PAGE NO. :
DATE : 1 May 21

MONGO DB WORKSHOP DAY-1

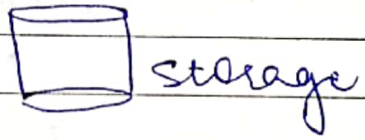
- Flexible document oriented db

#mongodb.com

- document based database
- noSQL

Data

- file
  - folder



- read-write operation == Input-Output operation (I/O)

  I/O ∝ performance

- changing the organization of data in ~~c~~tain way or format can change the whole i/o ~~performance~~ operation.

- mongodb download
  - (Community version)
    - version: 4.4.5 (current)
    - pkg: msi

PAGE NO :
DATE : / /

MongoDB

landline | fax | city2

schema → field

eg. table

Sno	name	phone	city	→ record
1	vimal	1111	jaipur	
2	krish	2222	mum	
3	jack	3333	delhi	
4	tom		mum	7777
5	yash		delhi	
6	harry	4444	mum	6666

- Flexibility in schema while inserting / saving of data.
- schema is normally ~~or~~ fixed ~~How we have structured data~~ Here, it is (schema less).
- SQL kind : structured data & schema (we have fixed plan).

7 User2 7787 delhi mum

- Or we can create a list array in city or we can create another column (city2).

- In mongodb/no sql db we create structure for every field.
  It is like a sparsed matrix

- In mongodb records are (Documents) independent
- The way we model or organise our data is known as Data Model.

PAGE NO :
DATE : / /

Server ← Client
↳ mongod ↳ mongo

win → #: mongo
> show dbs
↳ lists all the db.
> use tw
↳ If not created, it first creates & switches to tw db.

> show collections

> db
↳ tw

> db.createCollection('testcc')
> show collections
↳ testcc

- we directly created a table without defining a schema...
we need to define it with records.

> db.testcc.insert({"name": "vimal", "phone": 1111
})
> db.testcc.find()

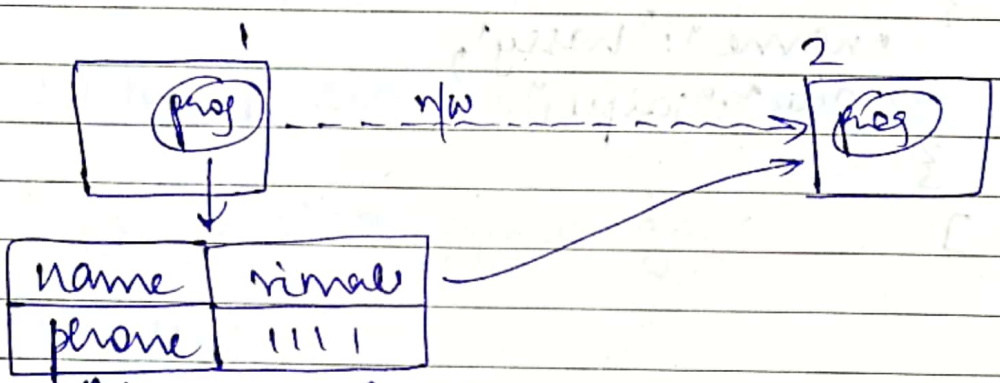
> insert({'name": "jack", "landline": 1111,
"city": "jaipur"})

PAGE NO. :
DATE : / /

> db.testcc.find({"name": "nimal"})
↳ shows the nimal rec.

> db.testcc.drop()
↳ deletes the collection

> db.dropDatabase("cw")
↳ deletes cw db.



{key {value
∴ key-value pair

- Anybody want to transfer data over network they will use standard format, json (in key-value pair)

{
"name": "nimal",
"phone": 1111
}

document

json → javascript Object notation

PAGE NO.:
DATE: / /

```
[
  {
    "name": "vinod",
    "phone": [1111, 2222]
  },
  {
    "name": "jack",
    "landline": 2222
  },
  {
    "name": "harry",
    "city": "jaipur"
  }
]
```

nested/embedded in mongodb

↳ {"primary": 1111, "secondary": 2222}

> db.customers
↳ creates collection customers

> db.customers.find().pretty()

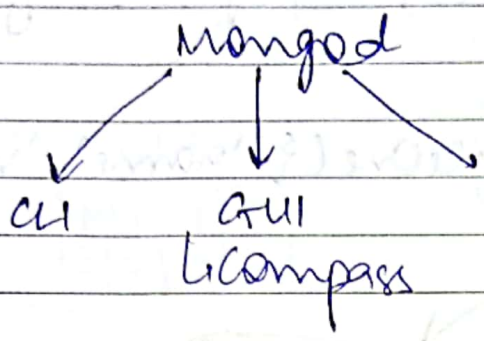
> db.c__.find({"phone": {$gt: 2222}})
↳ pre created key

$gt → greater than
$st → smaller than

> db.find({"name": "jack"}, {"landline": 1})

PAGE NO. :
DATE : / /

- _id is the unique key in mongodb.



