MongoDB

MongoDB is an open source NoSQL database management program. **NoSQL (Not only SQL)** is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.

MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.

> Structured Query Language (SQL) is a standardized programming language that is used to manage relational databases. SQL normalizes data as schemas and tables, and every table has a fixed structure.

Instead of using tables and rows as in relational databases, as a NoSQL database, the MongoDB architecture is made up of collections and documents.

Collections are equivalent of SQL tables, contain document sets. Documents are made up of key-value pairs -- MongoDB's basic unit of data.

Difference between SQL and NoSQL

SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have a dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Examples: MySQL, PostgreSQL, Oracle, MS-SQL Server, etc	**Examples:** MongoDB, GraphQL, HBase, Neo4j , Cassandra, etc

MongoDB vs. RDBMS: What are the differences?

A relational database management system (RDBMS) is a collection of programs and capabilities that let IT teams and others create, update, administer and otherwise interact with a relational database. RDBMS store data in the form of tables and rows. RDBMS most commonly uses SQL.

One of the main differences between MongoDB and RDBMS is that RDBMS is a relational database while MongoDB is nonrelational. Likewise, while most RDBMS systems use SQL to manage stored data, MongoDB uses BSON for data storage -- a type of NoSQL database.

While RDBMS uses tables and rows, MongoDB uses documents and collections. In RDBMS a table -- the equivalent to a MongoDB collection -- stores data as columns and rows. Likewise, a row in RDBMS is the equivalent of a MongoDB document but stores data as structured data items in a table. A column denotes sets of data values, which is the equivalent to a field in MongoDB.

RDBMS	**MongoDB**
Database	**Database**
Table	**Collection** - stores data as columns and rows.
Row	**Document** - stores data as structured data items
Column	**Field** - denotes sets of data values
Primary Key	Primary Key (Default key _id provided by MongoDB itself)

MongoDB is also better suited for hierarchical storage.

Why is MongoDB used?

An organization might want to use MongoDB for the following:

- **Storage**. MongoDB can store large structured and unstructured data volumes and is scalable vertically and horizontally. Indexes are used to improve search performance. Searches are also done by field, range and expression queries.
- **Data integration.** This integrates data for applications, including for hybrid and multi-cloud applications.
- **Complex data structures descriptions.** Document databases enable the embedding of documents to describe nested structures (a structure within a structure) and can tolerate variations in data.
- **Load balancing.** MongoDB can be used to run over multiple servers.

Features of MongoDB

Features of MongoDB include the following:

- **Replication.** A replica set is two or more MongoDB instances used to provide high availability. Replica sets are made of primary and secondary servers. The primary MongoDB server performs all the read and write operations, while the secondary replica keeps a copy of the data. If a primary replica fails, the secondary replica is then used.

- **Scalability.** MongoDB supports vertical and horizontal scaling. Vertical scaling works by adding more power to an existing machine, while horizontal scaling works by adding more machines to a user's resources.
- **Load balancing.** MongoDB handles load balancing without the need for a separate, dedicated load balancer, through either vertical or horizontal scaling.
- **Schema-less.** MongoDB is a schema-less database, which means the database can manage data without the need for a blueprint.

  > **Schema-less databases are a type of NoSQL database that do not require a predefined schema to store data. Instead, they allow data to be stored in flexible and dynamic formats, such as JSON documents, key-value pairs, graphs, or columns.**

- **Document**. Data in MongoDB is stored in documents with key-value pairs instead of rows and columns, which makes the data more flexible when compared to SQL databases.

Above Scalability feature of mongoDB is explained in detail.

Scaling

Scaling alters the size of a system. In the scaling process, we either compress or expand the system to meet the expected needs. The scaling operation can be achieved by adding resources to meet the smaller expectation in the current system, by adding a new system to the existing one, or both.

Scaling can be categorized into 2 types:

- **Vertical Scaling:**

When new resources are added to the existing system to meet the expectation, it is known as vertical scaling.

