# MongodB

- NOSQL dB
- open source, cross platform, document oriented dB written in C++.

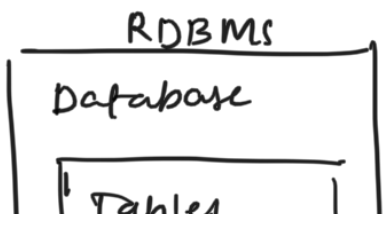
Purpose of mongodB-
Scalability
Performance
High Availability
Scaling from single server deployments to large complex.
multi site architectures

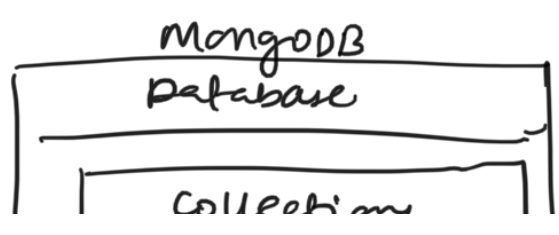
Develop faster
Deploy Easier
Scale Bigger

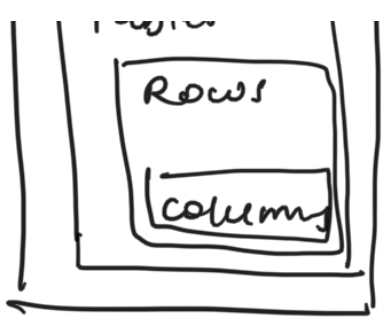
Document Oriented DB - data stored as documents

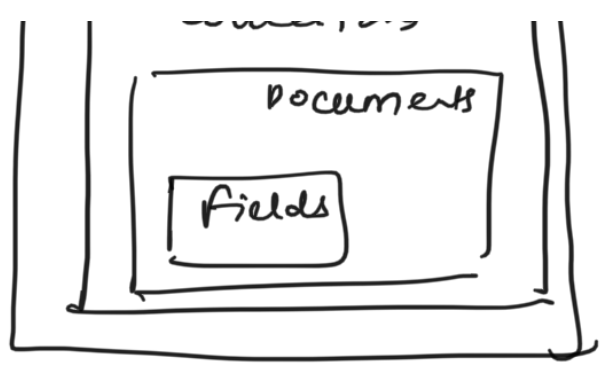
Features of mongodB.
- Support adhoc queries
- Indexing
- Replication
- Duplication of data.
- load Balancing
- Map Reduce & Aggregation
- schema less.
- High Performance
- Auto sharding for horizontal scalability
- Built in replication for high availability.
- 100 x faster than traditional dB's

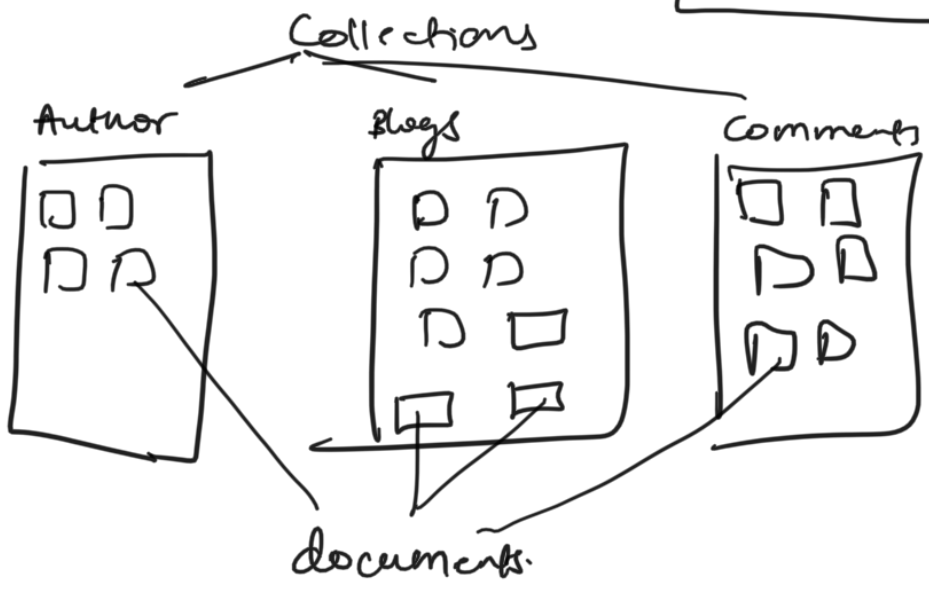
DataTypes
String
Integer
Boolean
Double
Min/Max keys → compare a value against lowest & highest bson elements
Arrays
Objects
Null
Symbol
Date






Documents are BSON
can be nested.
Have a _id field -> identifier.

Create DB
There is no createdB.
- Creates automatically when you save the value into the defined collection at first time

use Database-name → if db exists it will return that
Show dbs → displays existing dbs with atleast 1 document
db → shows current db in use.