- CURD operations are main in db.

PP Integration with mongodb

pip install pymongo

python

```
from pymongo import MongoClient
client = MongoClient("use")
```

↓

mongo://127.0.0.1:27017

```
output = Client['w3']['customers'].find({'name': 'vinod'})
```

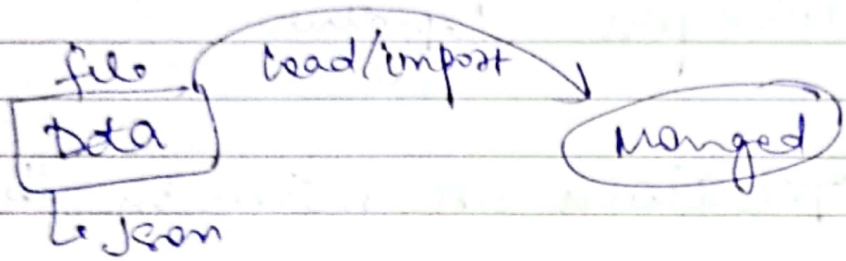
↳ gives the cursor (no output)

```
for i in output:
    print(i)
```

PAGE NO :
DATE : / /

> db.customers.update ({'name': 'jack'}
{$set : {'city': 'jaipur', 'age': 23}})

> db.customers.deleteOne ({'name': 'vimal'})



• Download mongoimport tool & set env var

mongoimport persons.json -d demo
-c contacts --jsonArray

> use mperson
> show collections
↳ contacts

> db.contacts.find()
↳ by default prints 20 records

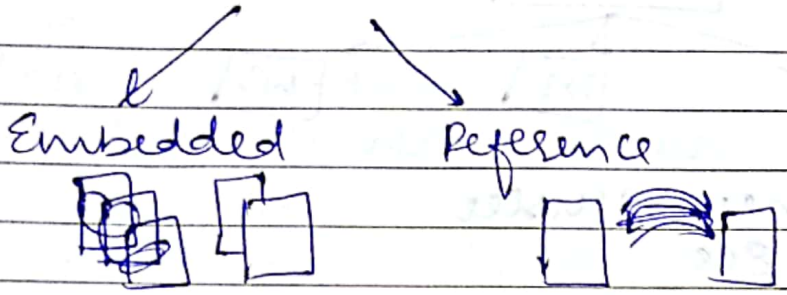
> db.find().count()
↳ 5000

PAGE NO. :
DATE : / /

Documented

↓

Data model



~~>db.contacts.find({"dob":{"age":25}})~~

>db.contacts.find({"dob.age":52})

↳ find in dob

↳ age

>db.cont__.find({"dob.age":{$gt:60}})

PAGE NO.:
DATE: 7 May 21

MONGO DB WORKSHOP DAY 2

API
- Python
- Node js
- Scala

- We can install drivers and use MongoDB these.

WebUI
- Mongo Compass

net start mongod
#mongo - If the srv is not started
> show dbs

- Rows == Documents
- Table == Collections

> mongoimport persons.json -d contactData (database)
-c ~~dett~~ contacts --jsonArray
(collection) (format(file))

> use contactData
> show collections
> db.contacts.find()

> db.contacts.findOne()
- prints one random document
- automatically prints in pretty format

> db.contacts.find({"dob.age": 29})

> _____ _____.count()

PAGE NO. :
DATE : / /

> db.contacts.find({"dob.age" : {$gt:60}}).count()

> db.contacts.explain().find(....)
↳ gives info abt the fn, here, find()
↳ Query Planner.

> db.contacts.explain("executionStats").find()
↳ executionTimeMillis : 6,
execution Time

> db.contacts.getIndexes()

> db.contacts.createIndex({"db.age": 1})

: 1 ascending order
: -1 descending order

> . . getIndexes()

> - . — . explain("executionStats").find(
{"dob.age": {$gt : 70}})

> db.contacts.dropIndexes({"dob.age": 1})
↳ deletes the index.

