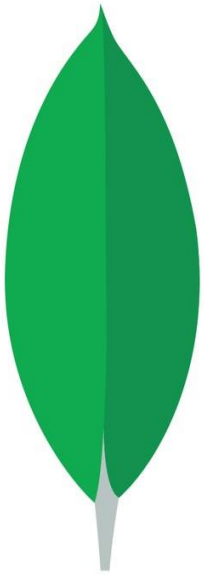

mongoDB®

INTRODUCTION

What is **MongoDB?**

MongoDB is a source-available, cross-platform, document-oriented database program.
Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas .

What is database?

- A database is an organized collection of data stored in a computer system and usually controlled by a database management system (DBMS).
- The data in common databases is modeled in tables, making querying and processing efficient.

Structured Data:

- Structured data refers to data that is organized and formatted in a specific way to make it easily readable and understandable by both humans and machines.
- This is typically achieved through the use of a well-defined schema or data model, which provides a structure for the data.

Database Management System:

- A database management system (DBMS) is system software for creating and managing databases.
- A DBMS makes it possible for end users to create, protect, read, update and delete data in a database.

SET UP:

https://www.geeksforgeeks.org/how-to-install-mongodb-on-windows/?ref=ml_lbp

Few Commands to test after connections

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, Collections And Datatypes

Document:

A **document** is a fundamental unit of data storage. It's a record that contains field-and-value pairs, similar to a JSON object.

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.

{"greeting" : "Hello, world!"}

Collections:

A collection is a grouping of MongoDB documents. Each document within a collection can have different fields. They are analogous to tables in relational databases.

Database:

MongoDB groups collections into databases.

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

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.

Datatype:

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 & CRUD

WHERE

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

```
// Find all students with GPA greater than 3.5
db.students.find({ gpa: { $gt: 3.5 } });

// Find all students from "City 3"
db.students.find({ home_city: "City 3" });
```

OUTPUT

OR

The $or operator is used to specify a compound query with multiple conditions, where at least one condition must be satisfied for a document to match.

```
// Find all students who are hotel residents OR have a GPA less than 3.
db.students.find({
  $or: [
    { is_hotel_resident: true },
    { gpa: { $lt: 3.0 } }
  ]
});
```

OUTPUT

AND

The $and operator allows you to specify multiple conditions that documents must satisfy to match the query.

```
// Find all students who live in "City 5" AND have a blood group of "A+"
db.students.find({
  $and: [
    { home_city: "City 5" },
    { blood_group: "A+" }
  ]
});
```

OUTPUT

CRUD

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

This is applicable for a Collection (Table) or a Document (Row)

INSERT

- The **insert() method** in MongoDB inserts documents in the MongoDB collection. This method is also used to create a new **collection** by inserting documents.
- You can insert documents with or without the **_id field**. If you insert a document in the collection without the _id field, then MongoDB will automatically add an _id field and assign it with a unique ObjectId.

And if you insert a document with the _id field, then the value of the _id field must be unique to avoid the duplicate key error.

insertOne()

- **insertOne()** method inserts a document into the collection. This method inserts only one document at a time. This method can also be used inside multi-document transactions.

```
// Define the student data as a JSON document
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
};

// Insert the student document into the "students" collection
db.students.insertOne(studentData);
```

OUTPUT

```
db> 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
... };

db> db.students.insertOne(studentData);
{
  acknowledged: true,
  insertedId: ObjectId('665a0fffd09da8f9e346b799')
}
db>
```

UPDATE

- The **update()** method in MongoDB updates a document or multiple documents in the collection. When the document is updated the **_id field** remains unchanged.
- This method can be used for a single updating of documents as well as multiple documents. By default, the **db.collection.update() method** updates a single document.

UpdateOne()

- The **updateOne()** method in MongoDB updates the first matched document within the collection based on the given query.
- The value of the **_id field** remains unchanged after updating the value. This method updates one document at a time and can also add new fields to the given document.

```
// Find a student by name and update their GPA
db.students.updateOne({ name: "Alice Smith" }, { $set: { gpa: 3.8 } });
```

OUTPUT

```
db> db.students.updateOne({name:"Alice Smith"},{$set:{gpa:3.8}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
```

UpdateMany()

- The **updateMany()** method updates all the documents in MongoDB collections that match the given query. When you update your document, the value of the _id field remains unchanged.
- You can also use this method inside multi-document transactions.

```
// Update all students with a GPA less than 3.0 by increasing it by 0.5
db.students.updateMany({ gpa: { $lt: 3.0 } }, { $inc: { gpa: 0.5 } });
```

OUTPUT

```
db> db.students.updateMany({gpa:{$lt:3.0}},{$inc:{gpa:0.5}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 261,
  modifiedCount: 261,
  upsertedCount: 0
}
```

DeleteMany()

- The deleteMany() method is a part of MongoDB's CRUD (Create, Read, Update, and Delete) operations. As the name suggests, it is used to delete more than one document that satisfies the specified criteria.
- deleteMany() returns acknowledged as true if the function runs with the writeConcern parameter; otherwise, false.

```
// Delete all students who are not hotel residents
db.students.deleteMany({ is_hotel_resident: false });
```

OUTPUT

```
db> db.students.deleteMany({is_hotel_resident:false});
{ acknowledged: true, deletedCount: 255 }
```

PROJECTION

- MongoDB provides a special feature that is known as **Projection**. It allows you to select only the necessary data rather than selecting whole data from the document.

```
// Get only the name and gpa for all students
db.students.find({}, { name: 1, gpa: 1 });

// Exclude the "_id" field from all queries by default
db.students.find({}, { _id: 0 });
```

OUTPUT

db.students.find({},{name:1,gpa:1});

db.students.find({},{_id:0});

```
db> db.students.find({},{_id:0});
[
  {
    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: 2.77,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.82,
    blood_group: 'B+',
    is_hotel_resident: true
  },
{
  name: 'Student 346',
  age: 25,
  courses: "['Mathematics', 'History', 'English']",
  gpa: 3.31,
  home_city: 'City 8',
  blood_group: 'O-',
  is_hotel_resident: true
},
{
  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
},
{
  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
},
```

```
{
  name: 'Student 440',
  age: 21,
  courses: "['History', 'Physics', 'Computer Science']",
  gpa: 2.56,
  home_city: 'City 10',
  blood_group: 'O-',
  is_hotel_resident: true
},
{
  name: 'Student 256',
  age: 19,
  courses: "['Computer Science', 'Mathematics', 'History', 'English']",
  gpa: 3.44,
  home_city: 'City 1',
  blood_group: 'B+',
  is_hotel_resident: true
},
{
  name: 'Student 177',
  age: 23,
  courses: "['Mathematics', 'Computer Science', 'Physics']",
  gpa: 3.02,
  home_city: 'City 10',
  blood_group: 'A+',
  is_hotel_resident: true
},
{
  name: 'Student 487',
  age: 21,
  courses: "['History', 'Physics', 'Computer Science']",
  gpa: 2.6,
  home_city: 'City 3',
  blood_group: 'B-',
  is_hotel_resident: true
},
{
  name: 'Student 213',
  age: 18,
  courses: "['English', 'History']",
  gpa: 2.89,
  blood_group: 'B-',
  is_hotel_resident: true
},
{
  name: 'Student 690',
  age: 22,
  courses: "['History', 'Physics', 'Mathematics']",
  gpa: 2.75,
  home_city: 'City 7',
  blood_group: 'AB-',
  is_hotel_resident: true
},
```