Drop DB
db.dropDatabase() → In case you haven't selected any, default 'test' db is dropped

Create collection
db.createCollection(name, options)
size, max, autoIndexId, capped

show collections.
db.SSIT.insert({"name": "Manu"}) → automatically creat SSIT if not exits.
show collections
db.collection_name.find() → to see inserted document

Drop collection
db.collection_Name.drop()

Insert Documents.
db.collection.insert() //or save() or update() do the same

db.collection_name.insert(document).

eg db.javapoint.insert(
{
name :"Harry"
course :"mongo"
})

Returns a WriteResult obj on successful insertion

To view inserted do
db.collection_name.find()

Insert multiple docs..
-pass an array of docs.

```
var AllCourses =
[
  {
    course: "java",
    batch: 20
  },
  {
    course: "C++"
    batch: 30
  }
]
```

db.javapoint.insert(AllCourses);
Returns a BulkWriteResult Object.

Insert with bulk—:
var bulk = db.javapoint.initializeUnorderedBulkOp();

Add insert ops
bulk.insert( { })
bulk.insert( { })

bulk.execute()

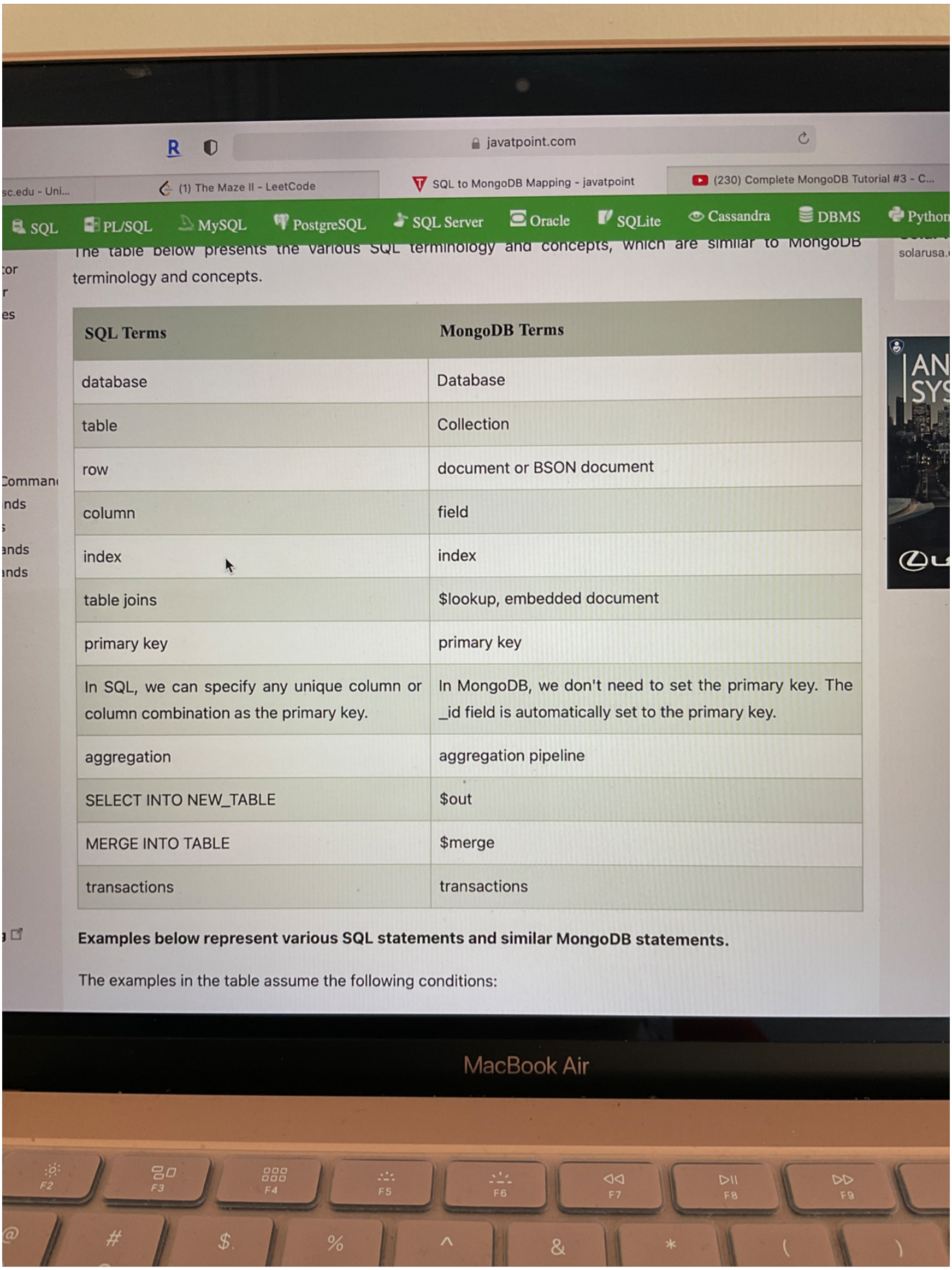
Update document
db.collection_name.update(selectioncriteria, updated data).
db.mydb.update({'course':'java'}, {$set: {'course':'android'}})

Delete documents.
db.collection.name.remove(deletion criteria)
db.mydb.remove({}) //removes all documents from mydb
db.mydb.remove({type: 'PL'}) → remove all with type PL
,1) → remove single doc with type PL

