**WORK WITH MONGODB**

We have to download MongoDB community server and shell, shell should be integrated with community to work with mongo server DB

After install both of this you have to add environment variable for mongo and mongo shell

After adding it, check if it is working by typing the command mongod and mongosh

**DIFFERENCE BETWEEN MONGOD AND MONGOSH**

 **mongod** → The MongoDB **server process** that runs in the background to manage databases.

 **mongosh** → The MongoDB **shell (CLI tool)** used to interact with the server, run queries, and manage data.

**USE OF MONGO DB**

* Helps to handle large amount of data.
* It stores in json format so retrieving data will be fast.
* It is schema less DB, so that we can add new columns without altering the table like mysql

**TYPES OF DATA**

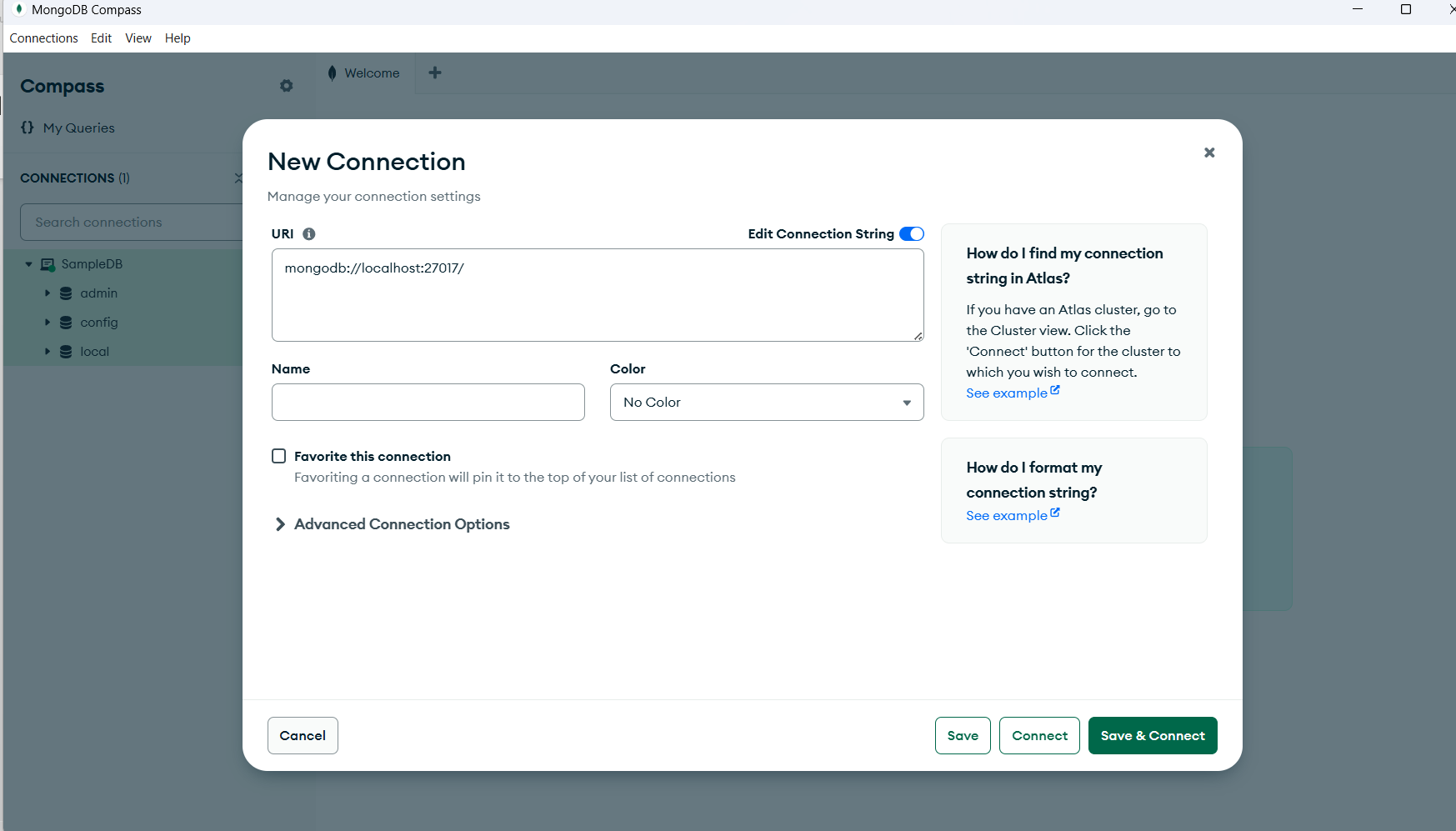
* Structure data – normal int, string data
* Ununstructure data – Video, Audio
* Semi structure – Can store structure data and unstructured data.

