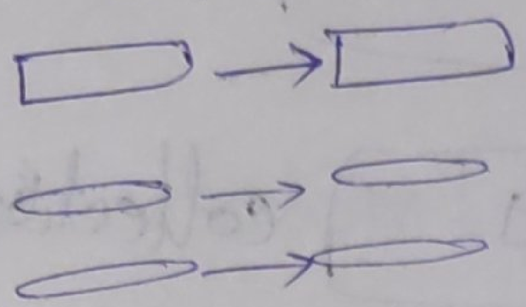
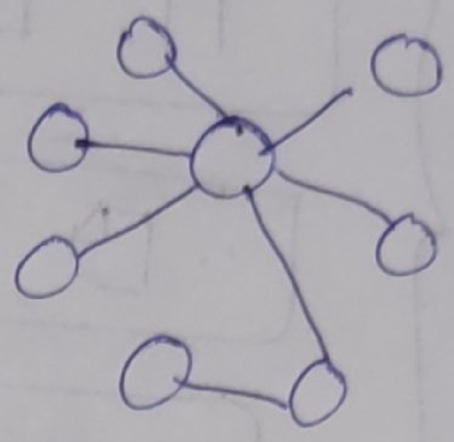
mongo DB

*) It is a no SQL database management System that can manage humongous amount of data.

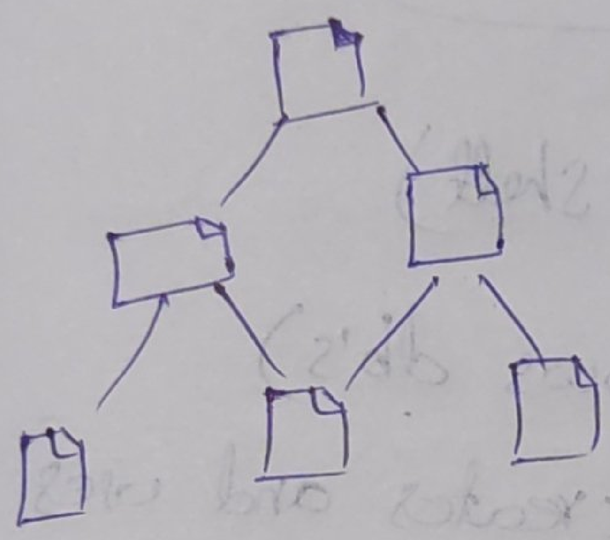
NoSQL : key-value



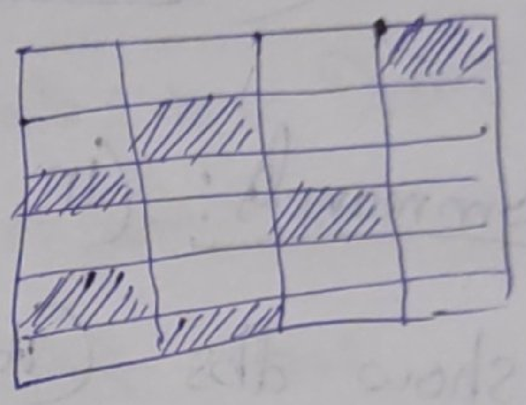
Graph



Document

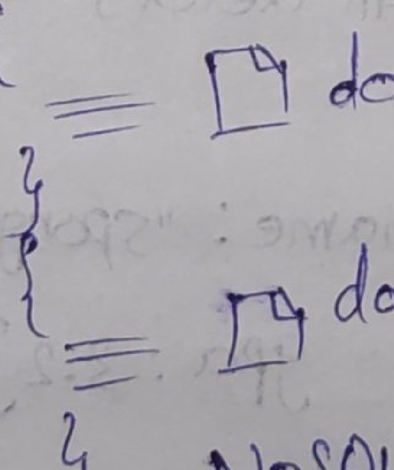


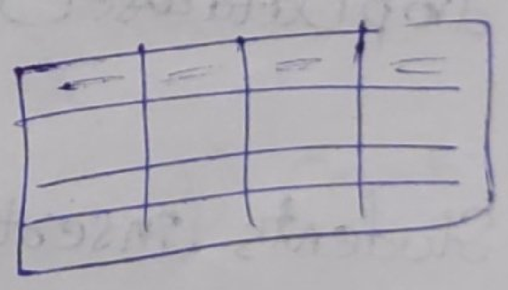
Wide column



NoSQL : Not Only Structural Query language

*) In SQL the data will store in table structure
In NoSQL the data will store in documents structure.





