MongoDb-Documentation

Mongodb is the document oriented NOSQL used for high volume data storage.
It was initially developed as PAAS(platform as a service)
No concepts of schema and relationship
Faster than traditional db
scalable

NOSQL DB PRODUCTS → **redis,mongodb,** couchbase,amazonDynamodb,cassandra,apache drill,cloudant

MONGO DB MANAGING TOOL → **Studio 3T**,Robo 3T, MongoDb compass

RDBMS	**MONGODB**
Table	**Collection**
Column	**Field**
Value	**Value**
Row	**Document/Object**

Document →key value pair
Collection → set of document and function(same as table)
JSON → javascript object notation(human readable)

MongoDB download link :
https://www.mongodb.com/try/download/community
Studio 3T download link : https://studio3t.com/download/

→ Open Studio 3T
→Click on connect
→ Click on new connection
→ Enter connection name,host,port(27017)
→ save
→ intellishell→

```
                → show dbs
                → db
                → use dbname
                → db.createCollection("collectioname")
                →db.collectioname.insert(
                  {
                   "Name" : "chail" ,
                   "Mobile" : 987456311
                   }
                   )
                → db.dropDatabase()
                → db.collectioname.drop()
                → db.collectioname.find()
                → db.collectioname.findOne()
//projection
                → db.collectioname.find({ "city" : "Ahemdabad"})
                → db.collectioname.find(
                  { } ,
                  { "city" : "Ahemdabad", "Mobile": 9}
                  )
                → db.collectioname.find(
                  {"state" : "Gujarat" } ,
                  { "city" : "Ahemdabad", "Mobile": 9}
                  )
```

```
→db.collectioname.insertOne(
  {
   “Name” : ”avni” ,
   “Mobile” : 987456311
   }
   )

→db.collectioname.insertMany( [
   {
   “Name” : ”avni” , “Mobile” : 987456311
   },
   {
    “Name” : ”chail” ,“Mobile” : 987456311
   },
   {
    “Name” : ”manali” ,“Mobile” : 987456311
   }
   ]
   )
```

All in one :

```
db.createCollection(“allinone”)
Db.allinone.insert (
{
 “Name” : “chail”, //string
“Mobile” : 9632587410 ,  //number
“Doj” : Date() , //date
“Salary” : 2500.00 , //double
“Roles” : [ “admin” , “user”] , //array
"Address" :
 {
   "City" : "Ahemdabad",
   "Area" : "Gift city"
} , //object
"isAdmin" : false ,  //boolean
"Dor" : null //empty
}
)
```

Query Operator :

1) **IN** → db.collectioname.find({ "city" : {$in : ["pune","vapi"]}})
2) **Less than** → db.collectioname.find({ "city" : {$lt : 30 } })
3) **Grater than** → db.collectioname.find({ "city" : {$gt : 30 } })
4) **Less than equal** → db.collectioname.find({ "city" : {$lte : 30 } })
5) **Grater than equal** → db.collectioname.find({ "city" : {$gte : 30 }})
6) **Not equal to** → db.collectioname.find({ "city" : {$ne : 30 } })
7) **OR** → db.collectioname.find({ $or : [{ "city" : "pune"} , { "marks" : 30 }] })
8) **AND** → db.collectioname.find({ $and : [{ "city" : "pune"} , { "marks" : 30 }] })
   → db.collectioname.find ({ "city" : "pune" , "marks" : 30 })

Functions →

1) **Limit** →db.collectioname.find ({}).limit(3)
2) **Sort** → db.collectioname.find ({}).sort({ "name" : 1 }) //ascending
   → db.collectioname.find ({}).sort({ "moble" : -1 }) //descending
3) **Skip** → db.collectioname.find ({}).skip(3) //top 3 skip
4) **Hybirid** → db.collectioname.find({"marks" : { $gt :20}} , { "name" :1 , "marks" : 1 , "_id" : 0}).sort({ "marks" : -1 }).limit(3).skip(3)

Note: negative value in limit and skip doesn't makes any sense.