STRUCTURE OF MONGO

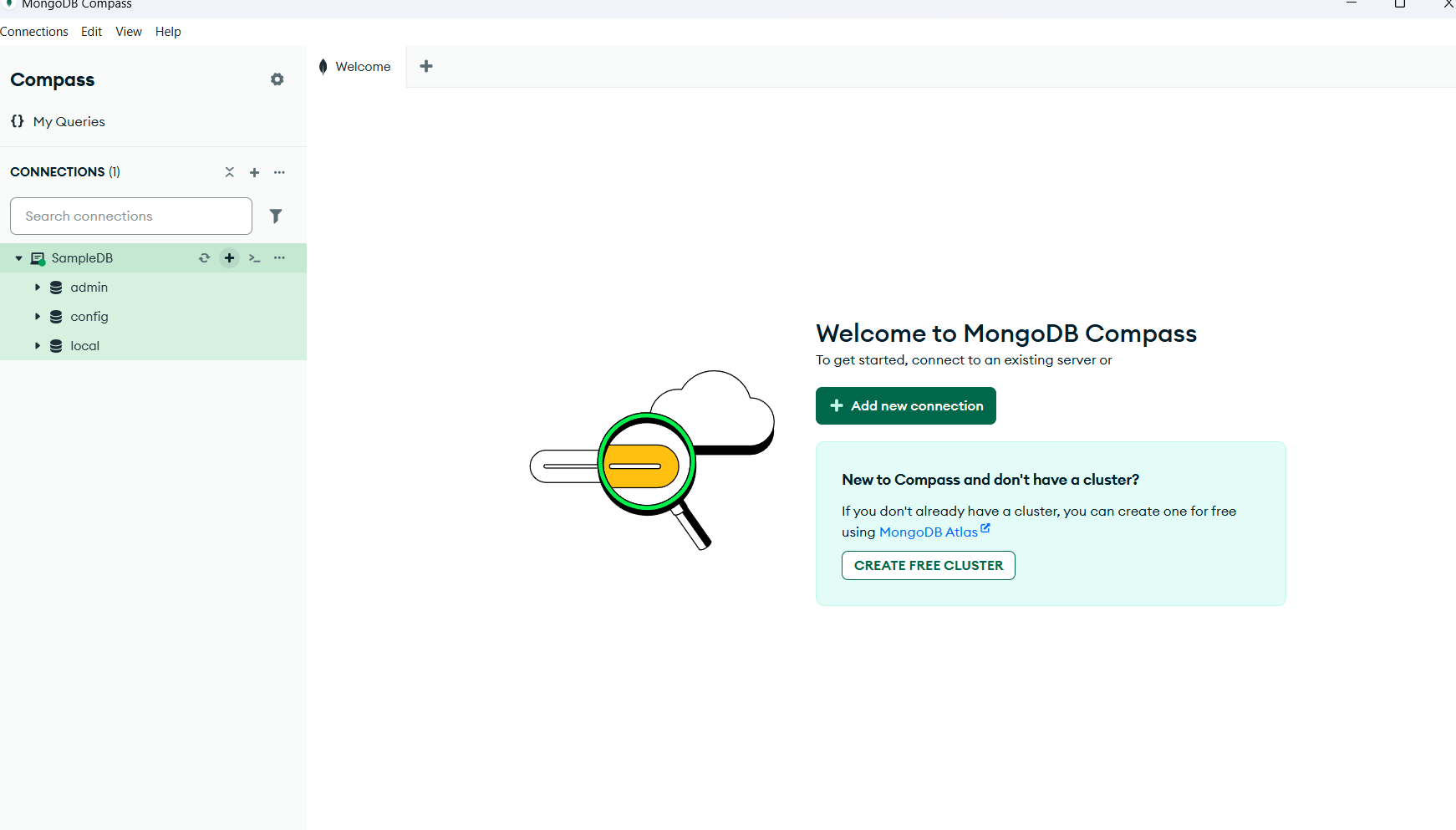
1. DB – Which is on server
2. Collection – Like tables
3. Document – Like rows in SQL EX: { column:value }
4. Fields – Like columns in SQL

