## MongoDB:

**MongoDB** is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents.

Documents consist of key-value pairs which are the basic unit of data in MongoDB

Collections contain sets of documents and function which is the equivalent of relational database tables

## MongoDB Features

* 1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
  2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
  3. The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
  4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

1. Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database

## Why Use MongoDB?

Below are the few of the reasons as to why one should start using MongoDB

1. Document-oriented – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.
2. Ad hoc queries - MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
3. Indexing - Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
4. Replication - MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
5. Load balancing - MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

## Installation and usage.

To begin using MongoDB, connect a [mongo.exe](https://docs.mongodb.com/manual/reference/program/mongo/#mongodb-binary-bin.mongo) shell to the running MongoDB instance. Either:

* From Windows Explorer/File Explorer, go to C:\Program Files\MongoDB\Server\4.4\bin\ directory and double-click on [mongo.exe](https://docs.mongodb.com/manual/reference/program/mongo/#mongodb-binary-bin.mongo).
* Or, open a Command Interpreter with Administrative privileges and run:

|  |
| --- |
| "C:\ProgramFiles\MongoDB\Server\4.4\bin\mongo.exe" |
|  |

Install mongodb database tools:

Starting with MongoDB 4.4, the MongoDB Database Tools are now released separately from the MongoDB Server and use their own versioning

<https://docs.mongodb.com/database-tools/installation/installation/>

**Commands**

|  |  |
| --- | --- |
|  |  |
| Connect to database | C:\Program Files\MongoDB\Server\4.4\bin>mongo |
| Show databases | show dbs |
| Import existing collections | mongoimport --db cooker recipes.json --jsonArray |
| Displays all the collections available in database | Show collections |
| Use database | Use databasename |
| Get the database connected to | db.getName(); |
| Insert record | db.directions.insertOne(doc)  directions is collection name, doc is the Bson which need to be inserted |
| Select docs from collection | db.directions.find() |
| Select records in formatted form | db.directions.find().pretty() |
| Select records based on column value(where condition) | db.myreciepes.find({"title" : "Chicken hard Tacos"}).pretty() |
| Select based on multiple columns | db.myreciepes.find({"title" : "Chicken hard Tacos" , "cook\_time":10} ).pretty() |
| Select only specific fields- this will get only title column data | db.myreciepes.find({"title" : "Chicken hard Tacos"} , {"title" :1} ).pretty() |
| Exclude column- this will not fetch title column | db.myreciepes.find({"title" : "Chicken hard Tacos"} , {"title" :0} ).pretty() |
| This will get all the rows from table and only column title | db.myreciepes.find({} , {"title" :1} ).pretty() |
| This query is like string contains . this will return records containing title as Chicken hard. I is for case insensitive | db.myreciepes.find({"title" : {$regex :/Chicken hard/i}} , {"title" :0} ).pretty() |
| Will display all the collection names | show collections |
| Find the count of documents in collection | db.recipes.find().count() |
| To fetch only limited count of documents | db.recipes.find({},{"title":1}).limit(3); |
| Sorting the documents (ASC) | db.recipes.find({},{"title":1}).sort({"title":1}); |
| Reverse sorting(DESC) | db.recipes.find({},{"title":1}).sort({"title": -1}); |
| It will skip the one record | db.recipes.find({},{"title":1}).sort({"title":-1}).skip(1); |
| Query to fetch less than greater than record | db.recipes.find({"cook\_time" : {$lte : 30}} , {"title" : 1})  $lte – less than or equal to  $gt - Greater than  $lt – Less than |
| Or query | db.recipes.find({ $or : [{"cook\_time" : {$lte : 30} , "prep\_time" : {$lte:10}}]} , {"title" : 1}) |
| Tags is a array column. This query fetch array contains easy. | db.recipes.find({"tags" : "easy"} , {"title":1 , "tags":1}) |
| Tags is a array column. This query fetch array contains easy and quick. | db.recipes.find({"tags" : {$all : ["easy", "quick"]}} , {"title":1 , "tags":1}) |
| Tags is a array column. This query fetch array contains easy or quick. | db.recipes.find({"tags" : {$in : ["easy", "quick"]}} , {"title":1 , "tags":1}) |
| Documents with inner documents use. to get values | db.recipes.find({ "ingredients.name" : "egg"} , {title:1}) |
| $set is used to update the values. here title Pizza is updated with Thin crust Pizza | db.examples.updateOne({"title" : "Pizza"} , {$set : {"title" : "Thin crust Pizza"}}); |
| This will remove the column or field vegan from document | db.examples.updateOne({"title":"Thin crust Pizza"} , {$unset : {"vegan" :1}}) |
| This will increment likes\_count field | db.examples.updateOne({"title" : "Tacos"} , { $inc : {"likes\_count" : 1}}); |
| Add element to existing array – use $push | db.examples.updateOne({"title" : "Tacos"} , {$push : {"likes" : 60}}); |
| Remove element from existing array – use pull | db.examples.updateOne({"title" : "Tacos"} , {$pull : {"likes" : 60}}); |
| Delete a record | db.examples.deleteOne({"\_id" : ObjectId("5ee69e393260aab97ea0d58e")}); |
| Gives detail info what happened when executing the query | db.recipes.find({"cook\_time" :10} , {"title" : 1}).explain("executionStats"); |
| Create index | db.collection.createIndex( { “title”: -1 } ) |
| Create User | db.createUser({ user : "taco" , pwd : passwordPrompt() , roles: [{role:"userAdminAnyDatabase",db:"admin"},"readWriteAnyDatabase"]}); |
| Shut down db | db.adminCommand({shutdown :1}) |
| Start server | mongod |
| Connect with user | mongo --authenticationDatabase "admin -u "toco" -p |
| It will the db for writing into data. While taking backup we need o run this command so that no one will insert into this | db.fsyncLock() |
| Reverse of lock | db.fsyncUnLock() |
| It will dump the datatbase to folder | mongodump |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# **Collections**

