MongoDB Course Notes

mongoDB console
mongoDB atlas(gui for mongo)

mongoDB is a NoSql(Not Only Structured Query Language)

mongoDb word come from the word ***Humongous*** (which means large amount of data)

mongoDB is all about storing data helps to manage large amount of data
It follows noSQL methodology

In mongoDB a database contain Collection which contains documents which are Bson Object in the form of {key:value} pair

Sql(structure query language) uses relational database store data in the form of tables having fixed schema.
The columns are fixed.
The data in the table are in the form of rows called as records.

NoSql data is stored in collections which are tables in relational sql.
Collections contains documents which are loosely type schema less BSON(Binary JSON) objects in the form of {key:value} pair similar to JSON.

For example

```
{
 name:"udit",
 age:27,
 gender:"male"
}
```

Its a record in a document

MongoDB is scalable as compared to mysql

In Mysql the the tables columns(with data-types) are predefined schema

In mongoDB the documents are loosely type they are flexible Not strict schema and can be expanded easily.

Commands and there output

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Show all the dbs in local server

```
> use test-db
switched to db test-db
```

Created the db with name <test-db> and switched to it

```
> db
test-db
```

Shows the current db you are working on

```
> db.createCollection("posts")
{ "ok" : 1 }
```

Created the collection in the test-db with name posts

```
> show dbs
```

```
admin    0.000GB
config   0.000GB
local    0.000GB
test-db  0.000GB
```

The > show dbs will show your database only if it has one or more collections in it.

```
> show collections
posts
```

Show the collections in current db.

```
> db.posts.drop()
true
```

Use to delete a collection in mongoDB
db.<collection-name>.drop()

```
> db.dropDatabase()
{ "dropped" : "test-db", "ok" : 1 }
```

Drop the database you are currently using

```
> cls
```
To clear the console

```
> use cheat-sheet-db
switched to db cheat-sheet-db
```

created the db with name cheat-sheet-db

```
> db
```

Cheat-sheet-db

Show the current db you are working on

```
> db.createCollection('posts')
{ "ok" : 1 }
```

created the collection in db with name posts

```
> show dbs
admin           0.000GB
cheat-sheet-db  0.000GB
config          0.000GB
local           0.000GB
```

Show all the db present in mongo

```
> show collections
posts
```

Show the collections present in db

```
> db.posts.insert({
...   "title": "postOne",
...   "body": "postOneBody",
...   "category": "news",
...   "likes": 4,
...   "tags": ["news", "events"],
...   "moderator": {
...     "name": "jon doe",
...     "role": "author"
...   },
...   "date": Date()
... })
WriteResult({ "nInserted" : 1 })
```

Insert a record into the posts collection

```
> db.posts.find().pretty()
{
	"_id" : ObjectId("5f2934ca809db97f898d475d"),
	"title" : "postOne",
	"body" : "postOneBody",
	"category" : "news",
	"likes" : 4,
	"tags" : [
		"news",
		"events"
	],
	"moderator" : {
		"name" : "jon doe",
		"role" : "author"
	},
	"date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)"
}
```

Show all the documents in current collection i.e posts

```
> db.posts.insertMany([{
...     "title": "postTwo",
...     "body": "postTwoBody",
...     "category": "sports",
...     "likes": 9,
...     "tags": ["sports", "news"],
...     "moderator": {
...       "name": "gilly gibson",
...       "role": "writer"
...     },
...     "date": Date()
... },
```

```
... {
...     "title": "postThree",
...     "body": "postThreeBody",
...     "category": "politics",
...     "likes": 2,
...     "tags": ["politics", "news"],
...     "moderator": {
...       "name": "billy jean",
...       "role": "author"
...     },
...     "date": Date()
... },

... {
...     "title": "postFour",
...     "body": "postFourBody",
...     "category": "entertainment",
...     "likes": 17,
...     "tags": ["entertainment", "events"],
...     "moderator": {
...       "name": "will buyers",
...       "role": "publisher"
...     },
...     "date": Date()
... }])
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5f29384d809db97f898d475e"),
        ObjectId("5f29384d809db97f898d475f"),
        ObjectId("5f29384d809db97f898d4760")
    ]
}
```