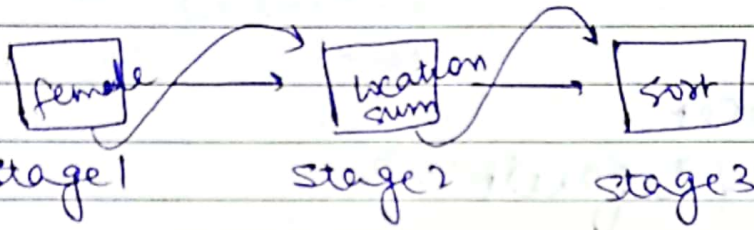
- Indexes are not always helpful.
- We can also create indexes on the basis of 2 keys.

PAGE NO. :
DATE : / /

```
>db.contacts.createIndex({"dob.age":1,
                          "gender":1})
```

↳ Compound Index

- Aggregation Framework
  - ↳ Pipeline



- 5000 doc
  - ↳ Gender: Female ⟶ $match
  - ↳ location.state : xxx <sum> of female>
  - ↳ sort ⟶ $sort

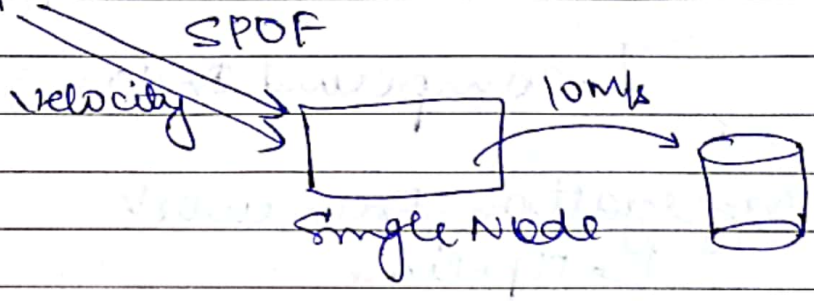
```
> db.contacts.find({gender:"female"}).count()

> db.contacts.aggregate([
... {$match:{gender:"female"}}
.... ])
```

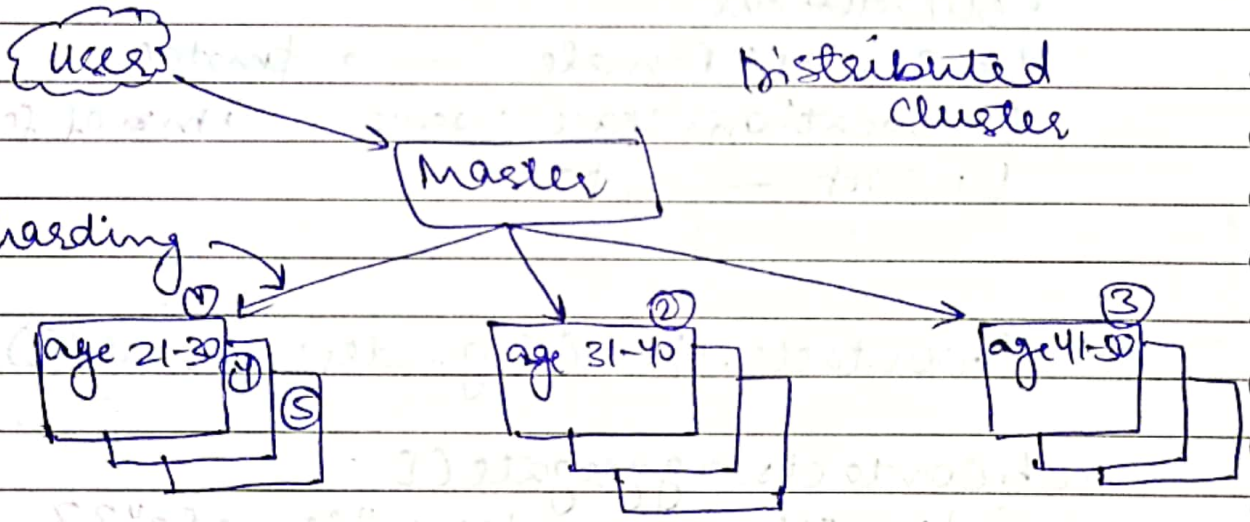
```
> db.contacts.aggregate([
    {$match:{gender:"female"}},
    {$group:{_id:"$location.state",
             y:{$sum:1}}},
    {$sort:{y:-1}}
])
```