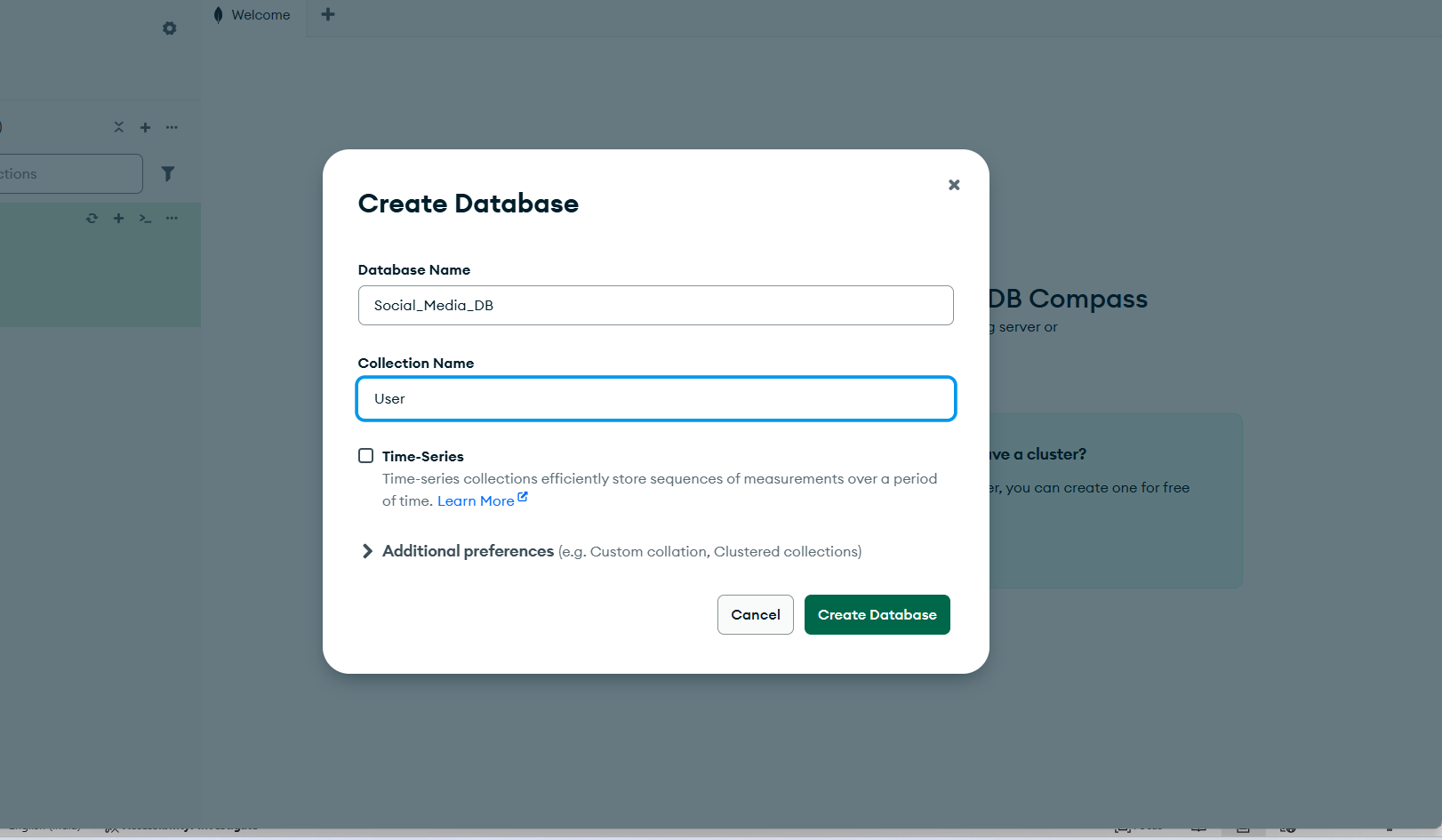
**WORKING WITH MONGODB**

1. **Create connection**



1. **Create DB with collection -** After creating connections click on the connection to make collection DB, admin, config, local are created automatically





**COMMANDS IN MONGO**

1. **show database –** shows the active database which has the collections and document, empty collection’s database will not be shown here.
2. **use db\_name**
3. **show collections**

**INSERT COMMAND**

1. **InsertOne –** inserts only one data at a time

**SYNTAX:**

**Shift + Enter for space**

Db.collectionname.insertOne({

Field: “value”

})

1. **InsertMany –** It returns acknowledgement with index of the inserted data

**SYNTAX:**

Db.collectionname.insertMany(

[

{ field: ‘value’ },

{ field: ‘value2’ },

]

)

**FIND COMMAND**

1. **Find() –** finds all data

SYNTAX: db.collectionname.find()

1. **FindOne({col:’value’}) –** Finds only one data.

