Mongo DB 🍃

INTRODUCTION:

Mongo DB is an open-source document-oriented database that id designed to store a large scale of data and also allows you to work with data efficiently. It is categorized under the NoSQL database because the storage and retrieval of data in the MongoDB are not in the form of tables.

The MongoDB database is developed and managed by MongoDB.Inc under SSPL (server side public license) and initially released in February 2009.It also provides official driver support for all the popular languages like C, C++, C#, and .Net, Go, Java, Node.js, Python, etc.,. Some of the companies that used MongoDB like Facebook, Nokia, eBay, Adobe, Google, etc. to store their large amount of data.

Why and where you should use mongo DB:

Since, MongoDB is a NoSQL database, so we need to understand when and why we need to use this type of database in the real-life applications. In normal circumstances, MongoDB always preffered by the developers when our main concern is the deal with large volume of data with a high performance. If we want to insert thousand of records in a second, then the MongoDB is the best choice for that. MongoDB is good for some of the situations like:

- ➢ E-Commerce type of product-based applications.
- ➢ Blog and content management system.
- ➢ Need to maintain location wise Geospatial data.
- ➢ For maintains data related to the social and networking types.

Comparison between MySQL and MongoDB:

MySQL	MongoDB
Database	Database
Table	Collection
Index	Index
Row	BSON Document
Column	BSON Field
Primary key	Primary key
Group By	Aggregation

WHAT IS DATABASE:

Structured Data:
The information is typically organized in a specific format, often using tables with rows and columns. This makes it easier to search, filter, and analyze the data.

Database Management System(DBMS):
This is the software that acts like the filling cabinet manager. It allows you to store, retrieve, update, and manage all the data within the database.

Data Types:
Database can hold various kinds of information, including text , numbers, images, videos, and more.

The above diagram is the best example for MongoDB and MySQL. In this if someone ask for a Paracetamol capsule then the shop owner search easily if they are arranged in correct order , if he don't now about the tablets he can easily find it seeing the arrangements and it is more faster it is possible incase of MySQL. But in MongoDB there is no arrangement they are messed up and it will consume more time and it make delay to search any data because they are in unorganized order.

Advantages and Limitations of MongoDB:
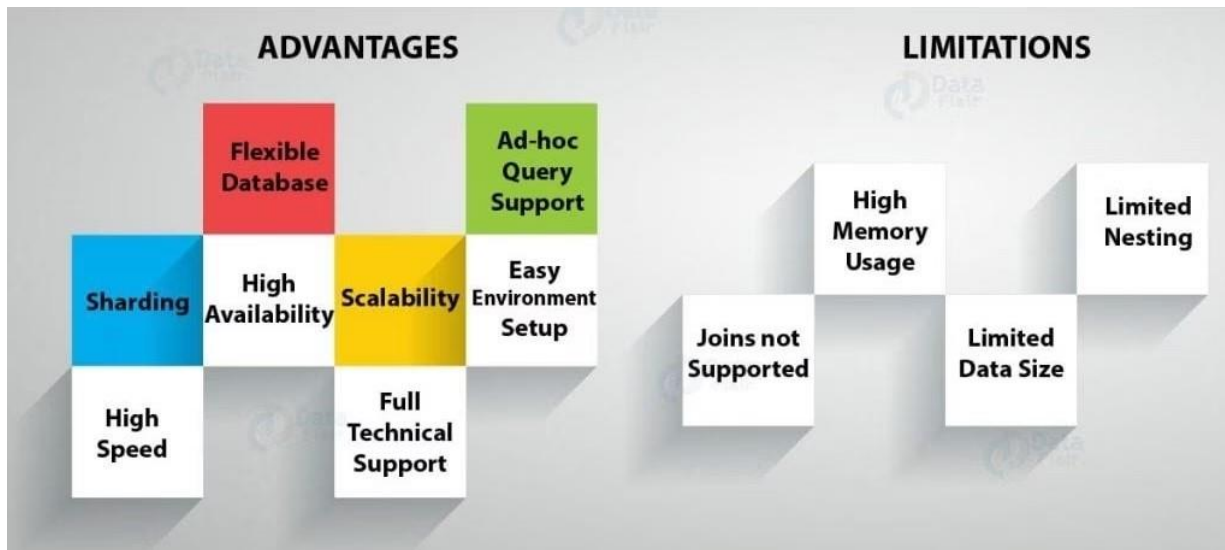
Few Commands to test after connections:

Command	Notes
Show dbs	All Database are shown
Use db	Connect and use db
show collections	Show all tables
db.boo.insert({"car" :"bus"})	Insert a record to collection. create collection if not
db.boo.find()	Print all rows
db.boo.remove()	Remove the table

Installation of MongoDB :

- Mongo Shell download [link](#)
- All the work is expected to do it in mongo shell not in mongo compass.