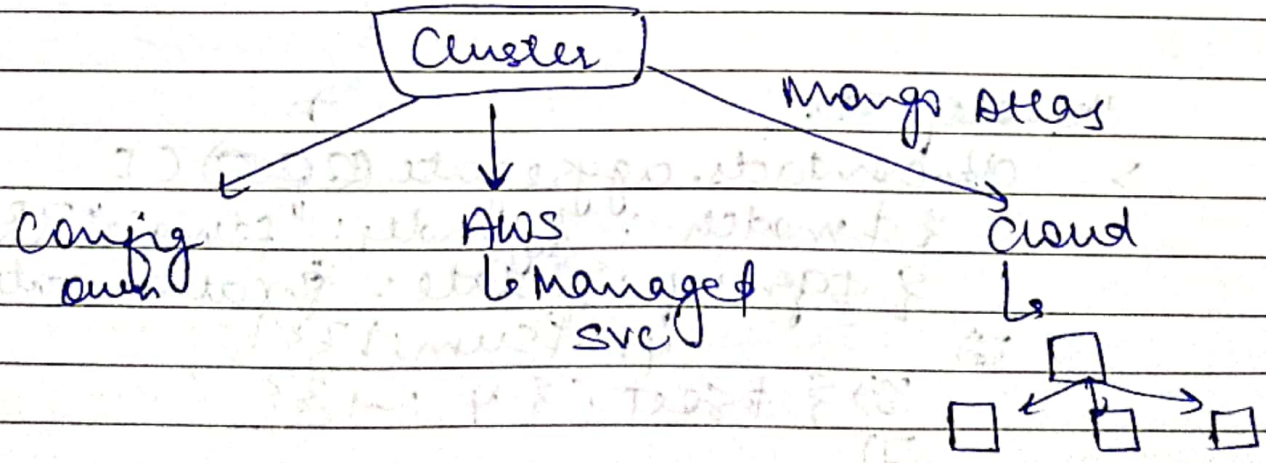
PAGE NO. :
DATE : / /

- User / App



- ReplicaSet
  - By default : 3



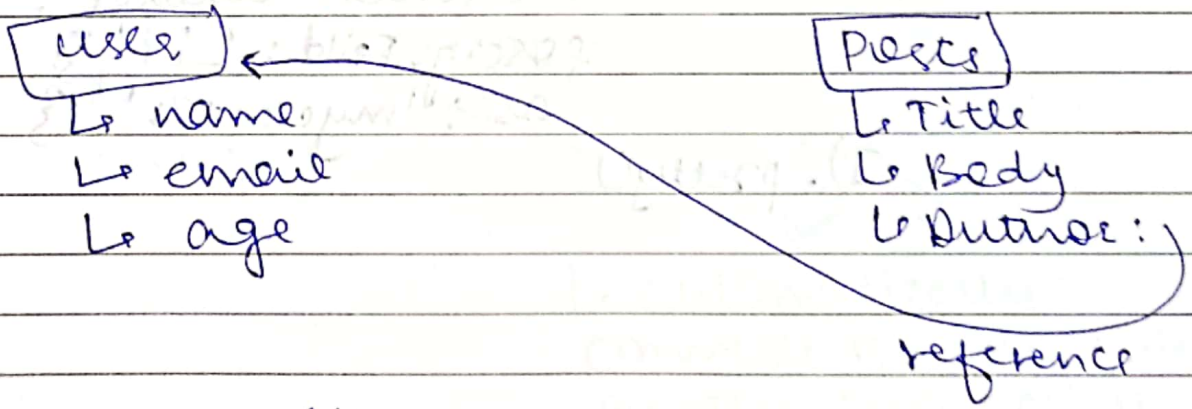


PAGE NO :
DATE : / /

AWS
- DocumentDB (AWS MongoDB SVC)
  - Launch
    - name
    - version
    - instance class
    - no of instances (slaves)
    - Auth.
      - User
      - passwd.
        - > Create Cluster

mongo
> use blog
> ~~show collection~~
> ~~db.users.insert()~~

Referencing Model



collection

```
> db.users.insert({'name':'vimal',
    age:21, email:'v@lw.com'})
```

PAGE NO.:
DATE: / /

> db.posts.insert({ title: 'my first post', body: 'body', author: ObjectId("xox") })

> db.posts.find().pretty()
Ⓛ will give the ref in author but we want to print the details of the author

~~> db.posts.~~

> db.posts.aggregate([{$lookup: {from: 'users', localField: 'author'}}])

> db.posts.aggregate([
{ $lookup: { from: 'users', localField: 'author', foreignFeild: '_id', as: 'mycreator' } }
]).pretty()