment
3. UAT (User Acceptance Testing) Environment - Clients
4. Production Environment - Live For customers

Use replication to keep your application running and your data safe, even if something happens to one or more of your servers.

Such replication can be created by a replica set with MongoDB. A replica set is a group of servers with one primary, the server taking writes, and multiple secondaries, servers that keep copies of the primary's data. If the primary crashes, the secondaries can elect a new primary from amongst themselves.

Architecture in MongoDB.



Some utilities for backup and restore in MongoDB?

The mongo shell does not include functions for exporting, importing, backup, or restore. However, MongoDB has created methods for accomplishing this, so that no scripting work or complex GUIs are needed. For this, several utility scripts are provided that can be used to get data in or out of the database in bulk. These utility scripts are:

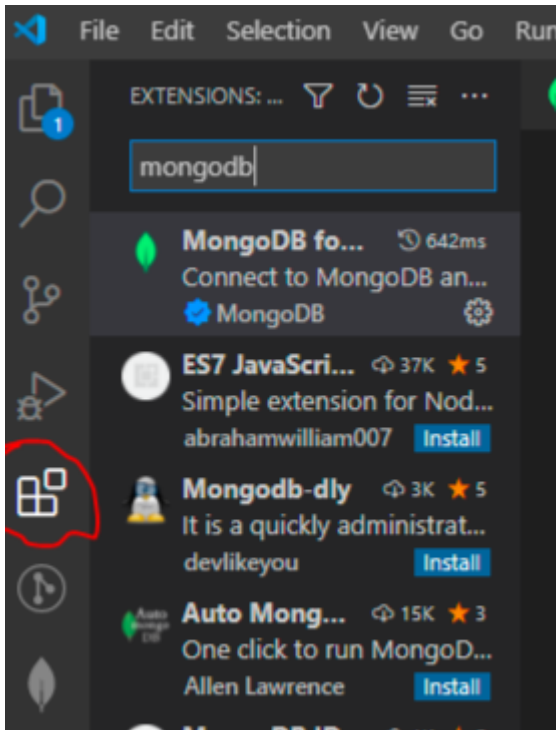
- mongoimport
- mongoexport
- mongodump
- mongorestore

MONGODB ATLAS WITH VSCODE

Using mongodb in vs code and starting a mongoDB playground.

Steps:-

1. goto - vscode extension option as below



Search for mongodb

2. click on install button

3. click on MongoDB icon



3. connect to your mongoDB connection string. As below



Connect using VS Code


Connect to a MongoDB host in Visual Studio Code




Go Back


Close


Click on the connect button. Copy your connection string and paste as below screen. Add the password and press enter.

 MongoDB

Enter your connection string (SRV or standard) (Press 'Enter' to confirm or 'Escape' to cancel)

 **MongoDB.**
Navigate your databases and collections, use playgrounds for exploring and transforming your data

● Connected to: **blogcluster.3gg4i.mongodb.net** 

All set. Ready to start?
Create a playground. 

Create playground

Connect with
Connection String

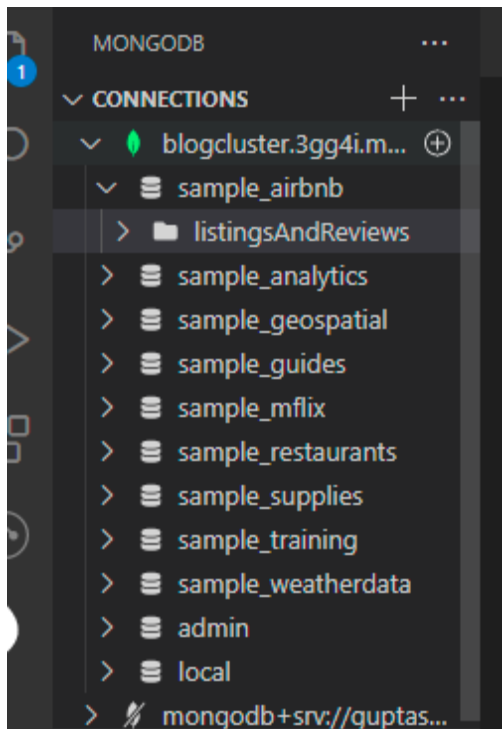
Connect

Advanced
Connection Settings

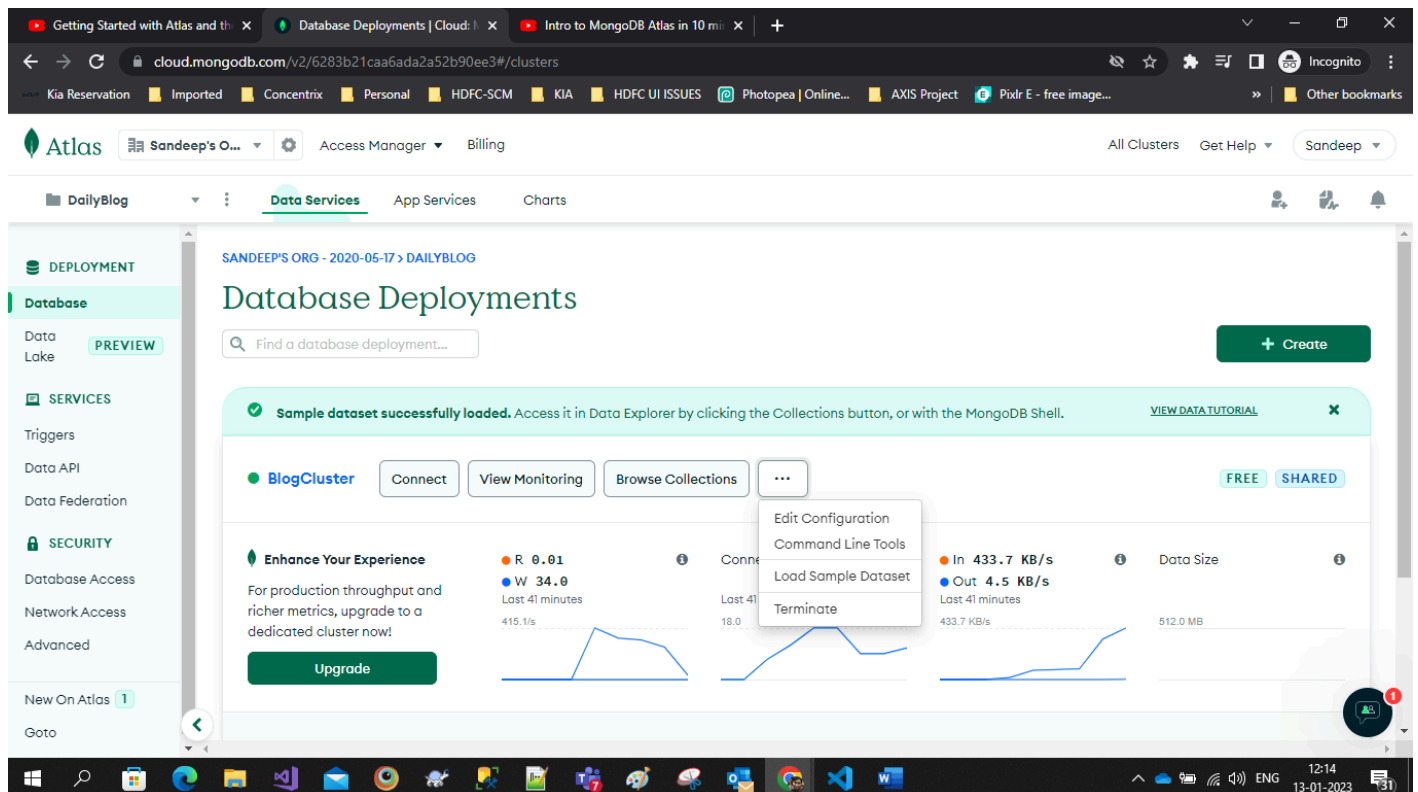
Open form

Ctrl + Shift + P for all MongoDB Command Palette options

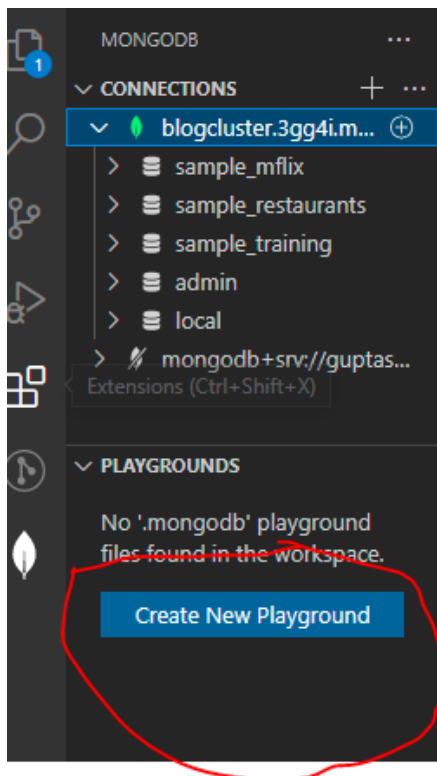
After successful connection you will see your cluster and databases.