Consider a rack of servers and resources that comprises the existing system. (as shown in the figure). Now when the existing system fails to meet the expected needs, and the expected needs can be met by just adding resources, this is considered vertical scaling. Vertical scaling is based on the idea of adding more power(CPU, RAM) to existing systems, basically adding more resources.

Vertical scaling is not only easy but also cheaper than Horizontal Scaling. It also requires less time to be fixed.

- **Horizontal Scaling:**

When new server racks are added to the existing system to meet the higher expectation, it is known as horizontal scaling.

Now when the existing system fails to meet the expected needs, and the expected needs cannot be met by just adding resources, we need to add completely new servers. This is considered horizontal scaling. Horizontal scaling is based on the idea of adding more machines to our pool of resources. Horizontal scaling is difficult and also costlier than Vertical Scaling. It also requires more time to be fixed.

Advantages of MongoDB

MongoDB offers several potential benefits:

- ✓ **Schema-less.** Like other NoSQL databases, MongoDB doesn't require predefined schemas. It stores any type of data. This gives users the flexibility to create any number of fields in a document, making it easier to scale MongoDB databases compared to relational databases.
- ✓ **Document-oriented.** One of the advantages of using documents is that these objects map to native data types in several programming languages., Having embedded documents also reduces the need for database joins, which can lower costs.
- ✓ **Scalability.** A core function of MongoDB is its horizontal scalability, which makes it a useful database for companies running big data applications. In addition, sharding lets the database distribute data across a cluster of machines. MongoDB also supports the creation of zones of data based on a shard key.
- ✓ **Third-party support.** MongoDB supports several storage engines and provides pluggable storage engine APIs that let third parties develop their own storage engines for MongoDB.

Disadvantages of MongoDB

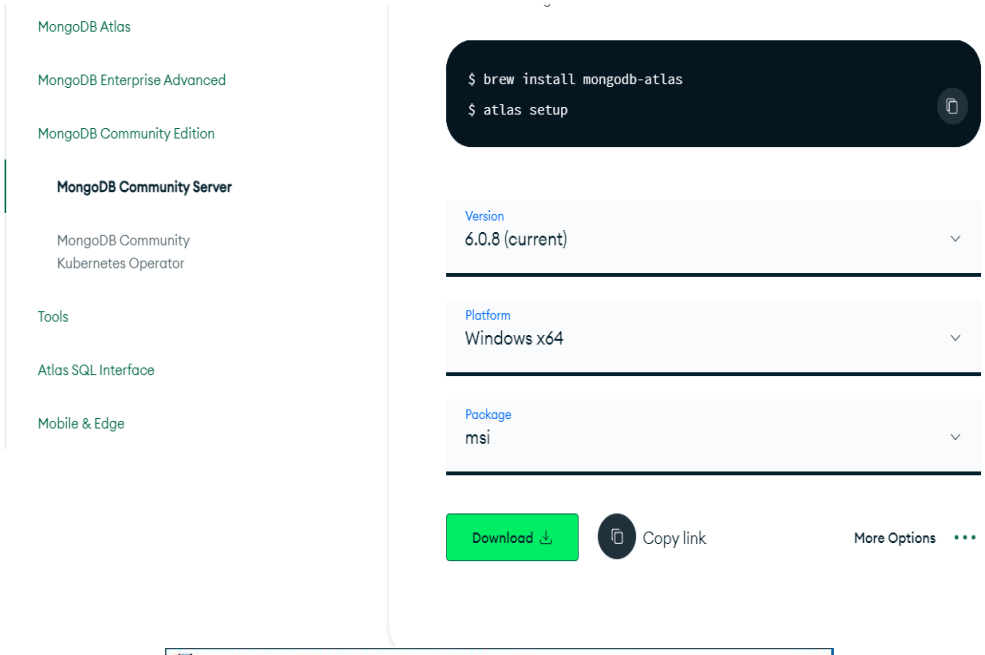
Though there are some valuable benefits to MongoDB, there are some downsides to it as well.