Use to insert multiple documents with {key:value} pair inside the collection(posts)

```
> db.posts.find({tags:'news'}).pretty()
{
	"_id" : ObjectId("5f2934ca809db97f898d475d"),
	"title" : "postOne",
	"body" : "postOneBody",
	"category" : "news",
	"likes" : 4,
	"tags" : [
		"news",
		"events"
	],
	"moderator" : {
		"name" : "jon doe",
		"role" : "author"
	},
	"date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)"
}
{
	"_id" : ObjectId("5f29384d809db97f898d475e"),
	"title" : "postTwo",
	"body" : "postTwoBody",
	"category" : "sports",
	"likes" : 9,
	"tags" : [
		"sports",
		"news"
	],
	"moderator" : {
		"name" : "gilly gibson",
		"role" : "writer"
	},
	"date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
{
	"_id" : ObjectId("5f29384d809db97f898d475f"),
```

```
        "title" : "postThree",
        "body" : "postThreeBody",
        "category" : "politics",
        "likes" : 2,
        "tags" : [
                "politics",
                "news"
        ],
        "moderator" : {
                "name" : "billy jean",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}

> db.posts.find({tags:'news'}).pretty()
```

The above query will give all the documents from the collection <posts> where <tags> array contains <news> as an element.

It is same as
Select * from posts where tags=”news”;

```
SORT IN mongoDB
> db.posts.find().sort({likes:1}).pretty()
{
        "_id" : ObjectId("5f29384d809db97f898d475f"),
        "title" : "postThree",
        "body" : "postThreeBody",
        "category" : "politics",
        "likes" : 2,
        "tags" : [
                "politics",
                "news"
        ],
```

```
        "moderator" : {
                "name" : "billy jean",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOneBody",
        "category" : "news",
        "likes" : 4,
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)"
}
{
        "_id" : ObjectId("5f29384d809db97f898d475e"),
        "title" : "postTwo",
        "body" : "postTwoBody",
        "category" : "sports",
        "likes" : 9,
        "tags" : [
                "sports",
                "news"
        ],
        "moderator" : {
                "name" : "gilly gibson",
                "role" : "writer"
        },
```

```
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
{
        "_id" : ObjectId("5f29384d809db97f898d4760"),
        "title" : "postFour",
        "body" : "postFourBody",
        "category" : "entertainment",
        "likes" : 17,
        "tags" : [
                "entertainment",
                "events"
        ],
        "moderator" : {
                "name" : "will buyers",
                "role" : "publisher"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
```

```
> db.posts.find().sort({likes:1}).pretty()
```

```
> db.posts.find().sort({likes:-1}).pretty()
```

This will sort all the posts by the number of the likes in the document
1: depicts ascending order
-1: depicts descending order

```
#count the no. of document in the collection in mongoDb that matches the condition
```

```
> db.posts.find().count()
4
```

The above query gives the total number of document in posts collection which is 4

```
> db.posts.find({category:'news'}).count()
1
```

The above query gives the total number of document in posts collection whose category is <news> which is 1

```
# limiting the output of the query in mongoDb
With limit()
> db.posts.find().limit(2).pretty()
{
	"_id" : ObjectId("5f2934ca809db97f898d475d"),
	"title" : "postOne",
	"body" : "postOneBody",
	"category" : "news",
	"likes" : 4,
	"tags" : [
		"news",
		"events"
	],
	"moderator" : {
		"name" : "jon doe",
		"role" : "author"
	},
	"date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)"
}
{
	"_id" : ObjectId("5f29384d809db97f898d475e"),
	"title" : "postTwo",
	"body" : "postTwoBody",
	"category" : "sports",
	"likes" : 9,
	"tags" : [
		"sports",
```

```
            "news"
        ],
        "moderator" : {
            "name" : "gilly gibson",
            "role" : "writer"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
```

```
> db.posts.find().limit(2).pretty()
```

Gives us the only 2 documents from the whole collection

```
# you can combine the limit with sort
> db.posts.find().sort({likes:1}).limit(1).pretty()
{
        "_id" : ObjectId("5f29384d809db97f898d475f"),
        "title" : "postThree",
        "body" : "postThreeBody",
        "category" : "politics",
        "likes" : 2,
        "tags" : [
            "politics",
            "news"
        ],
        "moderator" : {
            "name" : "billy jean",
            "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
```

```
> db.posts.find().sort({likes:1}).limit(1).pretty()
```