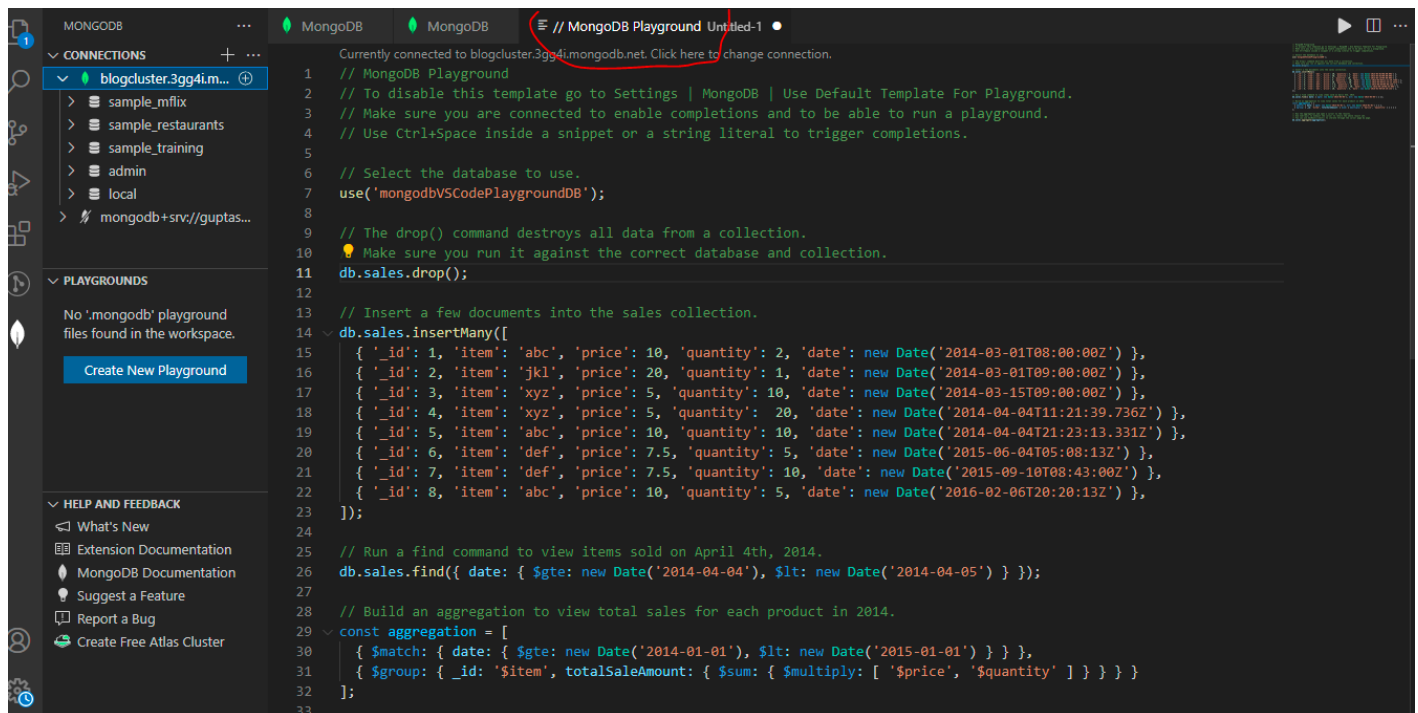
If there is no collection. You will see a sample collection option in MongoDB Atlas. Select that, MongoDB will create some sample databases and collections for you.



4.click to create a new playground button.



You will see below screen



We have following command already in the playground

```
use('mongodbVSCodePlaygroundDB'); // if this doesnot exist MongoDB will create this
and then use it.
```

```
db.sales.drop(); // delete your collection
```

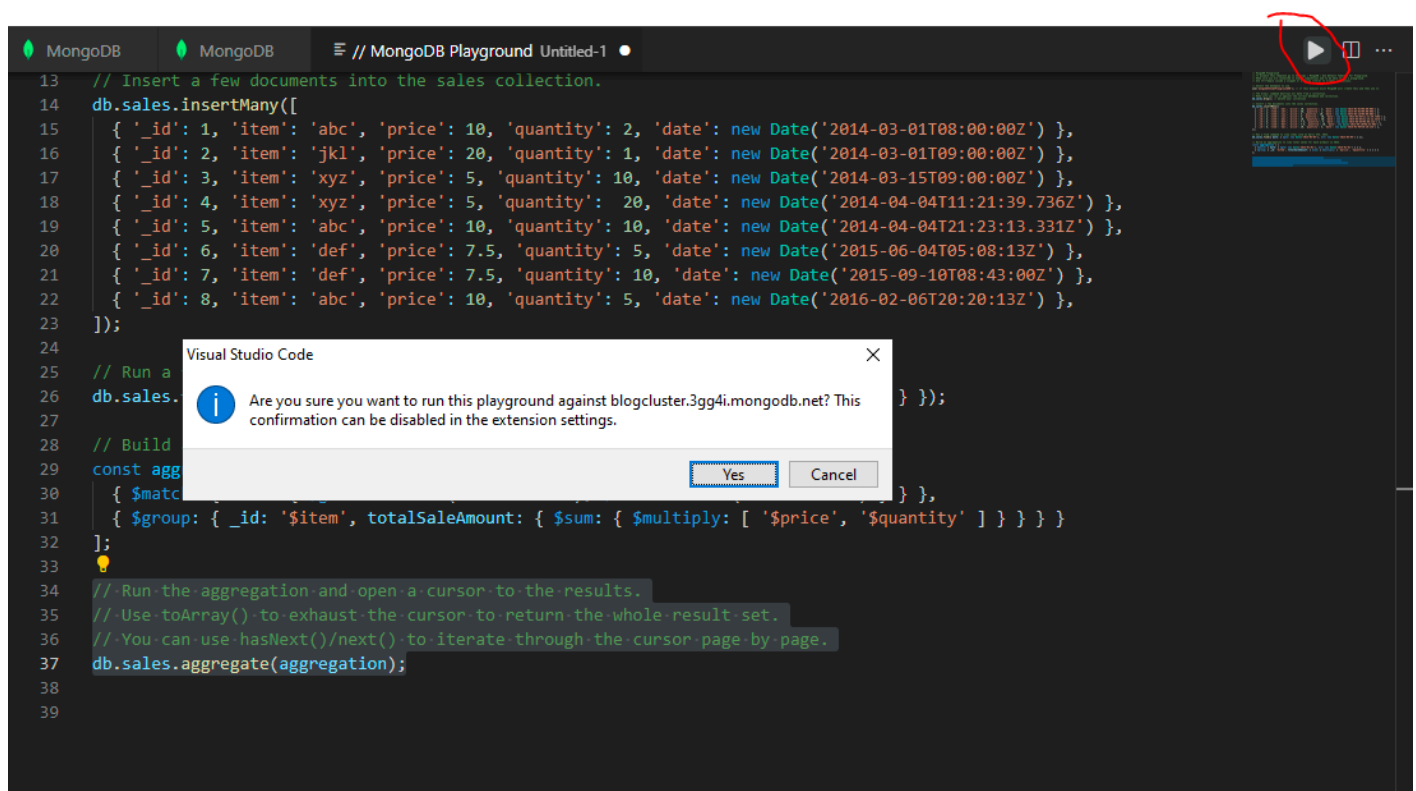
```
Insert a few documents into the sales collection.
db.sales.insertMany()
```

```
// Run a find command to view items sold on April 4th, 2014.
db.sales.find()
```