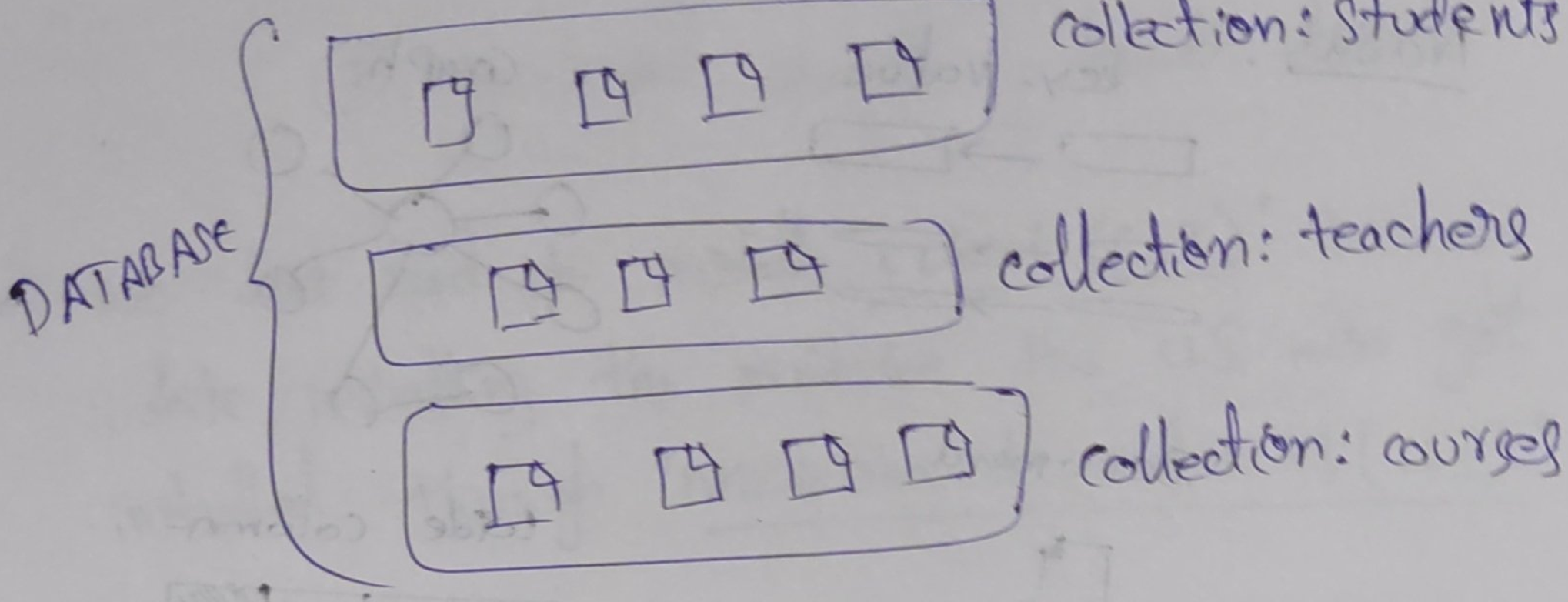
SQL

*) Data in each document is stored as field value pairs similar to a Json format (technically ~~bson~~ -binary JS object)

*) A "document" is a group of "field value pairs" to represent an object.

*) A 'collection' is a group of one or more documents.

*) A "database" is a group of one or more collections.



*) Commands: (in mongodb shell)

① show dbs (it will shows db's)

② use school (it will creates and uses a new db at a time).

③ db.createCollection("students") (it will creates a new collection)

④ db.dropDatabase() (it will deletes current db).

⑤ db.students.insertOne({name: "spongebob", age: 30, gpa: 3.2})

⇒ this will create a one document in the collection (students), if students collection is not available, it will create when you run this command.

⑥ db.students.find()

⇒ it will give all the documents data present in students collection.

② db.students.insertMany([{name:"Patrick", age:38, gpa:1.5},
{name:"sandy", age:27, gpa:4.0},
{name:"Gary", age:18, gpa:2.5}])

⇒ This will create many documents at a time in a students collection.

⑧ db.students.insertOne({name:"Larry",
age: 32,
gpa: 2.8,
fullTime: false, #(or) true
registerDate: new Date(),
#(or) new Date("2025-01-02T00:00:00")
graduationDate: null,
courses: ["biology","chem","calcul"],
address: {street:"123 fake st.",
city:"Bikini Bottom",
zip: 12345}})

⇒ This contains max all the datatypes.

Sorting:

⑨ db.students.find().sort({name:1}) (prints in normal order)

⑩ db.students.find().sort({name:-1}) (print in reverse order)

⑪ db.students.find().sort({gpa:1})

⇒ It will prints the data in ascending order (gpa).

⑫ db.students.find().sort({gpa:-1})

⇒ It will prints the data in desending order (gpa).

limiting:

(13) db.students.find().limit(2)

⇒ It will gives the first 2 created 2 documents.

(14) db.students.find().sort({gpa:-1}).limit(1)

⇒ It will gives the one document who has the highest GPA.

(15) db.students.find().sort({gpa:1}).limit(1)

⇒ It will gives the 1 doc. who has lowest GPA.

---

(16) db.students.find({name:"spongebob"})

⇒ It will gives the all documents those have name is "spongebob".

(17) db.students.find({name:"spongebob"}, age:30})

⇒ It will gives all the documents those are having the name is "spongebob" and age is "30".

It will acts like "and" function not "or" function
✓ ×

(18) db.students.find({}, {name:true})

⇒ It will gives every document but only gives their name.