Query documents.

db.collection.find() → retrieve all docs from a collection
-Returns a cursor to retrieved docs.

db.collectionname.find({}) → retrieve all docs from collection



db.insertOne(Obj)
db.insertMany([{},{}])
db.insert()

db.collectionname.find() → limits to 20
it 20-40
it 40-60 and so on.

db.author.find({name:"kaustubh"})
db.author.find({name:"kaustubh", rating:9})
& 

db.author.find({name:"kaustubh"}, {title:1, author:1})
db.books.find({}, {title:1, author:1})

db.books.findOne({_id: ObjectId("6224...")})
db.books.findOne({author:"Harry"})

db.books.find().count() → no. of docs returned
db.books.find({name:"kaustubh"}).count()
db.books.find().limit(3) → limit to 3 results

db.books.find().limit(3).count() → We can chain methods together

db.books.find().sort({title:1})
asc -1 desc
db.books.find().sort({title:1}).limit(3)

Operators are denoted by $ signs
db.books.find({rating: {$gt:7}})
$gt >
$lt <
$gte >=
$lte <=

db.books.find({rating: {$gt:7}, author:"xyz"})
1 & 2

db.books.find({$or:[{rating:7},{rating:9}]})

db.books.find({$or:[{pages:{$lt:300}}, {pages:{$gt:400}}]})

db.books.find({rating: {$in:[7,8,9]}})

db.books.find({rating: {$nin:[7,8,9]}})

db.books.find({genre: "magic"}) → return book with magic as one of the genre

db.books.find({genre: ["magic"]}) → magic as the only genre

——"—— ——: ["magic", "fantasy"]) → both fantasy & magic in genre.

db.books.find({genre: {$all: ["magic", "fantasy"]}})
→ get books containing all of the genres magic, fantasy.
- just only magic, fantasy.
can have more but must contain those 2.

db.books.find({"reviews.name": "luigi"})
- any book containing review with name luigi

db.books.delete({id: objectId("621...")})
↓ delete a document with that id
db.books.deleteOne( —"— — )
db.books.deleteMany({author: "Terry Patche"})

db.books.updateOne({_id = objectId("621...2f")}, ← which book
{$set: {rating: 10, pages: 360}}) ← what to update.

db.books.updateMany({author: "Terry"}, {$set: {author: 'Terry P'}})

db.books.updateOne({_id = objectId("...")},
{$inc: {pages: 2}})

$inc: {pages: -2} → decrement

db.books.updateOne({_id: objectId("...")},
{$pull: {genres: "fantasy"}})
(removes fantasy from genres)

push an object — {$push: {genres: "fantasy"}}

db.books.updateOne({_id: objectId("...")},
{$push: {genres: {$each: ["1", "2"]}}})
(push many objects at once)

db.books.createIndex({rating: 8})

db.books.dropIndex({rating: 8})

db.books.getIndexes()

## Aggregation

- process data records and return computed results.
- group values from multiple documents together and can perform a variety of operations on grouped data to return a single result.

```
db.collectionname.aggregate(AGGREGATE_OPERATION)
```

```
db.mycol.aggregate([{$group : {_id : "$by-user",
                               num_tutorial : {$sum : 1333}}])
```

```
db.mycol.aggregate([{$group : {_id : "$by-user",
                               num_tutorial : {$min : "$likes"333}}])
```

** $sum, avg, min, max, push, addtoset, first, last.

$project, match, group, sort, skip, limit, unwind

### mapreduce:

- a data processing paradigm for condensing large volumes of data into useful aggregated results.

```
db.collection.mapReduce(
    function() {emit(key, value);}   //map fn.
    function(key, values) {return reducefn},
        {
          out: collection,                    — reduce fn.
          query: documents,
          sort: document,
          limit: number
        }
)
```

Bookmarks Develop Window Help

tutorialspoint.com

University of Southern California... | mongodb interview questions - Google Search | MongoDB

Category | Search tutorials ... | Prime Packs | Courses | eBooks | Libr

```
    function(key,values) {return reduceFunction}, {   //reduce func
       out: collection,
       query: document,
       sort: document,
       limit: number
    }
)
```

The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

In the above syntax –

- **map** is a javascript function that maps a value with a key and emits a key-value pair
- **reduce** is a javascript function that reduces or groups all the documents having the same key
- **out** specifies the location of the map-reduce query result
- **query** specifies the optional selection criteria for selecting documents
- **sort** specifies the optional sort criteria
- **limit** specifies the optional maximum number of documents to be returned

## Using MapReduce

ckup
nt
ships
se References
d Queries
ing Queries
Operations
ced Indexing
ing Limitations
ctId
Reduce
Search
ular Expression

See it, Sec
Manage it
Cisco

See it, Secu
Manage it
Cisco

From the m
PERS

MacBook Air

F2 F3 F4 F5 F6 F7 F8 F9 F10

# $ % ^ & * ( )

ObjectId - 12byte BSON

First 4 bytes → seconds since unix epoch

next 3 → machine identifier

next 2 → process id.

last 3 → random counter value