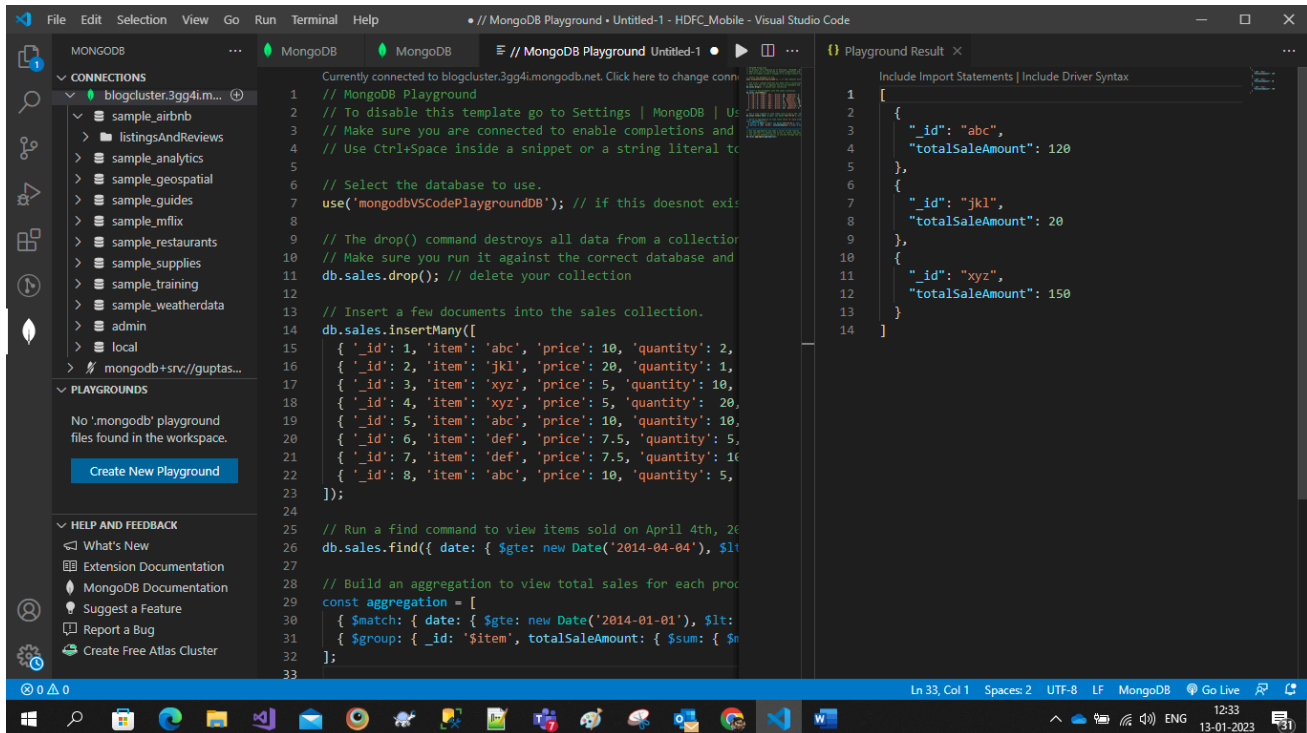
```
// Build an aggregation to view total sales for each product in 2014.
const aggregation = [
  { $match: { date: { $gte: value , $lt: value } } } ];
```

```
// Run the aggregation and open a cursor to the results.
db.sales.aggregate(aggregation);
```

To run your code or playground use the run icon in top right of VS CODE as below



After running your MongoDB Query , you will see below results

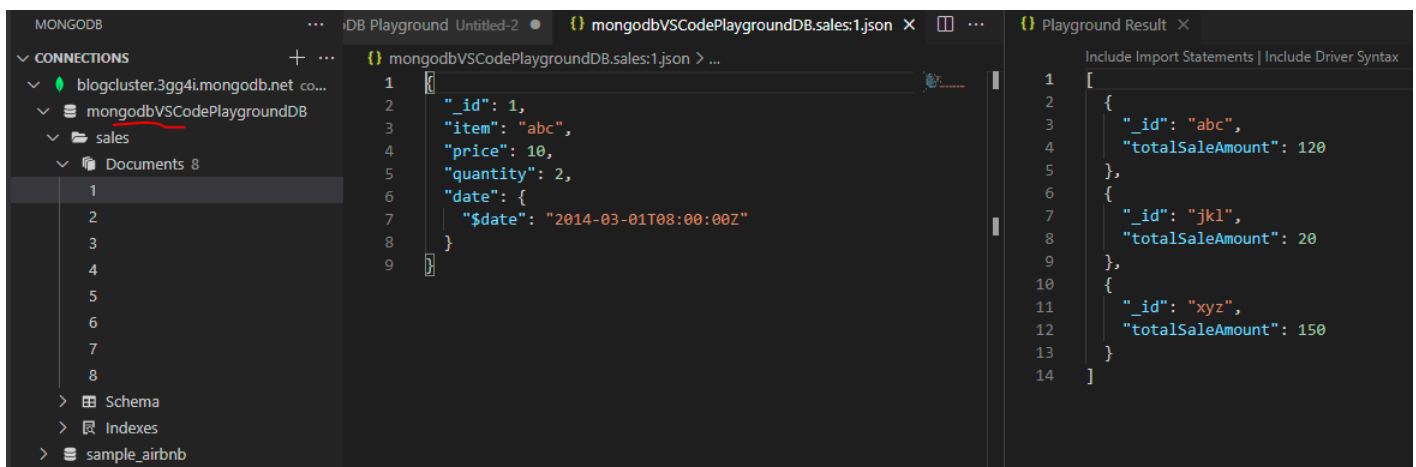


The screenshot shows the MongoDB Playground interface in Visual Studio Code. The left sidebar displays the 'CONNECTIONS' list with 'blogcluster.3gg4i.m...' selected. The main editor area contains a JavaScript query for MongoDB. The query includes comments and commands to connect to the database, select the 'sales' collection, and insert documents. The right sidebar shows the 'Playground Result' with the output of the query, which is an array of documents.

```
// MongoDB Playground
// To disable this template go to Settings | MongoDB | U...
// Make sure you are connected to enable completions and
// Use Ctrl+Space inside a snippet or a string literal to
// Select the database to use.
use('mongodbVSCoDePlaygroundDB'); // If this doesnot exist
// The drop() command destroys all data from a collection
// Make sure you run it against the correct database and
db.sales.drop(); // delete your collection
// Insert a few documents into the sales collection.
db.sales.insertMany([
  { '_id': 1, 'item': 'abc', 'price': 10, 'quantity': 2,
  { '_id': 2, 'item': 'jkl', 'price': 20, 'quantity': 1,
  { '_id': 3, 'item': 'xyz', 'price': 5, 'quantity': 10,
  { '_id': 4, 'item': 'xyz', 'price': 5, 'quantity': 20,
  { '_id': 5, 'item': 'abc', 'price': 10, 'quantity': 10,
  { '_id': 6, 'item': 'def', 'price': 7.5, 'quantity': 5,
  { '_id': 7, 'item': 'def', 'price': 7.5, 'quantity': 10,
  { '_id': 8, 'item': 'abc', 'price': 10, 'quantity': 5,
]);
// Run a find command to view items sold on April 4th, 2014
db.sales.find({ date: { $gte: new Date('2014-04-04'), $lt:
// Build an aggregation to view total sales for each product
const aggregation = [
  { $match: { date: { $gte: new Date('2014-01-01'), $lt:
  { $group: { _id: '$item', totalSaleAmount: { $sum: { $
];
```

```
[
  {
    "_id": "abc",
    "totalSaleAmount": 120
  },
  {
    "_id": "jkl",
    "totalSaleAmount": 20
  },
  {
    "_id": "xyz",
    "totalSaleAmount": 150
  }
]
```

Now refresh your cluster, You will see your Database is created with the collection and documents(JSON Data).



The screenshot shows the MongoDB Atlas interface. The left sidebar displays the 'CONNECTIONS' list with 'blogcluster.3gg4i.mongodb.net' selected. The main editor area shows the 'mongodbVSCoDePlaygroundDB' database with a 'sales' collection. The 'Documents' tab is selected, showing a list of 8 documents. The right sidebar shows the 'Playground Result' with the output of the query, which is an array of documents.

```
{ "_id": 1,
  "item": "abc",
  "price": 10,
  "quantity": 2,
  "date": {
    "$date": "2014-03-01T08:00:00Z"
  }
}
```

```
[
  {
    "_id": "abc",
    "totalSaleAmount": 120
  },
  {
    "_id": "jkl",
    "totalSaleAmount": 20
  },
  {
    "_id": "xyz",
    "totalSaleAmount": 150
  }
]
```

Goto MongoDB Atlas - Browse collection - you will see your Database and collection with documents (JSON Data) here as well.