The above query gives the document with the least likes cause sorted on the basis of likes:1

```
> db.posts.find().sort({likes:-1}).limit(1).pretty()
{
	"_id" : ObjectId("5f29384d809db97f898d4760"),
	"title" : "postFour",
	"body" : "postFourBody",
	"category" : "entertainment",
	"likes" : 17,
	"tags" : [
		"entertainment",
		"events"
	],
	"moderator" : {
		"name" : "will buyers",
		"role" : "publisher"
	},
	"date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
```

```
> db.posts.find().sort({likes:-1}).limit(1).pretty()
```

The above query gives the document with the most likes cause sorted on the basis of likes:-1

```
> db.posts.findOne({tags:'events'})
{
	"_id" : ObjectId("5f2934ca809db97f898d475d"),
	"title" : "postOne",
	"body" : "postOneBody",
	"category" : "news",
	"likes" : 4,
	"tags" : [
		"news",
		"events"
	],
	"moderator" : {
		"name" : "jon doe",
		"role" : "author"
	},
	"date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)"
}
```

```
> db.posts.findOne({tags:'events'})
```

The above query gives the first document that matches the criteria.

forEach() in mongoDB

```
> db.posts.find().forEach(function(post){print('post Title is : '+post.title)})
post Title is : postOne
post Title is : postTwo
post Title is : postThree
post Title is : postFour
```

The above query will give all the documents in posts and then posts will go through the forEach() loop where the loop will print the title of each post.

forEach() loop in mongoDb works similar to map and anonymous functions in java.

 # update in mongoDB works in two ways

Ist way ------>one way is when the document is completely overridden by the object passed as argument.

Overriding the postTwo post with new document pass as the argument
Earlier the postTwo looks like this

```
> db.posts.find({title:'postTwo'}).pretty()
{
	"_id" : ObjectId("5f29384d809db97f898d475e"),
	"title" : "postTwo",
	"body" : "postTwoBody",
	"category" : "sports",
	"likes" : 9,
	"tags" : [
		"sports",
		"news"
	],
	"moderator" : {
		"name" : "gilly gibson",
		"role" : "writer"
	},
	"date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)"
}
```

```
> db.posts.update({title:'postTwo'},{title:'updated post two',body:'updated post two body',author:'post two author'},{upsert:true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

The above query will update the document in <posts> collection with title <postTwo> and override with the Document passed as second argument.
The upsert:true tells the query to add a new key:value passed in the second argument to be updated in the document.

The result of the update will be

```
{
	"_id" : ObjectId("5f29384d809db97f898d475e"),
	"title" : "updated post two",
	"body" : "updated post two body",
	"author" : "post two author"
}
```

#2-------> The second way to update the in mongoDB is to use the $set operator which only overrides only the key:value passed as the argument, not the whole document.

```
b.posts.update({title:'postOne'},{$set:{body:'postOne body is updated'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

The above query uses the $set operator to update only the given key value pair passed in the second argument instead of overriding the whole document

We mostly use this approach.

```
> db.posts.find({title:'postOne'}).pretty()
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
        "category" : "news",
        "likes" : 4,
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)"
}
```

 # the increment operator in mongoDB

Increment operator can be used to increment a number in the document.

```
> db.posts.update({title:'postOne'},{$inc:{likes:2}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

The above query will increment the likes key: value by 2 in the document with posts title:’postOne’

So the output will be--->

```
> db.posts.find({title:'postOne'}).pretty()
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
        "category" : "news",
        "likes" : 6,
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)"
}
```

#update-all documents of the collection with

```
db.collection.updateMany({},{})
```

#rename operator in mongoDB

Rename of the key in mongoDB...

```
> db.posts.updateMany({},{$rename:{'likes':'views'}})
{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 3 }
```

The above query will update all the documents in the collection<posts>
And rename all the fields in documents of likes with the views

The output after rename will be

```
> db.posts.find().pretty()
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
        "category" : "news",
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)",
        "views" : 6
}
```

In mongoDB there is no relation between the documents but we have sub Documents i.e a document inside another document.

So all the info we need is present inside the document

For example in relational database we have ***comment table*** and a ***post table***

And there would be a relation of one to many between a post and comment table
As a post will have multiple comments

But in mongoDb no sql world we have subDocuments ie all the info is present in the document

I.e the posts collection contain a post document which have an array of comment documents in it

 # adding a subDocument inside a already existing document using $push operator