- Install [Studio3T](#) .connect to mongodb://localhost:27017

Add, Update and Delete Data:

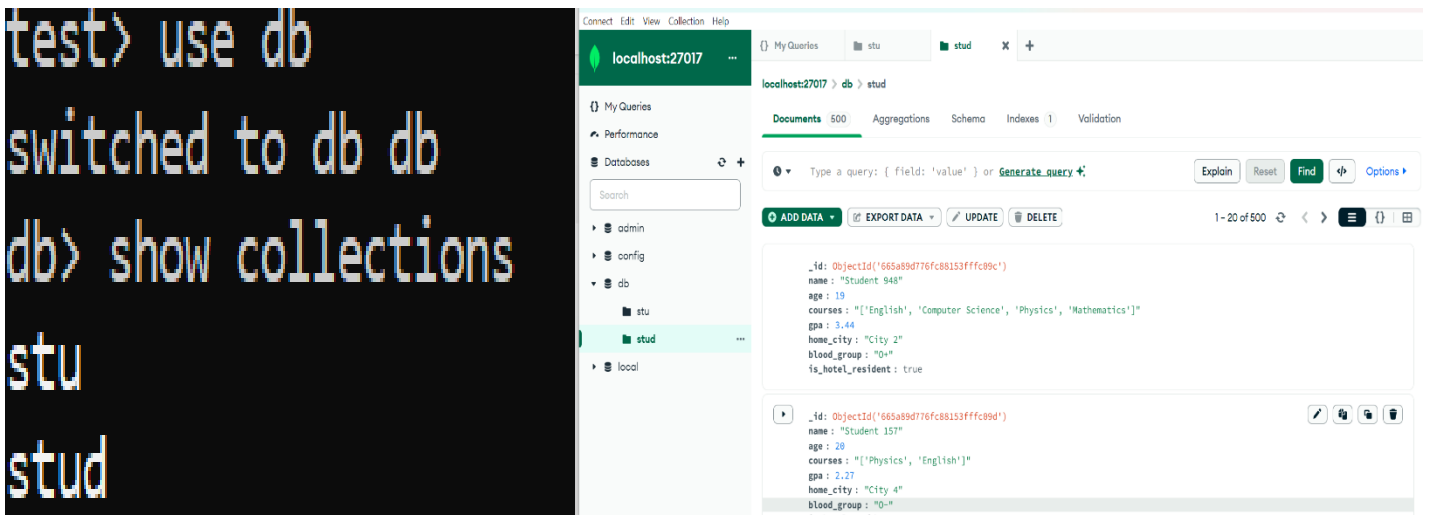First step is we want to switch our database to the given collection by using command.

"use db"

```
test> use db
switched to db db
```

Now the database is switched to db.

To find whether the data present in the given collection, here the collection name is about the information of students. we can use the command

"Show collections"

In the above example the collection name is stud or stu.

To find the total number of collection of the database use the command.

"db.stud.find().count()"

```
db> db.stud.find().count()
500
```

It shows the total collections of students in the database.

To find the collection of the database use the command.

"db.stud.find()"

```
db> db.stud.find()
[
  {
    _id: ObjectId('665a89d776fc88153fffc09c'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09d'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09e'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.32,
    blood_group: 'B+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc09f'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
```

Collections:

A collection is a group of documents. If a document is the MongoDB analog of a row in a relational database, then a collection can be thought of as the analog to a table.

Database:

MongoDB groups collections into databases. A single instance of MongoDB can host several databases, each grouping together zero or more collections.

WHERE AND OR & CRUD:

- ## WHERE:

Given a collection you want to filter a subset based on a condition. That is the place WHERE is used.

To find all students with GPA greater than 3.5, we use command-

"db.stud.find({gpa:{$gt:3.5}});"

```
db> db.stud.find({gpa:{$gt:3.5}});
[
  {
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a2'),
    name: 'Student 268',
    age: 21,
    courses: "['Mathematics', 'History', 'Physics']",
    gpa: 3.98,
db> db.stud.find({gpa:{$gt:3.5}});
[   is_hotel_resident: false
  {,
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',9d776fc88153fffc0ac'),
    age: 25,tudent 368',
```

Here $gt represent the greater than and it gives the information that are belongs to greater than 3.5 gpa.

- ## AND:

Given a collection you want to filter a subset based on multiple conditions.

To find all students who live in "City 5" AND have a based group of "A+"

Here we use the command:

Db.stud.find({

$and: [

{home_city : "City 5"},

{blood_group: "A+"}

]

});

```
db> db.stud.find({
... $and:[
... {home_city:"City 5"},
... {blood_group:"A+"}
... ]
... });
[
  {
    _id: ObjectId('665a89d776fc88153fffc0d3'),
    name: 'Student 142',
    age: 24,
    courses: "['History', 'English', 'Physics', 'Computer Science']",
    gpa: 3.41,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('665a89d776fc88153fffc1f3'),
    name: 'Student 947',
    age: 20,
    courses: "['Physics', 'History', 'English', 'Computer Science']",
db>
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc265'),
    name: 'Student 567',
    age: 22,
    courses: "['Computer Science', 'History', 'English', 'Mathematics']",
    gpa: 2.01,
    home_city: 'City 5',
    blood_group: 'A+',
    is_hotel_resident: true
  }
]
```

Above example is filtered based on some conditions like: 'home_city: City5' and

'blood_group : A+'.