Now we can play with mongoDB playground Screen. Run queries. See the results!!

create a Database -

```
// To create a Database in MongoDB
use("LearnMongoDBDatabase");
```

Note :- Empty databases are not visible in MongoDB Atlas.

create a collection and insert one record/document -

```
// creating a collection (studentRecord) and inserting 1 record

db.studentRecord.insertOne(
  {
    name : "James Kerry",
    course : "BTech",
    branch : "CS",
    session : 2023,
    yearFee : 80000,
    feePaid : 'Yes',
    courseBooks : ['c', 'java', 'python'],
    section : "A"
  }
)
```

Inserting Many Records/documents -

```
// inserting Many Records , insertMany() is a pre-defined MongoDB method to
insert multiple records.
```

```
db.studentRecord.insertMany(
  [{
    name : "Rohit Singh",
    course : "BE",
    branch : "IT",
    session : 2023,
    yearFee : 1200000,
    feePaid : 'Yes',
    courseBooks : ['c++', 'Math', 'Fundamental'],
    section : "C"
  }, {
    name : "Neha Sharma",
    course : "MBA",
    branch : "Marketing",
    session : 2023,
    yearFee : 90000,
    feePaid : 'Yes',
    courseBooks : ['Behaviour', 'Sales Analysis', 'Market Stregthy'],
    section : "A"
  }, {
    name : "Kethy John",
    course : "CA",
    branch : "CA",
    session : 2023,
    yearFee : 70000,
    feePaid : 'No',
    courseBooks : ['income', 'Indian Tax regime', 'Indian Economy'],
    section : "B"
  }
]
);
```

MONGODB OPERATORS

MongoDB operators?

MongoDB offers different types of operators that can be used to interact with the database. Operators are special symbols or keywords that inform a compiler or an interpreter to carry out mathematical or logical operations.

The query operators enhance the functionality of MongoDB by allowing developers to create complex queries to interact with data sets that match their applications.

MongoDB offers the following query operator types:

- Comparison
- Logical
- Element
- Evaluation
- Geospatial
- Array
- Bitwise
- comments

Comparison Operators

MongoDB comparison operators can be used to compare values in a document. The following table contains the common comparison operators.

Operator	Description
\$eq	Matches values that are equal to the given value.
\$gt	Matches if values are greater than the given value.
\$lt	Matches if values are less than the given value.
\$gte	Matches if values are greater or equal to the given value.
\$lte	Matches if values are less or equal to the given value.
\$in	Matches any of the values in an array.
\$ne	Matches values that are not equal to the given value.
\$nin	Matches none of the values specified in an array.

E.G – use('StudentDB'); // Database Name.

studentData // collection Name.

```
[
  {
    "_id": {
      "$oid": "63c616e5be27db09da98859b"
    },
    "Name": "Sushant",
    "Qualification": "BTech",
    "Subjects": [
      "c",
      "java",
      "Python"
    ],
    "yearOfPassing": 2020
  },
  {
    "_id": {
      "$oid": "63c616e5be27db09da98859c"
    },
    "Name": "rohit",
    "Qualification": "BTech",
    "Subjects": [
      "c++",
      "java",
      "Python"
    ],
    "yearOfPassing": 2022
  },
  {
    "_id": {
      "$oid": "63c617ce69797359b9fd7ce5"
    },
    "Name": "randy",
    "Qualification": "BCA",
    "Subjects": [
      "Cloud",
      "java",
      "Python"
    ],
    "yearOfPassing": 2021
  }
]
```

```
},
{
  "_id": {
    "$oid": "63c61c592fde3291a01adc04"
  },
  "First_name": "Adugna",
  "Last_name": "Dentamo",
  "Objective": "I am a self motivated, highly disciplned person seeking to find",
  "Education": "Engineering",
  "School": "UNC Charlotte",
  "Graduation": 2010,
  "Experience": [
    "SQL",
    "Python",
    "Javascript",
    "MongoDb"
  ],
  "Personality": "I am friendly, easy to get along withand a team player"
},
{
  "_id": {
    "$oid": "63c7662b214be7d85dff67a2"
  },
  "Name": "alex",
  "Qualification": "MBA",
  "Subjects": [
    "bu",
    "java",
    "Python"
  ],
  "yearOfPassing": 2020
},
{
  "_id": {
    "$oid": "63c8b47c51ffcf994789b722"
  },
  "Name": "Priyanka Sing",
  "Qualification": "BBA",
  "Subjects": [
    "Engish",
    "Finance",
    "Business"
  ],
  "yearOfPassing": 2016
}
```

```

    },
    {
      "_id": {
        "$oid": "63c8b47c51ffcf994789b723"
      },
      "Name": "Priya Sharma",
      "Qualification": "Phd",
      "Subjects": [
        "Socialogy",
        "English",
        "History"
      ],
      "yearOfPassing": 2018
    }
  ]

```

\$eq

```
db.studentData.find({"Qualification" : {"$eq" : "BTech"}});
```

\$gt and \$lt Operators

In this example, we retrieve the documents where the `quantity` is greater than 5000.

```
db.inventory.find({"quantity": { $gt: 5000}}) // returns all records which has
quantity greater than 5000.
```

\$gte and \$lte Operators

Find documents with `quantity` greater than or equal to 5000.

```
db.inventory.find({"quantity": { $gte: 12000}})
```

\$in

```
db.studentData.find({ "Qualification": { "$in": ["BTech"]} })
```

returns all records which have Btech as qualification.

If you want to find documents where the price fields do not contain the given values, use the following query.

```
db.studentData.find({ "Qualification": { "$nin": ["BTech"]} })
```

```
db.studentData.find({"yearOfPassing" : {"$eq" : 2020}});  
// pretty() just format your data.  
returns the record whose id is equal to LS0009100.
```

Logical Operators

MongoDB logical operators can be used to filter data based on given conditions. These operators provide a way to combine multiple conditions. Each operator equates the given condition to a true or false value.

Here are the MongoDB logical operators:

Operator	Description
\$and	Joins two or more queries with a logical AND and returns the documents that match all the conditions.
\$or	Join two or more queries with a logical OR and return the documents that match either query.
\$nor	The opposite of the OR operator. The logical NOR operator will join two or more queries and return documents that do not match the given query conditions.
\$not	Returns the documents that do not match the given query expression.

\$and Operator

Find documents that match both the following conditions

```
db.studentData.find({ $and : [{"yearOfPassing" : 2020}, {"Qualification" : "MBA"} ]  
});
```

Nested Query.

```
db.studentData.find({ $and :  
[  
  {"yearOfPassing" : {$gte : 2018}},  
  {"yearOfPassing" : {$lte : 2022}}  
] }) ;
```

\$or and \$not Operators

Find documents that match either of the following conditions.

```
db.studentData.find({  
  $or : [  
    {"yearOfPassing" : 2020},  
    {"Qualification" : "BTech"}  
  ]  
});
```

\$nor Operator

Find documents which do not match both conditions.

```
db.studentData.find({  
  $nor : [  
    {"yearOfPassing" : 2020},  
    {"Qualification" : "BTech"}  
  ]  
});
```

\$not Operator

Find documents where they do not match the given condition.

```
db.studentData.find({"yearOfPassing" : {$not : {$eq : 2022}}});
```

// returns all records except 2022.

Element Operators

The element query operators are used to identify documents using the fields of the document. The table given below lists the current element operators.

Operator	Description
\$exists	Matches documents that have the specified field.
\$type	Matches documents according to the specified field type. These field types are specified BSON types and can be defined either by type number or alias.

\$exists Operator

Find documents where the Subjects field exists and equal to "Cloud".

```
db.studentData.find({ "Subjects": { $exists: true, $in : ["Cloud"]} });
```

```
// returns all records of students who have taken at least cloud as a subject.
```

If you pass `$exists : false`, It won't go to run the second query and return `[]`.

\$type Operator

The following query returns documents if the yearOfPassing field is an integer. If we specify a different data type, no documents will be returned even though the field exists as it does not correspond to the correct field type.

```
db.studentData.find({ "yearOfPassing": { $type: "string" } });
```

```
// returns [] empty array, no records found with given type. yearOfPassing is the number type.
```

Evaluation Operators

The MongoDB evaluation operators can evaluate the overall data structure or individual field in a document.

Operator	Description
\$regex	Select documents that match the given regular expression.
\$text	Perform a text search on the indicated field. The search can only be performed if the field is indexed with a text index.
\$where	Matches documents that satisfy a JavaScript expression.

\$regex Operator

Find documents that contain the word "BT" or "Priy" in the name field in the studentData collection.

Check your regex :- <https://regex101.com/>

```
db.studentData.find({"Qualification": {$regex: 'BT'}}) // returns 2 records.
```

```
db.studentData.find({ "Name": {$regex : /Priy/} }); // 2 records found (Priya, Priyanka).
```

```
db.studentData.find({ "Name": {$regex : /^[a-z]/} });
```

You can also write above query as below

```
db.studentData.find({ "Name" : /Priy/ } ); // 2 records found (Priya, Priyanka).
```

MongoDB - Text Search

MongoDB supports text indexes to search inside string content. The Text Search uses stemming techniques to look for specific words in the string fields by dropping stemming stop words like a, an, the, etc. At present, MongoDB supports around 15 languages.

Stemming :-

Stemming is basically removing the suffix from a word and reducing it to its root word. For example: "Flying" is a word and its suffix is "ing",

if we remove "ing" from "Flying" then we will get the base word or root word which is "Fly".