```
> db.posts.update({ title:'postOne'},
... {
...  $push: { comments: { user : 'user1',
...  content : "the comment 1",
...  createdAt : Date() } }
... })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

The above query will add an array of embedded document under the key <comments> in the <posts> collection with document having title of <postOne>

```
> db.posts.find({title:'postOne'}).pretty()
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
        "category" : "news",
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
```

```
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)",
        "views" : 6,
        "comments" : [
                {
                        "user" : "user1",
                        "content" : "the comment 1",
                        "createdAt" : "Thu Aug 06 2020 12:47:06 GMT+0530 (IST)"
                }
        ]
}
```

 # deleting a document in mongoDB

Using db.collection.remove()

```
> db.posts.find({title:'updated post two'}).pretty()
{
        "_id" : ObjectId("5f29384d809db97f898d475e"),
        "title" : "updated post two",
        "body" : "updated post two body",
        "author" : "post two author"
}
```

Removing the above document in posts.

```
> db.posts.remove({title:'updated post two'})
WriteResult({ "nRemoved" : 1 })
```

The above query will remove the document in posts with title <updated post two>

 # $gt(greater than) operator in mongoDB

```
> db.posts.find({'views':{$gt:3}}).pretty()
```

The above query filters the collection using $gt(greater than) operator and gives all the documents where views are gt > 3.


```
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
        "category" : "news",
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)",
        "views" : 6,
        "comments" : [
                {
                        "user" : "user1",
                        "content" : "the comment 1",
                        "createdAt" : "Thu Aug 06 2020 12:47:06 GMT+0530 (IST)"
                }
        ]
}
{
        "_id" : ObjectId("5f29384d809db97f898d4760"),
        "title" : "postFour",
        "body" : "postFourBody",
```

```
        "category" : "entertainment",
        "tags" : [
                "entertainment",
                "events"
        ],
        "moderator" : {
                "name" : "will buyers",
                "role" : "publisher"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)",
        "views" : 17
}
```

 # $gte(greater than equal to) operator in mongoDB

```
> db.posts.find({'views':{$gte:17}}).pretty()
```

The above query filters the collection using $gte(greater than equal to) operator and gives all the documents where views are gt >=17.

```
{
        "_id" : ObjectId("5f29384d809db97f898d4760"),
        "title" : "postFour",
        "body" : "postFourBody",
        "category" : "entertainment",
        "tags" : [
                "entertainment",
                "events"
        ],
        "moderator" : {
                "name" : "will buyers",
                "role" : "publisher"
```

```
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)",
        "views" : 17
}
```

 # $lt(less than) operator in mongoDB

```
> db.posts.find({'views':{$lt:6}}).pretty()
```

The above query filters the collection using $lt(less than) operator and gives all the documents where views are lt <6.

```
{
        "_id" : ObjectId("5f29384d809db97f898d475f"),
        "title" : "postThree",
        "body" : "postThreeBody",
        "category" : "politics",
        "tags" : [
                "politics",
                "news"
        ],
        "moderator" : {
                "name" : "billy jean",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)",
        "views" : 2
}
```

 # $lt(less than or equal to) operator in mongoDB

```
> db.posts.find({'views':{$lte:6}}).pretty()
```

The above query filters the collection using $lte(less than or equal to) operator and gives all the documents where views are lte <=6.


```
{
	"_id" : ObjectId("5f2934ca809db97f898d475d"),
	"title" : "postOne",
	"body" : "postOne body is updated",
	"category" : "news",
	"tags" : [
		"news",
		"events"
	],
	"moderator" : {
		"name" : "jon doe",
		"role" : "author"
	},
	"date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)",
	"views" : 6,
	"comments" : [
		{
			"user" : "user1",
			"content" : "the comment 1",
			"createdAt" : "Thu Aug 06 2020 12:47:06 GMT+0530 (IST)"
		}
	]
}
{
	"_id" : ObjectId("5f29384d809db97f898d475f"),
	"title" : "postThree",
	"body" : "postThreeBody",
```

```
        "category" : "politics",
        "tags" : [
                "politics",
                "news"
        ],
        "moderator" : {
                "name" : "billy jean",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:58:29 GMT+0530 (IST)",
        "views" : 2
}
```