- ✓ **Continuity.** With its automatic failover strategy, a user sets up just one master node in a MongoDB cluster. If the master fails, another node will automatically convert to the new master. This switch promises continuity, but it isn't instantaneous -- it can take up to a minute.
- ✓ **Data consistency.** MongoDB doesn't provide full referential integrity through the use of foreign-key constraints, which could affect data consistency.
- ✓ **Security**. In addition, user authentication isn't enabled by default in MongoDB databases. However, malicious hackers have targeted large numbers of unsecured MongoDB systems in attacks, which led to the addition of a default setting that blocks networked connections to databases if they haven't been configured by a database administrator.

How to install and setup MongoDB

Step 1:

- ✓ Go to https://www.mongodb.com/
- ✓ Under the **Products** tab
  - Community Edition
    - Community Server
    - Select Package
    - Package (msi)
    - Download
- ✓ Run and install



Step 2:

Mongo Shell

The mongo shell is an interactive JavaScript interface to MongoDB. You can use the mongo shell to query and update data as well as perform administrative operations.

To download

- ✓ Go to https://www.mongodb.com/try/download/shell
- ✓ Download
- ✓ Extract files and from **bin** folder **Copy exe** and **dll** files to "**C:\Program Files\MongoDB\Server\6.0\bin**"

Step 3:

Set path property in environment variable

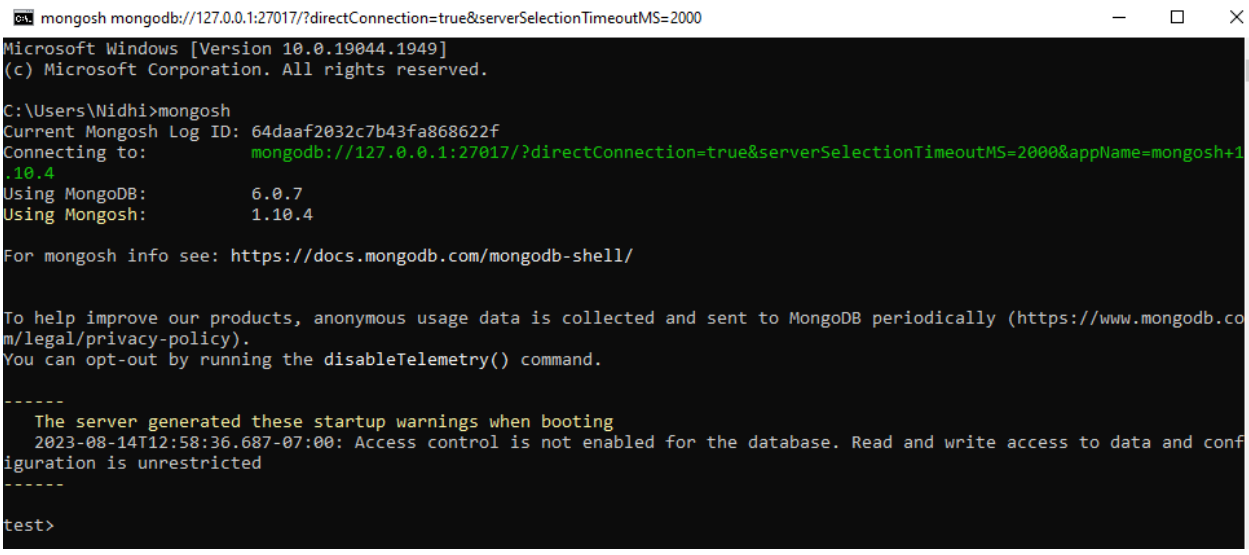
- ✓ Select "System Variable" > Path
- ✓ Click on "Edit"
- ✓ Add "C:\Program Files\MongoDB\Server\6.0\bin"

Step 4:

To check > open Command Prompt

Type **mongosh**

It will shown as below screenshot.