OR:

In the given collection that is student we want to filter a subset based on multiple conditions but any one is sufficient.

```
db.stud.find({ $or: [ { blood_group: "A+" }, { gpa: { $gt: 3.5 } }] })

_id: ObjectId('665a89d776fc88153fffc0a0'),
name: 'Student 930',
age: 25,
courses: "['English', 'Computer Science', 'Mathematics', 'History']",
gpa: 3.63,
home_city: 'City 3',
blood_group: 'A-',
is_hotel_resident: true
,
_id: ObjectId('665a89d776fc88153fffc0a2'),
name: 'Student 268',
age: 21,
courses: "['Mathematics', 'History', 'Physics']",
gpa: 3.98,
blood_group: 'A+',
is_hotel_resident: false
,
_id: ObjectId('665a89d776fc88153fffc0a7'),
name: 'Student 177',
age: 23,
courses: "['Mathematics', 'Computer Science', 'Physics']",
gpa: 2.52,
home_city: 'City 10',
blood_group: 'A+',
is_hotel_resident: true
,
_id: ObjectId('665a89d776fc88153fffc0ac'),
name: 'Student 368',
age: 20,
courses: "['English', 'History', 'Physics', 'Computer Science']",
gpa: 3.91,
```

In the above example , the stud database is filtered based on either 'blood_group : A+'

or 'gpa greater than 3.5'. ($gt: greater than).

- ## CURD:

C-Create/Insert

R-Remove

U-Update

D-Delete

This is applicable for a collection or a document.

- ## Insert:

To insert the data into collection we use the following command:

Const studentData={

"name": "Jam",

"age":12,,

"courses":["CS", "MATHS", "KANNADA"],

"gpa":3.2,

"home_city": "City 4",

"blood_group": "AB+",

"is_hotel_resident":true

}

```
lb> const studentData={
.. "name": "Jam",
.. "age" :12,
.. "courses" :["CS","Maths","Kannada"],
.. "gpa":3.2,
.. "home_city":"City 4",
.. "blood_group": "AB+",
.. "is_hotel_resident" : true
.. }
```

In the above example we are inserting the student details name 'Jam' and other information to

The collection of database called studentData.

- **Update:**

Here we can update any data that are present in the collections.

To update we use '$set' command.

```
db> db.students.updateOne( { name:"Sam"} , {$set:{
gpa:3} } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
db>
```

- **Delete:**

The delete operation is used to delete the data present in the given collection.

```
db> db.students.deleteOne({ name:"Sam" })
{ acknowledged: true, deletedCount: 1 }
db>
```

- **Projection:**

This is used when we don't need all columns or attributes.

```
db> db.students.deleteOne({ name:"Sam" })
{ acknowledged: true, deletedCount: 1 }
db> db.students.find({} , {name:1 , gpa:1 })
[
  {
    _id: ObjectId('66587b4a0a3749dfd07d78a0'),
    name: 'Student 948',
    gpa: 3.44
  },
  {
    _id: ObjectId('66587b4a0a3749dfd07d78a1'),
    name: 'Student 157',
    gpa: 2.27
  },
  {
    _id: ObjectId('66587b4a0a3749dfd07d78a2'),
    name: 'Student 316',
    gpa: 2.32
```

In the above example it shows only the name and gpa, because the command is

given as 'name:1' and 'gpa:1'.

Benefits of Projection:

➢ Reduced data transferred between the database and your application.

➢ Simplifies your code by focusing on the specific information you need.

➢ Improve query performance by retrieving only necessary data.

LIMIT AND SELECTORS:

- **Limit:**

The limit operator is used with the find method.It's chained after the filter criteria or any sorting operations.

Syntax:

db.collection.find({filter},

{projection}).limit(number)

```
Type "it" for more
db> db.students.find({} , {_id:0}).limit(5)
[
  {
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
```

To get only first five document we use limit(5).

- **Selectors:**
  ➢ Comparison gt and lt
  ➢ AND operator
  ➢ OR operator.

Comparison gt lt:

To find all students with age greater than 20.

```
db> db.stud.find({age:{$gt:20}});
[
  {
    _id: ObjectId('665a89d776fc88153fffc09f'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a0'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665a89d776fc88153fffc0a1'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
```

AND operator:

To find students from "city 2" with blood group "B+".

```
db.stud.find({
$and:[
{home_city: "City 2"},
{blood_group: "B+" }
]
});

_id: ObjectId('665a89d776fc88153fffc0b4'),
name: 'Student 504',
age: 21,
courses: "['Physics', 'Computer Science', 'English', 'Mathematics']",
gpa: 2.42,
home_city: 'City 2',
blood_group: 'B+',
is_hotel_resident: true
,
_id: ObjectId('665a89d776fc88153fffc0eb'),
name: 'Student 367',
age: 19,
courses: "['English', 'Physics', 'History', 'Mathematics']",
gpa: 2.81,
home_city: 'City 2',
blood_group: 'B+',
is_hotel_resident: false
,
```