Update → db.collectioname({ “city” : “pune” },{$set : { “mobile” :9632587410 , “marks” : 34 }})

UpdateOne → db.collectioname({ “city” : “pune” },{$set : { “mobile” :9632587410 }})

Update → db.collectioname({ “city” : “pune” },{$set : { “mobile” :9632587410 }} , { “multi” : true })

UpdateMany → db.collectioname({ “city” : “pune” },{$set : { “mobile” :9632587410 }})

removeAllRecords → db.collectioname.remove({})

removeMultipleRecords → db.collectioname.remove({ “city” : “AMD”})

removeAllRecords → db.collectioname.deleteMany({ })

removeMultipleRecords → db.collectioname.deleteMany({ “city” : “AMD”})

remove single first get record → db.collectioname.remove({ “city” : “AMD”},1)

remove single records → db.collectioname.deleteOne({ “city” : “AMD”})

Count total doc → db.collectioname.count()

Distinct → db.distinct(“city”)

Copy to another collection →db.oldcollection.copyTo(“new collection”)

Replace One → db.collection.replaceOne({ “id” : 104 } , { “name” : “chail” }) //unpassed column value set to null

Rename → db.collectioname.renameCollection(“new collection name”)

Get all collectioname → db.getCollectionNames()

Get current db Name → db.getName()

Version → db.version()

INDEXING → used for fast processing.
MongoDB uses indexing in order to make the query processing more efficient. If there is no indexing, then the MongoDB must scan every document in the collection and retrieve only those documents that match the query. Indexes are special data structures that stores some information related to the documents such that it becomes easy for MongoDB to find the right data file. The indexes are order by the value of the field specified in the index.

db.collectioname.ensureIndex({ “id” : 1 }) //ascending order
db.collectioname.find({ })
db.collectioname.ensureIndex({ “id” : -1}) //descending order
db.collectioname.find({ })
db.collectioname.dropIndex({ “id” : 1}) //delete index

Aggregation → **In MongoDB, aggregation operations process the data records/documents and return computed results. It collects values from various documents and groups them together and then performs different types of operations on that grouped data like sum, average, minimum, maximum, etc to return a computed result. It is similar to the Aggregate function of SQL.**
MongoDB provides three ways to perform aggregation

- **Aggregation pipeline**
- **Map-reduce function**
- **Single-purpose aggregation**

db.collectioname.aggregate([{ $group : { _id : “$city” , TotalCity : {$sum :1 }}}])

How to take backup of database in mongodb

Go to terminal →

>**Mkdir mongobackup**

>**Cd mongobackup**

>**mongodump** → all database will be saved as bson format

>**mongodump** –db dbname →single db saved

>**mongorestore** → replace and restore the database

>**mongoretore** –db dbname dump/dbname → restore single db

>**mongodump** –db dbname –collection collectioname → particular collection form db will be saved.

>**mongorestore** –db dbname –collection collectioname dump/dbname/collectioname.bson → restore with particular collection name

>**mongorestore** –db dbname –collection collectioname dump/dbname/cmongorestoreollectioname.bson –drop → drop old collection and restore .

Aggregation pipeline → In MongoDB, the aggregation pipeline consists of stages and each stage transforms the document. Or in other words, the aggregation pipeline is a multi-stage pipeline, so in each state, the documents taken as input and produce the resultant set of documents now in the next stage(id available) the resultant documents taken as input and produce output, this process is going on till the last stage

Stages: Each stage starts from stage operators which are:

$match: It is used for filtering the documents can reduce the amount of documents that are given as input to the next stage.

$project: It is used to select some specific fields from a collection.

$group: It is used to group documents based on some value.

$sort: It is used to sort the document that is rearranging them

$skip: It is used to skip n number of documents and passes the remaining documents

$limit: It is used to pass first n number of documents thus limiting them.

$unwind: It is used to unwind documents that are using arrays i.e. it deconstructs an array field in the documents to return documents for each element.

$out: It is used to write resulting documents to a new collection

Expressions: It refers to the name of the field in input documents for e.g. { $group : { _id : “$id“, total:{$sum:”$fare“}}} here $id and $fare are expressions.

Accumulators: These are basically used in the group stage

sum: It sums numeric values for the documents in each group

count: It counts total numbers of documents

avg: It calculates the average of all given values from all documents

min: It gets the minimum value from all the documents max: It gets the maximum value from all the documents

first: It gets the first document from the grouping

last: It gets the last document from the grouping