 # Find Specific Fields

```
> db.posts.find({},{'title':1,'moderator':1}).pretty()
```

The adobe query is same as

Select title, moderator from posts in sql

Fetching only specific fields from the documents of defined colletion


```
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        }
}
{
        "_id" : ObjectId("5f29384d809db97f898d475f"),
        "title" : "postThree",
        "moderator" : {
```

```
                "name" : "billy jean",
                "role" : "author"
        }
}
{
        "_id" : ObjectId("5f29384d809db97f898d4760"),
        "title" : "postFour",
        "moderator" : {
                "name" : "will buyers",
                "role" : "publisher"
        }
}
```

 # Find By Element in Array ($elemMatch)

```
> db.posts.find({
...   "comments": {
...     "$elemMatch": {
...       "user": "user1"
...     }
...   }
... }).pretty()
```

$elemMatch operator is used in above query to search a particular element in an array

The above query will search posts collection for a document where <comments> key array contains a document with key <user> having string as “user1”

```
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
```

```
        "category" : "news",
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)",
        "views" : 6,
        "comments" : [
                {
                        "user" : "user1",
                        "content" : "the comment 1",
                        "createdAt" : "Thu Aug 06 2020 12:47:06 GMT+0530 (IST)"
                }
        ]
}
```

Add Index

```
> db.posts.createIndex({title:'text'})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

The above query has created the index in mondoDb collection for searching using title

Searching the document on the basis of text index we just created which contain the titles

```
> db.posts.find({   $text: {     $search: "\"postOn\""     } }).pretty()
```

So postOn matches postOne Title


```
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
        "category" : "news",
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)",
        "views" : 6,
        "comments" : [
                {
                        "user" : "user1",
                        "content" : "the comment 1",
                        "createdAt" : "Thu Aug 06 2020 12:47:06 GMT+0530 (IST)"
                }
        ]
}
```

SQL vs NoSQL

sql(also called structured query language) ia query language that helps with crud operation on databases.
Helps you manipulation of DB

For ex- select id,name,age from persons;

SQL is used in relational databases
I.e the databases in which the data is stored in form of tables

We have a fixed schema of number of field pre defined for the table in relational database

All the records enter in the table have to follow the schema

The records in the table must be normalized
The type of relations are
One-to-one
One-to-many
many-to-many

So the two most important basic principle of the relational sql databases are
> there is fixed schema
> the tables have relation with each other

noSQL(not only sql) the db contains collection which are flexible and contain documents

```
For example
{
 name:"udit",
 age:27,
 gender:"male"
}
```

The documents are flexible and schema less each document is not dependent on each other
Each document inside the same collection can contain variable length of data.

There is no relation between the collection of db in noSQL but you can connect different collections and query them manually.

You put all info in a single document in the form of hierarchy.
For example
A post collection contains document which contains all info of post and an array of sub documents of comments


```
{
        "_id" : ObjectId("5f2934ca809db97f898d475d"),
        "title" : "postOne",
        "body" : "postOne body is updated",
        "category" : "news",
        "tags" : [
                "news",
                "events"
        ],
        "moderator" : {
                "name" : "jon doe",
                "role" : "author"
        },
        "date" : "Tue Aug 04 2020 15:43:30 GMT+0530 (IST)",
        "views" : 6,
        "comments" : [
                {
                        "user" : "user1",
                        "content" : "the comment 1",
                        "createdAt" : "Thu Aug 06 2020 12:47:06 GMT+0530 (IST)"
                }
        ]
}
```


These documents contain detail information about entity

Since there is no relation it helps in faster query
But there may be duplication of data in records contained in documents
Or the documents contain in different collections

If two collections are related if we change one document inside a collection we have to manually change other relation

To have consistency data

We must use noSQL when there are lots of reads but less writes from the databases.
We must not use noSQL
When collection or data in collections updated frequently
If we have a user document in multiple collections and these collection keeps on updating we should not use Sql

When there is a tight relation between entities we must use sql.
And when there are less reads but more writes we use SQL

SQL is not scalable and cannot be distributed among multiple servers but we can add more processors in the same server i.e SQL does not support Horizontal Scaling but can be scaled vertically.

noSQL on the other hand is highly flexible and scalable can be distributed among various servers.noSQl in loosely coupled so its best to use when we don't know what kind of data is stored in it.