(19) db.students.find({}, {gpa:true})

.find({query}, {projection})

⇒ It'll gives all documents, but only gives their gpa's.

The above (18) and (19) commands gives the o/p like

(18)
```
[ { _id : ObjectId("689cda2...."), name : 'spongebob' },
]
```

(19)
```
[ { _id : ObjectId("9ac23...."), gpa : 3.2 },
]
```

Now, i dont want (_id : ObjectId), then for this:

(20) db.students.find({}, {_id : false, name : true})

```
[ { name : 'spongebob' },
]
```

(21) db.students.find({}, {_id:false, name:true, gpa:true})

⇒ It will gives like:

```
[ { name : 'spongebob', gpa : 3.2 },
]
```

Update: (.updateOne(filter, update))

(22) db.students.updateOne({name : "spongebob"},
{$set : {fullTime : true}})

⇒ It will add another field value pair (fullTime), if that field not present, otherwise it will modify the existed one.

(23) db.students.updateOne({_id: ObjectId("68a ....")},
{$set: {name: "SSS", age: 90}})

=> It will update multiple "field value pairs" for one document by "ObjectId".

(24) db.students.updateOne({name: "Spongebob",
{$unset: {fullTime: ""}})

=> It will remove the "fullTime" field from the document named "Spongebob". (removing field value pairs).

(25) db.students.updateOne({name: "Spongebob",
{$unset: {age: "", gpa: ""}})

=> It will removes "multiple" field value pairs.

(26) db.students.updateMany({}, {$set: {fullTime: false}})

=> It will add (or) modify fullTime field value pair for every document present in students collection.

(27) db.students.updateMany({fullTime: {$exists: false}},
{$set: {fullTime: true}})

=> This will create fulltime fields for the documents those dont have that field (fullTime: true)

Delete:

(28) db.students.deleteOne({name:"Larry"})

⇒ It will delete only one document named with (name: Larry).

(29) db.students.deleteMany({fullTime:false})

⇒ It will delete all the documents those have fields (fullTime: false).

(30) db.students.deleteMany({registerDate:{$exists:false}})

⇒ It will delete all the documents those dont have the "registerDate field".

Comparison operators:

*) Operators are denotted with dollor sign "($)".

*) Comparision operators return data based on "value comparisons".

(31) db.students.find({name:{$ne:"spongebob"}})

⇒ this will returns all the documents except the document named with (name: "spongebob").

(32) db.students.find({age:{$lt:20}})

⇒ this will gives all the documents those have the age less than 20 (lt = less than).

(33) db.students.find({age:{$lte:20}}) (less than equal to)

(34) db.students.find({age:{$gt:20}}) (greater than)

(35) db.students.find({age:{$gte:20}}) (gt equal to)

(36) db.students.find({gpa:{$gte:3, $lte:4}})

=> this will return all the documents that have gpa between (3 to 4).

(37) db.students.find({name:{$in:["ram","isn","tns"]}})

=> this will returns the documents that have the name present in that list (present in command).

(38) db.students.find({name:{$nin:["ram","isn"]}})

=> this will returns the documents that dont have the name present in that list (not in(nin)).

Logical Operators:

=> logical operators return data based on expression that evaluate to true or false.

(1) and (2) or (3) not (4) nor

(39) db.students.find({$and:[{fullTime:true},
{age:{$lte:22}}]})

=> this will gives the documents those have fullTime field is True and age is less than or equal to 22.

(40) db.students.find({$or:[{fullTime:true},
{age:{$lte:22}}]})

=> this will gives the documents those have fullTime field is True or age is less than or equal to 22.

(41) db.students.find({$nor:[{fullTime:true},
{age:{$lte:22}}]})

-> this will gives the documents those do not have fullTime field is true and age is do nd have age is less than or equal to 22.

(42) db.students.find({age:{$not:{$gte:30}}})

=> this will gives the documents those ages are not greater than equal to 30. (If age is "null". in this case it will return that null aged document)

indexes:

Index allows for the quick lookup of a field however it takes up more memory and slows insert, update & remove operations.

=> If i run "db.students.find({name:"Larry"})" it will give o/p, but it will checks the documents line by line (like linear search). It makes more time instead this we use "index".

=> To apply an index to a field:

(43) db.students.createIndex({name:1}) (1 means axen. order)

=> (44) db.students.getIndexes()

```
[ {v:2, key:{_id:1}, name:'_id_'},
  {v:2, key:{name:1}, name:'name_1'}
]
```

This means, the 1st line indicates the index values for ObjectId's (autogenerated by mongodb), 2nd line indicates the index values for all name fields in collection (user created).

(45) db.students.dropIndex("name_1")

⇒ It will deletes the index values for the name fields in the collection.

for all documents.

Collections:

*) Collection is a group of "documents."

(46) show collections (It will print all collections present in that db)

(47) db.createCollection("teachers", {capped:true,
size:10000000,
max:100})

⇒ this will create a collection ~~but~~ with the maximum size of 10 MB and maximum document size of 100. (this means in this collection we should create upto only 100 documents, with the MB should below 10.

⇒ the "capped" is either "true" & "false"