Enabling Text Search

To enable Text Search we have to **Create a Text Index** for the document we want to search.

```
db.studentData.createIndex({Name:"text"});
```

Then run the below query to get all records with the name rohit.

```
db.studentData.find({$text:{$search:"rohit"}});
```

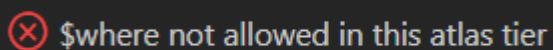
Mongodb where -

The operator is used to match and display the documents which were satisfying the condition of the JavaScript function or expression.

```
db.FbUser.find ({ $where: function() {  
    return (this.name == this.username );  
}  
});
```

another example -

Note :- The free tier (M0), M2 and M5 Instances have some limitations and command limitations, including limited metrics for analysis. Therefore while using \$where you can see the below error.

A dark rectangular box containing a red 'X' icon followed by the text "\$where not allowed in this atlas tier".

Array Operators

MongoDB array operators are designed to query documents with arrays. Here are the array operators provided by MongoDB.

Operator	Description
\$all	Matches arrays that contain all the specified values in the query condition.
\$size	Matches the documents if the array size is equal to the specified size in a query.
\$elemMatch	Matches documents that match specified \$elemMatch conditions within each array element.

\$all Operator

Find documents where the Subjects array field contains "java" and "Python" values.

```
db.studentData.find({ "Subjects": { $all: ["Python","java"]}})
```

// returns all data are case sensitive. If you write python it returns an empty array. []

\$size Operator

Find documents where the Subjects array field has three elements.

```
db.studentData.find({ "Subjects": { $size: 3}});
```

\$elemMatch Operator

The MongoDB \$elemMatch operator matches more than one component within an array element.

CustomerJuiceInterest // DB Name

```
"description" : [ {  
  
  "agegroup" : "10-18",
```

```

    "flavour" : "Mongo",

    "price" : 50

  }, {

    "agegroup" : "19-30",

    "flavour" : "Orange",

    "price" : 60

  } ]

```

```

>db.customerJuiceInterest.find( { "description": { $elemMatch: { "agegroup" :
"10-18","price" : 50}}});

```

What are push and pull operators?

Push and Pull are the two operations used to add and remove elements from arrays in MongoDB documents. Pushing and pulling elements are done using the MongoDB \$push and \$pull operators, respectively:

```

db.studentData.update({ Name : "randy"}, {$push : { City : "Mumbai"}});

```

push multiple records

```

db.studentData.update(
  {Name : "randy"},
  {
    $push : {
      Subjects : {$each : ["AWS","MongoDB"]}
    }
  }
);

```

\$pull

In MongoDB, the \$pull operator is used to remove all instances of a value from an existing array.

example 1-

```
db.studentData.update(
  {Name : "randy"},
  {
    $pull : {
      City : "Mumbai"
    }
  }
);
```

example 2-

```
db.studentData.update(
  {Name : "randy"},
  {
    $pull : {
      Subjects : {
        $in : ["MongoDB"]
      }
    }
  }
);
```

Add and Remove data inside nested document (\$set and \$unset) -

\$set - it is used for below operations.

1) In MongoDB, the \$set operator is used to update or add a field within a document. It allows you to specify new values for existing fields or create new fields if they don't already exist.

2) Creating a new field: If you want to add a new field to a document, you can use \$set along with the dot notation.

3) Modifying nested fields: With \$set, you can update or add fields within nested objects or arrays.

```
db.studentRecord.insertMany(
```

```
[
  {
    name: "Prakash Sharma",
    course: "Msc",
    branch: "Science",
    session: 2023,
    yearFee: 500000,
    feePaid: 'Yes',
    courseBooks: ['Literature', 'English', 'Geo Politics'],
    section: "B",
    address: [{
      current: {
        addressLine1: 'Abc Nagar',
        city: 'Indore',
        pincode: 213440
      },
      permanent: {
        addressLine1: 'Deccan Colony',
        city: 'Mumbai',
        pincode: 223441
      }
    }]
  },
  {
    name: "Rohit Singh",
    course: "BE",
    branch: "IT",
    session: 2023,
    yearFee: 1200000,
    feePaid: 'Yes',
    courseBooks: ['c++', 'Math', 'Fundamental'],
    section: "C",
    address: [{
      current: {
        addressLine1: 'Shiv Colony',
        city: 'Noida',
        pincode: 123453
      },
      permanent: {
        addressLine1: 'Keshav Nagar',
        city: 'Delhi',
        pincode: 223212
      }
    }]
  }
]
```

```
},
{
  name: "Sachin Kumar",
  course: "BTech",
  branch: "Civil",
  session: 2023,
  yearFee: 2400000,
  feePaid: 'Yes',
  courseBooks: ['c++', 'Math', 'Fundamental', 'Construction'],
  section: "C",
  address: [{
    current: {
      addressLine1: 'Karol Bagh',
      city: 'New Delhi',
      pincode: 411014
    },
    permanent: {
      addressLine1: 'Ram Nagar',
      city: 'Varanasi',
      pincode: 221122
    }
  }]
},
{
  name: "Anand Mehta",
  course: "BA",
  branch: "Arts",
  session: 2023,
  yearFee: 800000,
  feePaid: 'Yes',
  courseBooks: ['Literature', 'English', 'Poetry'],
  section: "A",
  address: [{
    current: {
      addressLine1: 'Kalyani Nagar',
      city: 'Indore',
      pincode: 213442
    },
    permanent: {
      addressLine1: 'Shivagi Nagar',
      city: 'Mumbai',
      pincode: 123448
    }
  }]
}
```

```
    }  
  });
```

```
db.studentRecord.updateOne(  
  { name: "Anand Mehta" },  
  { $set: { "address.0.current.state": "Madhya Pradesh" } }  
);
```

\$set -

```
db.studentData.update(  
  {Name: 'Sushant'},  
  {$set: {Name: 'Sushant Singh Rajpoot'}}  
);
```

\$unset - it is used for below operations.

In MongoDB, the **\$unset** operator is used to remove a field from a document. It effectively deletes the specified field, including its value and the field itself.

- 1) Removing a field: You can use **\$unset** to remove a specific field from a document
- 2) Removing a nested field: With **\$unset**, you can remove a field that is nested within an object or an array.
- 3) Removing multiple fields: The **\$unset** operator can be used to remove multiple fields from a document at once. You can specify multiple fields within the **\$unset** object.

e.g -

```
db.collection.updateOne({ _id: ObjectId("...") }, { $unset: { field1: "", field2: "" } });
```

```
db.studentRecord.updateOne(  
  { name: "Anand Mehta" },  
  { $unset: { "address.0.current.state": "Madhya Pradesh" } }  
);
```

SORT () -

The `sort()` method specifies the order in which the query returns the

matching documents from the given collection.

Ascending order-

```
db.studentData.find().sort({Name : 1});
```

Descending order -

```
db.studentData.find().sort({Name : -1});
```

Limit() -

In MongoDB, the limit() method limits the number of records or documents that you want.

```
db.studentData.find().limit(2);
```

Skip() -

In MongoDB, the skip() method will skip the first n document from the query result, you just need to pass the number of records/documents to be skipped.

```
db.studentData.find().skip(1);
```

DeleteOne() -

The deleteOne() method deletes the first document from the collection that matches the given selection criteria.

```
db.studentData.deleteOne({yearOfPassing : 2016});
```

```
db.studentRecord.deleteOne(  
  { _id : ObjectId('64809f03ca671adbfe406606') }  
);
```

DeleteMany() -

The deleteMany() method deletes the first document from the collection that matches the given selection criteria.

```
db.studentData.deleteMany({yearOfPassing : 2021});
```

```
db.studentRecord.deleteMany(  
  { _id : ObjectId('64809f03ca671adbfe406606') },  
  { _id : ObjectId('64809f03ca671adbfe406999') }
```



```
);
```

DELETE ANY SPECIFIC RECORD FROM COLLECTIONS

```
db.studentData.remove(  
  {Name: 'Sushant Sing Rajpoot'}  
);
```

TO DELETE COLLECTION

```
db.studentData.drop();
```

DELETE DATABASE

```
db.dropDatabase('MongoDbTestDB');
```

FIND ALL

```
db.studentData.find({});
```

MONGODB INDEX

In MongoDB, an index is a data structure that improves the speed of data retrieval operations on a collection.

Create an Index :

Along with the default index, we can create indexes on our own using the **createIndex()** method. This method creates one or more indexes on the specified collections.