**EXAMPLES:**

****

**UPDATE:**

1. **UpdateOne –** Updates first data which satisfies the condition first

**SYNTAX:**

**Db.collectioname.updateOne(**

**{**

**Field:’value’ #which reterives the data**

**},**

**{**

**$set:**

**{**

**Field:”updated value”,**

**Field2: “updated value2”**

**}**

**}**

**)**

1. **UpdateMany**

**REPLACE:**

Helps to replace the whole document

**SYNTAX:**

Db.collection.replaceOne(

{field:value }, # which statisfies the condition

{field1:value1, field2:value2}

)

**EX:**

****

After did the replace that particular document will be replaced with this document, so the filed isactive will not be there for the \_id 68c91a57..



**EXAMPLE FOR DELETE ONE**

****

**AGGREGATION –** Used to get combination version of data

1. **$match –** filters based on condition

**Syntax:** db.collectioname.aggregate([

$match:{field:value}

])

1. **$project –** reshapes each document in stream, it is like select in mysql, helps to select and show some particular columns in collection.

**Syntax:**

Db.collectionname.aggregate(

[

{

$project: {

Field : 1,

New\_field : { $multiply: [‘$field1’,’$field2’] },

\_id: 0

}

}

]

)

1. **$group –**

**Sytanx:**

db.collectionname.aggregate(

[

{

$group:

{

\_id:”$field\_value\_want\_to\_group”,

New\_field: {

$sum:

{

$multiple: [“$field1”,”$field2”]

}

},

New\_field: {

$sum : “$fieldname”

}

}

}

]

)

1. **$sort**

**Syntax:**

Db.collectioname.aggregate(

[

$sort : { field\_name: -1 or 1 }

]

)

1. **$limit –** Helps to take some values

**Syntax:**

db.collectioname.aggregate(

[

{ $sort : { field\_name: -1 or 1 } },

{ $limit: 2 }

]

)

1. **$skip**
2. **$unwind**
3. **$lookup –** Helps to perform join multiple collections to get combined data.

In below we are going to join two collections with common values in each field

EX: field from 1st collection will have same values of field form second collection.

**Syntax:**

Db.collectionname.aggregate(

[

{

$lookup:{

From:”first\_collection\_name”,

localField:”field\_name\_of\_2nd\_collection”,

foreignField:”field\_nameof\_1st\_collection”,

as: “newFieldName”

}

}

]

)

**USE OF ATLAS**

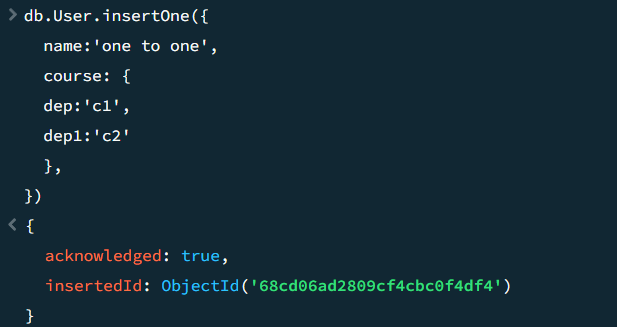
It is a could based server so that we can save our data to the cloud MongoDB

**DATA MODLING**

Defines how the data in MongoDB should stores in that DB. Has three types

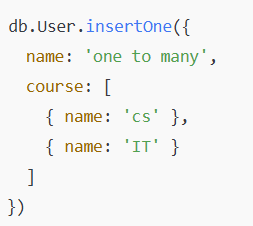
1. Entities – Document or collections, we can represent a data like in this entity format
2. Attribute – it is Field, we can represent the data with field values like grouping
3. Relationship – Like group by, it has 3 types
4. One – One: A collection has only one collection

**EX:**



1. One – Many – We can give multiple collection inside single collection

**EX:**

****

1. Many – Many – A field value in one collection will related to more than one values in another collection

**OPERATORS IN MONGODB**

Helps to perform comparison with condition

**SYNTAX:**

1. $eq
2. $ne
3. $in
4. $nin – not in
5. $lt,gt, lte, gte

Expect in, nin all are will compare only int == int comparision and str == str comparision

**Example:**

****

****

**LOGICAL OPERATOR**

1. $and
2. $or
3. $not
4. $nor – retrieves data that doesn’t satisfies any one of the condition in collection of condition

**SYNTAX:**

Db.collection.find(

Field: {

$operator: [ {field:value},{, field2:value } ]

}

)

**EXAMPLE:**

****

From the below data it takes above data with condition

