MONGODB

INTRODUCTION:

What is MongoDB?

MongoDB is a non-relational database system. There are two primary database types: SQL (relational) and NoSQL (non-relational). Relational databases store data in columns and rows. Organizations like Microsoft SQL Server Oracle and Sybase use the relational database management system (RDBMS).
MongoDB has cultivated a reputation as a versatile, flexible database and is currently used today as the backend data store of many high-profile businesses and organizations such as Forbes, Facebook, Google, IBM, Twitter, and many more.

Why MongoDB?

There are three major reasons why MongoDB is being deployed more often. They are:

- Flexibility: MongoDB uses documents that can contain sub-documents in complex hierarchies making it expressive.
- Flexible Query Model: The user can selectively index some parts of each document or a query based on regular expressions, ranges, or attribute values, and have as many properties per object as needed by the application layer.
- Native Aggregation: It allows users to extract and transform data from the database. The data can either be loaded into a new format or exported to other data sources

Benefits of MongoDB:

→ NoSQL databases are cheaper and easier to maintain. NoSQL databases have features like easier data distribution, simpler data models, and automatic repair. These benefits require less administrative costs and, consequently, are less expensive.

→ It's easily and highly scalable. Since NoSQL databases like MongoDB expand horizontally, you can scale by adding more machines to your resource pool.

→ MongoDB has no schema hassles. You can place data into a NoSQL database without requiring a predefined schema, so you can change the data model and formats without disrupting applications.

→ It's user-friendly. MongoDB offers plenty of useful features (Ad-hoc queries, aggregation, capped collection, file storage, indexing, load balancing, replication, server-side JavaScript execution) that makes it a user-friendly database.

MongoDB Applications:

1.Internet of Things
2.Time Analysis

Limitations of MongoDB:

1.MongoDB uses high memory for data storage.
2.The BSON document size cannot exceed 16MB.
3.Naming restrictions for databases in Windows.

INSTALLATION OF MONGODB

Here is the link for the process of installation link
Application link link

WHAT IS DATABASE

Structured data: The data organized in a specific format.This makes it easier to search the required data.

Database Management System: This is the software that acts like a filing cabinet manager.It allows to store,manage,update the data.

Data types: Databases can hold the various kind of information.

Different between My SQL and Mongodb

MY SQL	MongoDB
Database	Database
Table	Collections
Index	Index
Row	BSON Document
Column	BSON Field
Join	Embedded Documents
Primary Key	Primary Key
Group By	Aggregation


MYSQL
A BOWL FULL OF LEMONS
PANTRY ORGANIZATION
MONGODB

ADD,UPDATE AND DELETE

COMMANDS FOR MONGODB

Command	Expected Output	Notes
show dbs	admin 40.00 KiB config 72.00 KiB db 128.00 KiB local 40.00 KiB	All Databases are shown
use db	switched to db db	Connect and use db
show collections	Students	Show all tables
db.foo.insert({"bar" : "baz"})		Insert a record to collection. Create Collection if not exists

Command	Notes
db.foo.batchInsert([{"_id" : 0}, {"_id" : 1}, {"_id" : 2}])	Insert more than one document
db.foo.find()	Print all rows
db.foo.remove()	Remove foo table

DOCUMENTS,COLLECTION AND DATABASE

DOCUMENTS

The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.

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.
Given a Collection you want to FILTER a subset based on multiple conditions

DATABASE

MongoDB groups collections into databases.
A database has its own permissions, and each database is stored in separate files on disk.
A good rule of thumb is to store all data for a single application in the same database.

DATA TYPE

Basically each document will be in JSON format which will be as follows.
Where each attributes inside can be of multiple data types

```
{
    "name" : "John Doe",
    "address" : {
        "street" : "123 Park Street",
        "city" : "Anytown",
        "state" : "NY"
    }
}
```

WHERE,AND,OR AND CRUD

WHERE:

Given a Collection that want to FILTER a subset based on a condition. That is the place WHERE is used.

```
db> db.students.find({home_city:"City 4"});
[
  {
    _id: ObjectId('666852df0851d739a08f8ba5'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
```

AND:

Given a Collection you want to FILTER a subset based on multiple conditions.

```
db> db.students.find({ $and: [ {home_city:"City 2"}, { blood_group: "O+" }] });
[
  {
    _id: ObjectId('666852df0851d739a08f8ba4'),
    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
```

OR:

Given a Collection you want to FILTER a subset based on multiple conditions but Any One is Sufficient

```
db> db.students.find({ $or: [ {is_hotel_resident:false},{gpa:{$lt:3.0}}]});
[
  {
    _id: ObjectId('666852df0851d739a08f8ba5'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
```

CRUD:

C - Create / Insert
R - Remove
U - update
D - Delete

Insert:

```
const studentData = {
  "name": "Alice Smith",
  "age": 22,
  "courses": ["Mathematics", "Computer Science", "English"],
  "gpa": 3.8,
  "home_city": "New York",
  "blood_group": "A+",
  "is_hotel_resident": false
};
```

Update:

```
db> db.students.updateOne({name:"Alice Smith"},{$set:{gpa:3.6}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

Delete:

```
db> db.student.deleteOne({name:"Alice"});
{ acknowledged: true, deletedCount: 1 }
```

Update Many:

```
db> db.student.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.4}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

Delete Many:

```
db> db.students.deleteMany({is_hotel_resident:true});
{ acknowledged: true, deletedCount: 246 }
```

Projection:

```
db> db.students.find({},{name:1, gpa:2});
[
  {
    _id: ObjectId('666852df0851d739a08f8baa'),
    name: 'Student 268',
    gpa: 3.98
  },
```

Projection & Limit

Projection:

Use the projection document as the second argument to the find method.
Include field names with a value of 1 to specify fields to be returned.
Omit fields or set them to 0 to exclude them from the results.

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)

```
db> db.students.find({gpa:{$lt:3.0}}).limit(3);
[
  {
    _id: ObjectId('666852df0851d739a08f8bab'),
    name: 'Student 563',
    age: 18,
    courses: "['Mathematics', 'English']",
    gpa: 2.25,
    blood_group: 'AB+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666852df0851d739a08f8bad'),
    name: 'Student 536',
    age: 20,
    courses: "['History', 'Physics', 'English', 'Mathematics']",
    gpa: 2.87,
    home_city: 'City 3',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('666852df0851d739a08f8bb5'),
    name: 'Student 172',
    age: 25,
    courses: "['English', 'History', 'Physics', 'Mathematics']",
    gpa: 2.46,
    home_city: 'City 3',
    blood_group: 'A+',
    is_hotel_resident: false
```

Geospatial:

MongoDB offers a rich set of geospatial query operators, including $geoNear, $geoWithin, $geoIntersects, and $nearSphere.
These operators enable developers to perform complex geospatial queries.

```
db> db.places.insertOne({
...    name: "Central Park",
...    location: { type: "Point", coordinates: [ -73.97, 40.77 ] }
... });
{
  acknowledged: true,
  insertedId: ObjectId('66688975cd8c29a400cdcdf6')
}
```