- ✓ **Default new database**
  - **Test>**

- ✓ **To list all the available DBs write below command**
  - **test> show dbs**

- admin 40.00 KiB
- config 72.00 KiB
- local 76.00 KiB

✓ **test>db**
- To show current active database.

✓ **test>show collections**
- Display all available collections in existing database

✓ **To create/use database**
- **test> use mydb**
  - **mydb>**

❖ Insert documents

✓ **insertOne()**

To create collection with only one document
- **mydb>**db.student.insertOne({name:"Test",rollno:23})
  - **Un above example student is collection(table).**
  - **Two fields(column) are "name" and "rollno"**
  - **1st document(row) data are "Test" and "23" for "name" and "rollno" respectively.**

✓ **insertMany()**

To create collection with more than one documents
- **mydb>**db.student.insertMany([{name:"N1",age:20},{name:"N2",age:24}])

Output:

{ _id: ObjectId("64dfd8ffc925fb0136d77817"), name: 'N1', age: 20 },
{ _id: ObjectId("64dfd8ffc925fb0136d77818"), name: 'N2', age: 24 }

✓ **We can also store non-uniform document fields as shown in below example.**
- **mydb>**db.student.insertMany([
  {name:"N1",age:20,status:"Active"},
  {name:"N2",age:24},
  {name:"N3",age:27,status:"Active",city:"Ahmedabad"}
  ])

So here documents contain 3,2 4 fields respectively.

❖ find()

Find document/Read document
- Mydb>db.student.find()
  - Display all documents of **student** collection

Syntax

db.student.find (query/filter/condition, projection)

Query/filter/condition : Optional. Specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document

Projection Optional. Specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter. ({}).

Example: with only query

- **mydb> db.student.find({name:"N1"})**
  - As an output it will give all the documents with name N1 and also display all the fields as shown below.

Output :
[{ _id: ObjectId("64de65b454282c021d807835"), name: 'N1', age: 20 }]

Example: with query and projection

- **mydb> db.student.find({name:"N1"},{_id:0,age:false })**
  - As an output it will give all the documents with name N1 and displays only Name field as shown below

Output:
[{ name: 'N1' }]

Note: To display field write "true" or 1

To not display field write "false" or 0

✓ **findOne**()

- findOne() method returns only one document that satisfies the criteria entered. If the criteria entered matches for more than one document, the method returns only one document according to natural ordering, which reflects the order in which the documents are stored in the database.
  - **Db.student.findOne({name:"N1"})**

✓ **Limit()**

- The limit() method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want. Or in other words, this method uses on cursor to specify the maximum number of documents/ records the cursor will return.
- For example, N1 name exists in 4 documents and we want to display limited number of documents then we can use **limit** method

- Suppose, we have written 1 in limit then it will display only 1st document of 4 documents.
  - **Db.student.find({name:"N1"}).limit(1)**
    Or we can use findOne() to display only 1st document
  - Db.student.findOne({name:"N1"})

Displays only first document of collection

✓ **skip()**

- Call the skip() method on a cursor to control where MongoDB begins returning results. This approach may be useful in implementing paginated results.
- Suppose we have 3 documents with name "N1". Below command will skip 1st document and give only 2nd document as an output

  - **db.student.find({name:"N1"}).limit(1).skip(1)**

Output:
[{ _id: ObjectId("64de6dbb54282c021d807837"), name: 'N1', age: 23 }]

❖ Update documents

✓ **updateOne() and updateMany()**

Syntax

db.collection.updateOne(filter,document,options)
Updates only one document where filter is matched
db.collection.updateMany(filter, document, options)
Updates only all documents where filter is matched

Parameters:

1. **filter**: The selection criteria for the update, same as find() method.
2. **document**: A document or pipeline that contains modifications to apply.
3. **options**: Optional. May contains options for update behavior. It includes upsert etc.

- ```
  db.student.updateOne(
  {name:"N1"},
  {$set:
      {
          name:"N4"

      }

  })
  ```