A collection is a grouping of MongoDB [documents](https://docs.mongodb.com/compass/current/documents/#std-label-compass-documents). Documents within a collection can have different fields. A collection is the equivalent of a table in a relational database system. A collection exists within a single [database](https://docs.mongodb.com/compass/current/databases/#std-label-database-tab)

# **Documents**

each record in a MongoDB [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection) is document. Documents are a structure composed of file and value pairs, similar to JSON objects or other *mapping* data types

[JSON](https://docs.mongodb.com/manual/reference/glossary/#term-json) is a way of representing objects derived from JavaScript; however, many programming environments have support for converting JSON objects into native mapping types. Documents in MongoDB are [BSON](https://docs.mongodb.com/manual/reference/glossary/#term-bson), which is a binary data format that is *like* JSON, but includes additional type data.

**BSON**

BSON is a binary serialization of [JSON](http://json.org/)-like documents. BSON stands for “Binary JSON”, but also  contains extensions that allow representation of data types that are not part of JSON

 In a single MongoDB document, you can store up to 16MB binary data. However, MongoDB has its own file system GridFS, which stores binary files larger than 16MB in

BSON records tend to be littler than JSON records, which is the most reason for utilizing its interior MongoDB.

<https://www.geeksforgeeks.org/difference-between-json-and-bson/>



The value of a field can be any of the BSON [data types](https://docs.mongodb.com/manual/reference/bson-types/), including other documents, arrays, and arrays of documents.

BSOn datatypes: <https://docs.mongodb.com/manual/reference/bson-types/>

|  |
| --- |
| var mydoc = { |
| \_id: ObjectId(**"5099803df3f4948bd2f98391"**), |
| name: { first: **"Alan"**, last: **"Turing"** }, |
| birth: new Date(**'Jun 23, 1912'**), |
| death: new Date(**'Jun 07, 1954'**), |
| contribs: [ **"Turing machine"**, **"Turing test"**, **"Turingery"** ], |
| views : NumberLong(1250000) |
| } |

### Field Names

[Documents](https://docs.mongodb.com/manual/core/document/) have the following restrictions on field names:

Top-level field names **cannot** start with the dollar sign ($) character.

Otherwise, starting in MongoDB 3.6, the server permits storage of field names that contain dots (i.e. .) and dollar signs (i.e. $).

MongoDB uses the dot notation to access the elements of an array and to access the fields of an embedded document.

# **Indexes**

Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.

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

# **Authnetication & Authorization**

aa