Pass value 1 for ascending order and -1 for descending order

```
db.StudentRecord.createIndex({"name" : 1})
```

find -

```
db.StudentRecord.find({ $text : { $search : "Nisha Kumari" } })
```

or

```
db.StudentRecord.find({ "name" : "Rahul Jain" })
```

Types of index

MongoDB provides different types of indexes that are used according to the data type or queries. The indexes supported by MongoDB are as follows:

1. **Single field Index:** A single field index means index on a single field of a document. This index is helpful for fetching data in ascending as well as descending order.

```
db.StudentRecord.createIndex({"name" : 1})
```

find-

```
db.StudentRecord.find({ "name" : "Rahul Jain"})
```

2. **Compound Index:** We can combine multiple fields for compound indexing and that will help for searching or filtering documents in that way. Or in other words, the compound index is an index where a single index structure holds multiple references.

```
db.StudentRecord.createIndex({"branch" : 1, "section": 1})
```

find-

```
db.StudentRecord.find({"branch": "Arts", "section" : "A"})
```

3. **Multikey Index:** MongoDB uses the multikey indexes to index the values stored in arrays. When we index a field that holds an array value then MongoDB automatically creates a separate index of each and every value present in that array. Using these multikey indexes we can easily find a document that contains an array by matching the items.

```
db.StudentRecord.createIndex({"subjects" : 1})
```

Find Records -

```
db.StudentRecord.find({"subjects" : "react", "subjects" : "html", "subjects" :  
"javascript"})
```

The **dropIndex()** method drops or delete the specified index from the given collection.

```
db.StudentRecord.dropIndex({"name" : 1})
```

db.collection.explain

The `db.collection.explain()` method is used to return information on the query execution of various methods like `count()`; `find()`; `remove()`; and `update()`.

```
db.studentData.explain().find({Name : "Rohit Sharma"});
```

output -

```
Playground Result X
1  {
2    "explainVersion": "1",
3    "queryPlanner": {
4      "namespace": "MongoDbTestDB.studentData",
5      "indexFilterSet": false,
6      "parsedQuery": {
7        "Name": {
8          "$eq": "Rohit Sharma"
9        }
10     },
11     "queryHash": "EBFEE4C5",
12     "planCacheKey": "CEC3A330",
13     "maxIndexedOrSolutionsReached": false,
14     "maxIndexedAndSolutionsReached": false,
15     "maxScansToExplodeReached": false,
16     "winningPlan": {
17       "stage": "COLLSCAN",
18       "filter": {
19         "Name": {
20           "$eq": "Rohit Sharma"
21         }
22       },
23       "direction": "forward"
24     },
25     "rejectedPlans": []
26   },
27   "command": {
28     "find": "studentData",
29     "filter": {
30       "Name": "Rohit Sharma"
31     },
32     "$db": "MongoDbTestDB"
33   },
34   "serverInfo": {
35     "host": "blogcluster-shard-00-02.3gg4i.mongodb.net",
36     "port": 27017,
37     "version": "5.0.14",
38     "gitVersion": "1b3b0073a0b436a8a502b612f24fb2bd572772e5"
39   },
40   "serverParameters": {
41     "internalQueryFacetBufferSizeBytes": 104857600,
42     "internalQueryFacetMaxOutputDocSizeBytes": 104857600,
43     "internalLookupStageIntermediateDocumentMaxSizeBytes": 16793600,
44     "internalDocumentSourceGroupMaxMemoryBytes": 104857600,
45     "internalQueryMaxBlockingSortMemoryUsageBytes": 33554432,
46     "internalQueryProhibitBlockingMergeOnMongoS": 0,
```

Assignment

Insert below data in MongoDB atlas and write following queries

data - <https://fakestoreapi.com/products>

queries -

1) sort by price

Ans - `db.productList.find().sort({"price" : 1});`

2) find all products which price is greater than 100

```
Ans - db.productList.find({"rating.count" : {$gt : 100}});
```

3) search by title

```
Ans - db.productList.createIndex( { title: "text" } );  
db.productList.find($text : {$search : 'T-Shirts'}));
```

4) update particular product count by using id

```
Ans - db.productList.updateOne(  
  { "id": 6},  
  { $set: { "rating.count": 200 }}  
)
```

5) find products which price in between 100 to 500.

```
Ans - db.productList.find(  
  "price": {  
    $gt: 100,  
    $lt: 500  
  }  
)
```

6) search by category

```
Ans - db.productList.find({"category" : {"$eq" : "men's clothing"}});
```

7) search by complete title (exact same title)

```
Ans - db.productList.find( { "title": "Fjallraven 1 Backpack, Fits 15  
Laptops" } );
```

8) find by rating greater than 3

```
Ans - db.productList.find({"rating.rate": 3});
```

9) sort by rating greater than 3 in ascending order

```
Ans - db.productList.find({"rating.rate" : {"$gte": 3  
}}).sort({"rating.rate": 1});
```

MONGODB COMPASS

MongoDB Compass Download

Easily explore and manipulate your database with Compass, the GUI for MongoDB. Intuitive and flexible, Compass provides detailed schema visualizations, real-time performance metrics, sophisticated querying abilities, and much more.

Please note that MongoDB Compass comes in three versions: a full version with all features, a read-only version without write or delete capabilities, and an isolated edition, whose sole network connection is to the MongoDB instance.

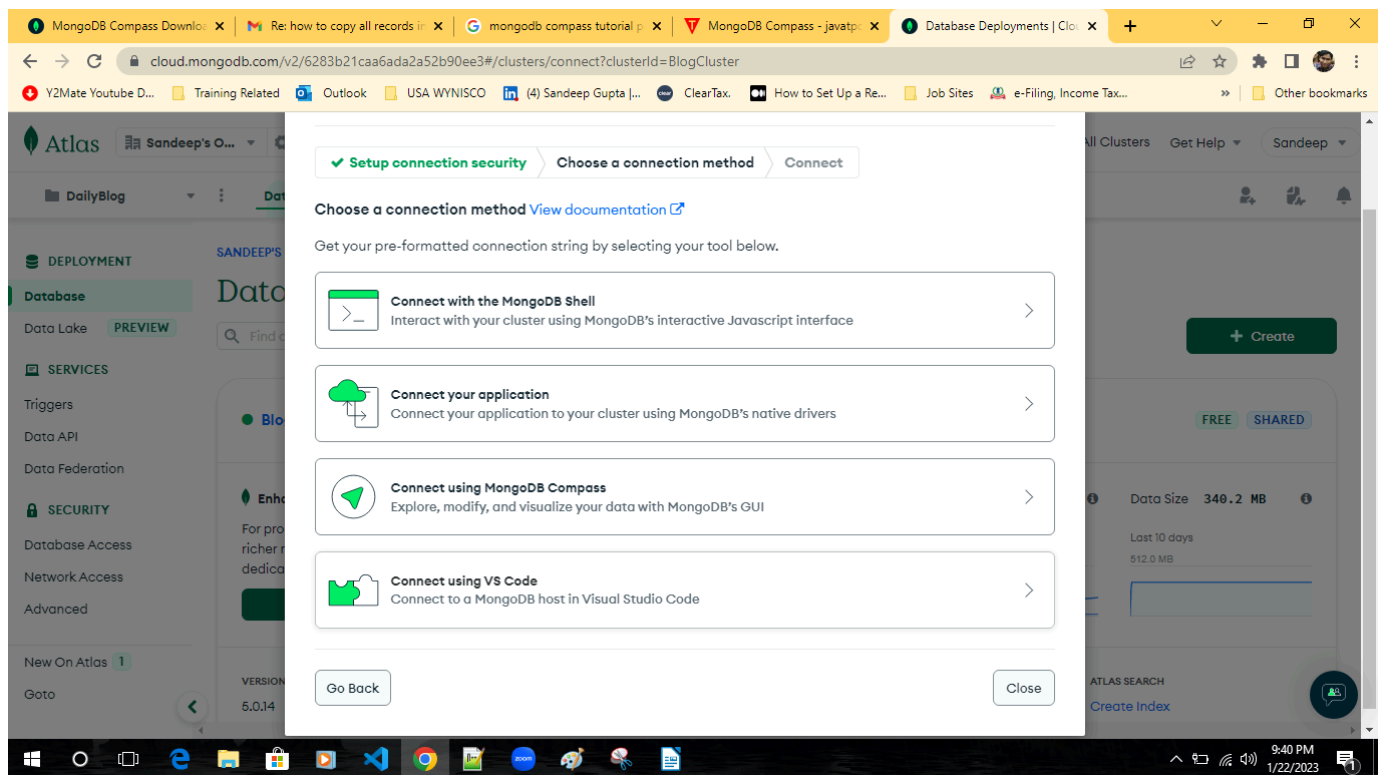
For more information, see our [documentation pages](#).