  Updates only one document with **name N4** where **name is N1**
  ```

- **db.student.updateOne({name:"N1",age:23},{$set:{name:"N4"}})**
  - Updates only one document with name N4 where name is N1 and age is 23

- **db.student.updateOne({name:"N1",age:23},{$set:{name:"N4",age:11}})**
  - Updates only one document with name N4 and age 11 where name is N1 and age is 23

- **db.student.updateMany({name:"N1"},{$set:{name:"N4"}})**
  - Updates all documents with name N4 and where name is N1

✓ **Upsert**

- In MongoDB, upsert is an option that is used for update operation e.g. update(), findAndModify(), etc. Or in other words, upsert is a combination of update and insert (update + insert = upsert).
- If the value of this option is set to true and the document or documents found that match the specified query, then the update operation will update the matched document or documents.
- Or if the value of this option is set to true and no document or documents matches the specified document, then this option inserts a new document in the collection and this new document have the fields that indicate in the operation.
- By default, the value of the upsert option is false. If the value of upsert in a sharded collection is true then you have to include the full shard key in the filter.

Syntax: upsert: <boolean>

  - The value of upsert option is either true or false. By default it is false.

- **db.student.updateOne({age:45},{$set:{name:"PQR"}},{upsert:true})**

In above example it will find document with age 45 in collection. If condition matched then update the name else insert new document with mentioned fields.

Note: Update will only update the values if the document is already exist with mentioned filter value. **If document does not exist in collection with mentioned filter value and we want to add document with the mentioned fields then we have to add 3rd parameter upsert:true.**

❖ Delete Documents

deleteOne() and deleteMany()

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.

  - **db.student.deleteOne({name:"N1"})**
    - This deletes only 1st document where name is "N1"

- ✓ **deleteMany()**

  The deleteMany() method will delete all documents that match the query provided.

  - **db.student.deleteMany({})**
    - This deletes all documents of collection

  - **db.student.deleteMany({name:"N1"})**
    - This deletes all the documents where name is "N1"

❖ db.createCollection()

MongoDB db.createCollection(name, options) is used to create collection.

Syntax
Basic syntax of createCollection() command is as follows −

```
db.createCollection(name, options)
```

In the command, name is name of collection to be created. Options is a document and is used to specify configuration of collection.

db.createCollection("student")

❖ The drop() Method

For Collection

MongoDB's db.collection.drop() is used to drop a collection from the database.

Syntax
Basic syntax of drop() command is as follows −

```
db.COLLECTION_NAME.drop()
```

Example
First, check the available collections into your database mydb.

>use mydb

switched to db mydb

>show collections

mycollection

user

student

Now drop the collection with the name **mycollection**.

```
>db.mycollection.drop()
true
```

Again check the list of collections into database.

>show collections

user

student

For Database

>use lju

switched to db lju

lju > db.dropDatabase()

{ ok: 1, dropped: 'lju' }

This will drop one database named “lju”

❖ renameCoolection()

Call the db.collection.renameCollection() method on a collection object, to rename a collection. Specify the new name of the collection as an argument.
For example:

```
db.student.renameCollection("students")
```

This renames “student” collection to “students”.

❖ Count()

db.collection.count(query)

Returns the count of documents that would match a find() query. The db.collection.count() method does not perform the find() operation but instead counts and returns the number of results that match a query.

The db.collection.count() method has the following parameter:

The db.collection.count() method is equivalent to the db.collection.find(query).count() construct.

db.people.count({uname:"N1"})

It will give an aswer with below warning.

DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.

or

db.people.find({uname:"N1"}).count()

As an output it will give 3 if uname "N1" exists in 3 documents

❖ sort()

Specifies the order in which the query returns matching documents. You must apply sort() to the cursor before retrieving any documents from the database.

The sort parameter contains field and value pairs, in the following form:

{ field: value }

Example:

To display all documents in descending order of age.

db.student1.find().sort({age:-1})

Ascending: 1 and Descending: -1