Compass

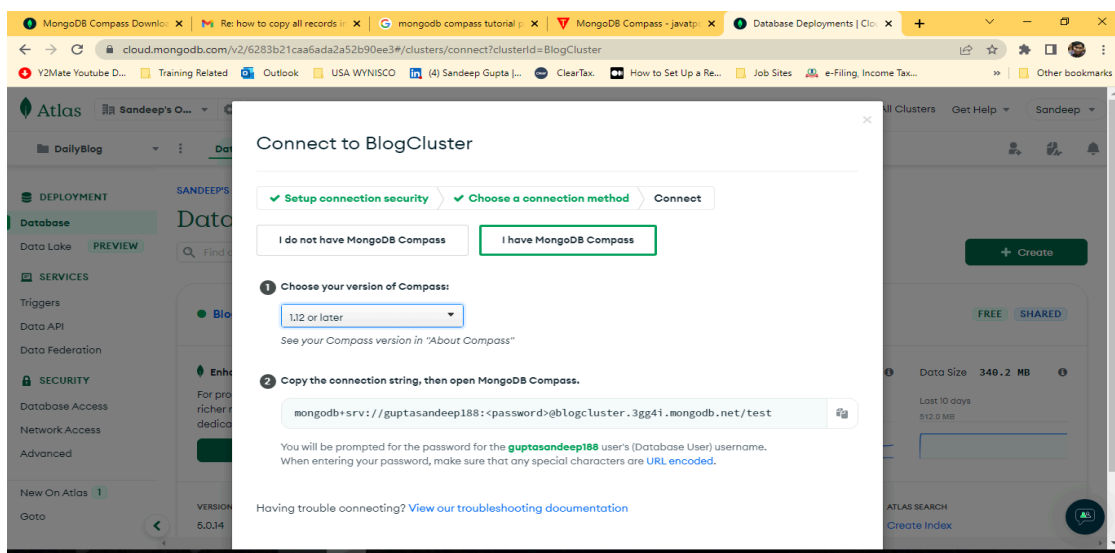
The full version of MongoDB Compass, with all features and capabilities.

Readonly Edition

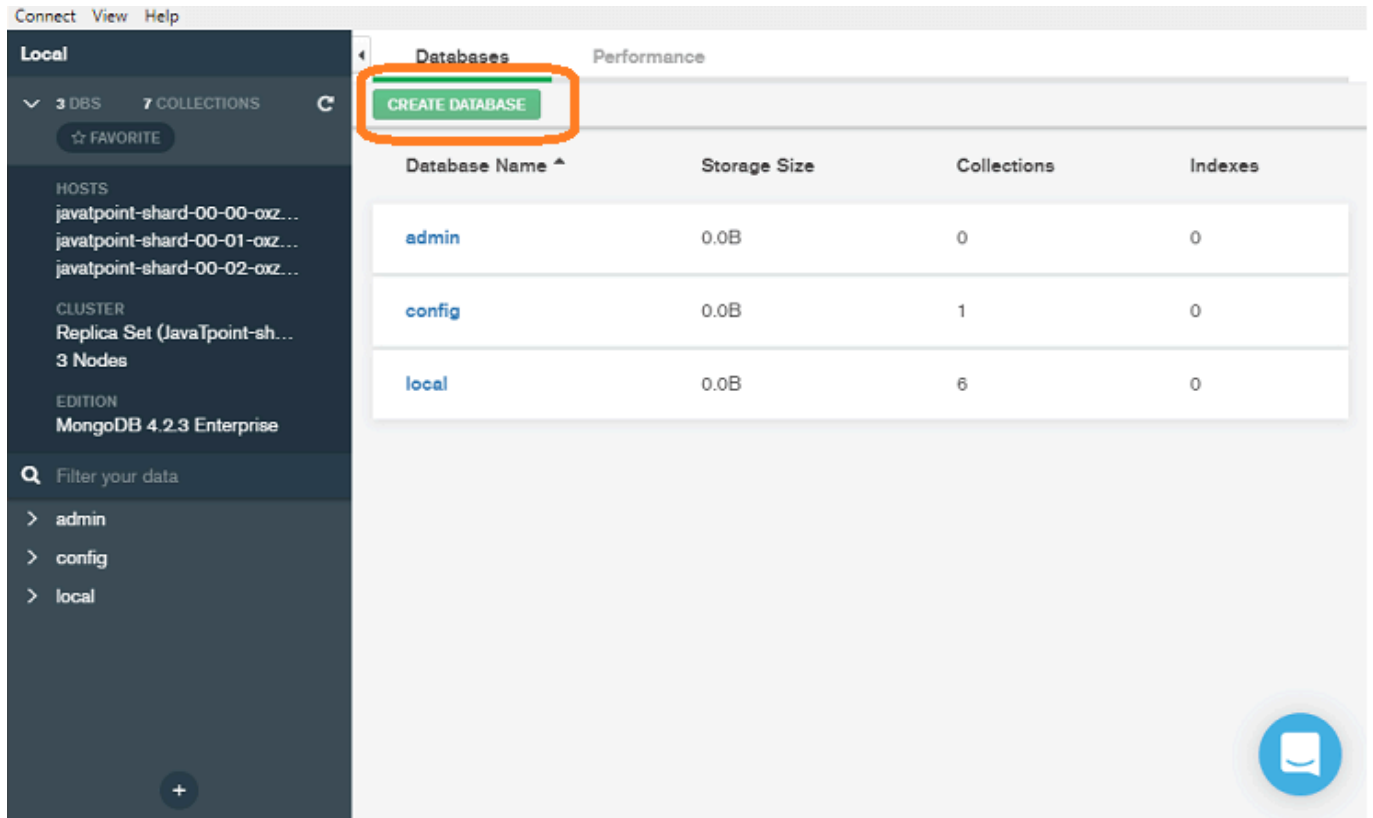
Step 2 -



Step 3 -



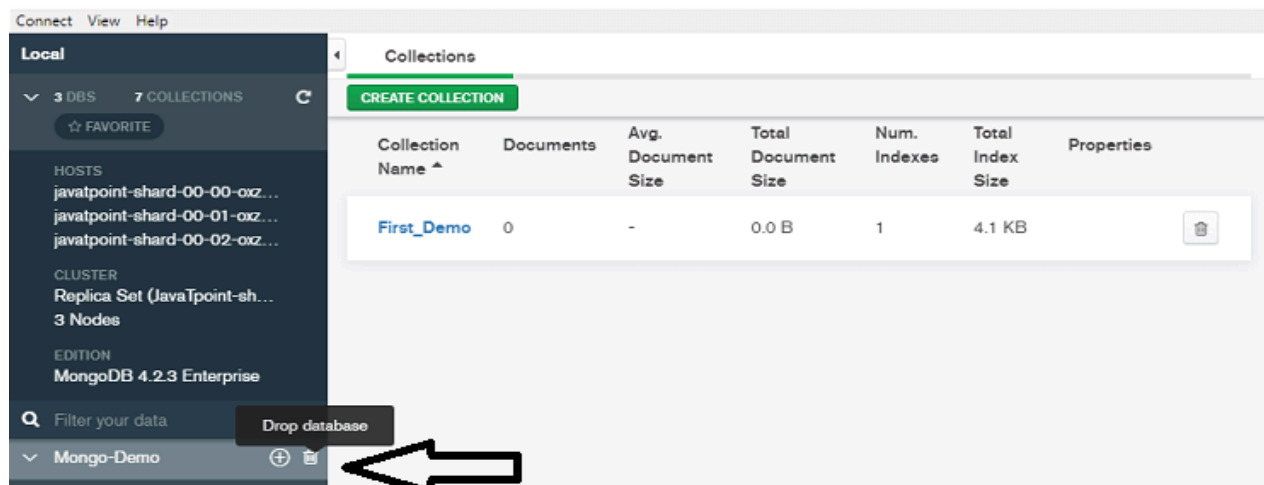
Step 4 - You can create a local DB or use Existing Atlas DB.




The screenshot shows the MongoDB Compass interface. On the left sidebar, under 'Local', it shows '3 DBS' and '7 COLLECTIONS'. The main panel is titled 'Databases' and has a 'CREATE DATABASE' button highlighted with an orange rectangle. Below this, a table lists the databases:

Database Name ^	Storage Size	Collections	Indexes
admin	0.0B	0	0
config	0.0B	1	0
local	0.0B	6	0

Delete DB



The screenshot shows the MongoDB Compass interface. On the left sidebar, under 'Local', it shows '3 DBS' and '7 COLLECTIONS'. The main panel is titled 'Collections' and has a 'CREATE COLLECTION' button. Below this, a table lists the collections:

Collection Name ^	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
First_Demo	0	-	0.0 B	1	4.1 KB	

A black arrow points to the 'Drop database' button in the bottom left corner of the interface.

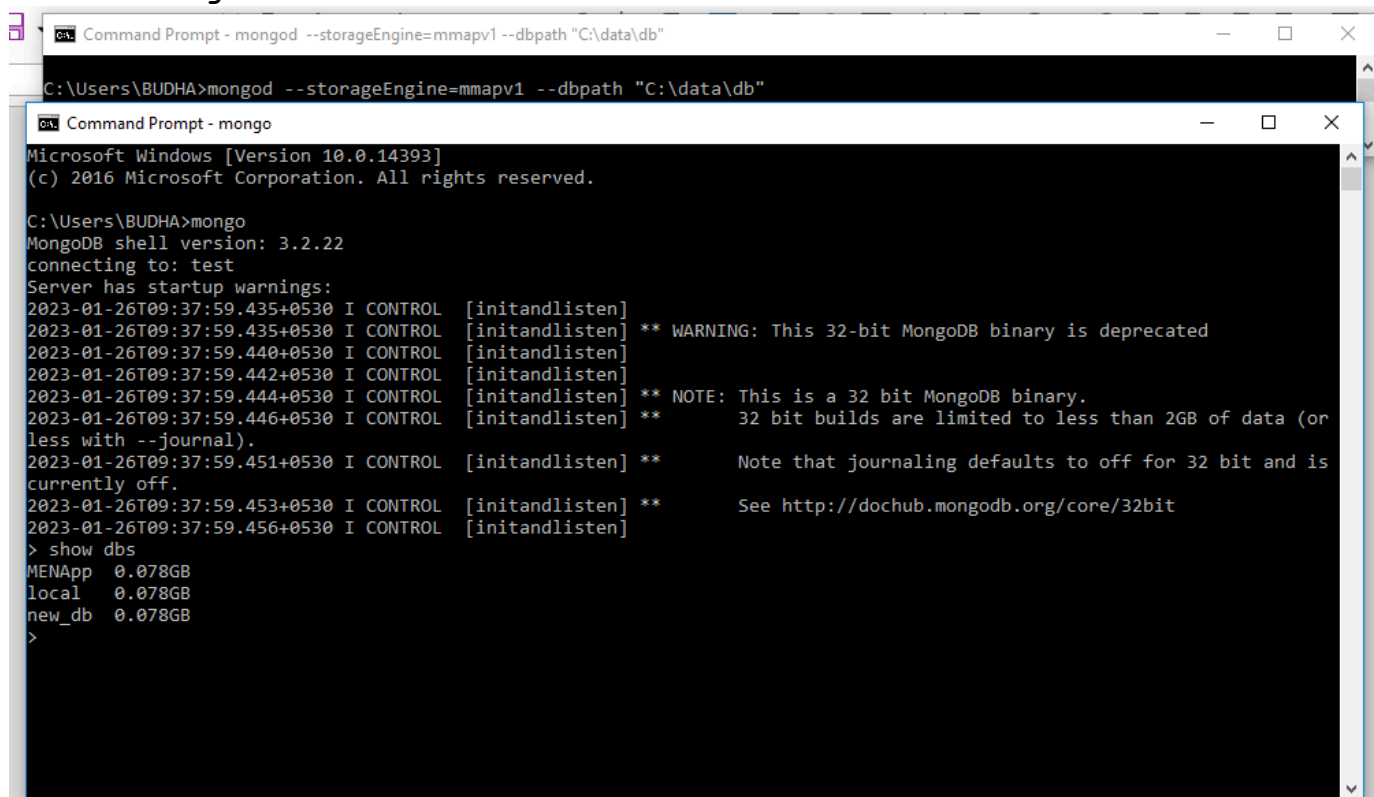
MONGODB SHELL COMMANDS

Steps -

MongoDB Shell Use Steps

1. install MongoDB Community Server Download from (<https://www.mongodb.com/try/download/community>)
2. add server installation path till bin folder(copy from program files)in environment variables(select system variable option)
4. now unzip MongoDB Shell
5. move to c program files
6. add this path till bin folder in environment variable(select system variable option)
8. go to c drive -> create **data** folder -> create **db** folder
- 9 cmd -> run mongod (to start mongodb server) keep this window active.
10. run mongo - -version
7. now open mongo.exe (mongodb shell exe) and press enter.
8. you are connected, now run show dbs to see default databases.

To run mongo commands first you have to open a new command prompt and run below command (for 32 bit) to keep the database active and in running state.



```
Command Prompt - mongod --storageEngine=mmapv1 --dbpath "C:\data\db"
C:\Users\BUDHA>mongod --storageEngine=mmapv1 --dbpath "C:\data\db"

Command Prompt - mongo
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\BUDHA>mongo
MongoDB shell version: 3.2.22
connecting to: test
Server has startup warnings:
2023-01-26T09:37:59.435+0530 I CONTROL [initandlisten]
2023-01-26T09:37:59.435+0530 I CONTROL [initandlisten] ** WARNING: This 32-bit MongoDB binary is deprecated
2023-01-26T09:37:59.440+0530 I CONTROL [initandlisten]
2023-01-26T09:37:59.442+0530 I CONTROL [initandlisten]
2023-01-26T09:37:59.444+0530 I CONTROL [initandlisten] ** NOTE: This is a 32 bit MongoDB binary.
2023-01-26T09:37:59.446+0530 I CONTROL [initandlisten] ** 32 bit builds are limited to less than 2GB of data (or
less with --journal).
2023-01-26T09:37:59.451+0530 I CONTROL [initandlisten] ** Note that journaling defaults to off for 32 bit and is
currently off.
2023-01-26T09:37:59.453+0530 I CONTROL [initandlisten] ** See http://dochub.mongodb.org/core/32bit
2023-01-26T09:37:59.456+0530 I CONTROL [initandlisten]
> show dbs
MENApp 0.078GB
local 0.078GB
new_db 0.078GB
>
```

Open new command prompt and run below command

```
> mongod --storageEngine=mmapv1 --dbpath "C:\data\db" (mongod for 64 bit)
```

Again Open new command prompt and run below command

```
> mongo
```

STEP BY STEP

TO CONNECT WITH MONGODB

```
>mongod (for 64 bit OS)
```

TO RUN MONGO COMMAND

```
>mongo
```

TO SEE ALL EXISTING DATABASES

```
>show dbs
```

TO SELECT ANY DATABASE

```
>use xyz_db
```

TO SEE COLLECTIONS/TABLES

```
>show collections
```

TO CREATE COLLECTIONS/TABLES

```
>db.createCollection("xyz_employees")
```

TO DELETE COLLECTION

```
>db.collection_name.drop()
```

TO DELETE DATABASE

```
>db.dropDatabase()
```

TO SEE LIST OF DB COMMANDS

```
>db.help()
```

TO CREATE A COLLECTIONS/TABLES AND INSERT DOCUMENT/RECORD

```
>db.xyz_employees2.insertOne({"first_name": "James"}) // it will create a collection xyz_employees2 and then insert the object.
```


TO CREATE COLLECTIONS/TABLES WITH FIXED DOCUMENT/RECORD OF MAX SIZE.

```
>db.createCollection('logs', {capped: true, size: 2048, max: 2})
>db.logs.insertOne({'count': 1})
>db.logs.insertOne({'count': 2})
>db.logs.insertOne({'count': 3})
```

ADDING NEW FIELD TO EXISTING RECORD

```
> db.cars.update({name: 'Ford'}, {$set: {quantity: 400}}, {$upsert: true})
```

INSERT NESTED RECORDS

```
> db.employee.update({'name': 'Virat'}, {$set: {'address': {'street': 'abc street'}}}, {$upsert: true})
```

DELETE ANY SPECIFIC RECORD FROM COLLECTIONS

```
> db.employee.remove({name: 'Virat' })
```

SEE SPECIFIC RECORD ONLY

```
>db.employee.find({name: 'Priya'}).pretty()
```

FIND SPECIFIC RECORD GIVING SOME PARAMETER

```
>db.employee.find( { "monthlySalary": { $elemMatch: { "name" : "Priya"}}}).pretty();
```

FIND ALL

```
>db.employee.find({}).pretty()
```

SEARCH QUERIES

SEARCH BY TEXT

To search by text, you need to add an index as text.

```
>db.employee.createIndex({name: 'text'})
>db.employee.find({$text: {$search: "\"Priya\""}}).pretty()
```

RENAME ALL FIELDS

```
> db.employee.update({},{$rename: {'salary': 'monthlySalary'}})
```

The SET Modifier in MongoDB?

If the value of a field does not yet exist, the "\$set" sets the value. This can be useful for updating schemas or adding user-defined keys.

Example:

```
> db.users.findOne()
{
  "_id" : ObjectId("4b253b067525f35f94b60a31"),
  "name" : "alice",
  "age" : 23,
  "gender" : "female",
  "location" : "India"
}
```

To add a field to this, we use "\$set":

```
> db.users.updateOne({"_id" :
```

```
ObjectId("4b253b067525f35f94b60a31")),  
... {"$set" : {"favorite book" : "Start with Why"}})
```