
mongoDB.


- Installing Database

  Download MongDB-

  1.Go to the MongoDB download page:

  Vist: Download MongoDB Community Server | MongoDB

  2.Choose Your operating system:

  Select the version that matches your computer's operating system(windows,macOS,Linux).

  Download the MongoDB Community Server and install

# CRUD (Create, Read, Update, Delete)

- Show all databases- db.getMongo().getDBs()
- Create Database- use(‘shop’)
- Create collection- db.createCollection(‘product’)
- Create document- db.product.insertOne({

```
    item:”apple”,
    type:”fruits”,
    price:80
})
```

- Create many document- db.product.insertMany([

```
{
    item:”grap”,
    type:”fruits”,
    price:70
},{
    item:”orange”,
    type:”fruits”,
    price:90
},{
    item:”tomato”,
    type:”vegitable”,
    price:0
}
])
```

- Show the documents- db.product.find()
- Show First document- db.product.findOne()
- Show N-documents- db.product.find().limit(2)
- Searching/Filtering-

db.product.find({type:” vegitable”})

- Projection :-

In MongoDB, **projection** means choosing which fields you want to see when you retrieve data from a collection. you can **select specific fields** to display.

- Projection- db.product.find({},{item:1})

( In this method only show the items in the product document)

- Projection- **db.product.find({},{item:0})**

( In this method hide the item in the product document and other all will show)

- Projection- **db.product.find({},{_id:0,item:1})**

( In this method the id will hide and only show the item in the product document)

- Projection- **db.product.find({type:”fruits”},{_id:0,item:1})** (In the method only the item will show the filter of type “fruits” only the fruits item will show the id will also hide)

- Sort in Ascending using item - **db.product.find().sort({item:1})**

- Sort in descending using item -

```
db.product.find().sort({item:-1})
```

- Sort in Ascending using item and price -

```
db.product.find().sort({item:1,price:1})
```

# Comparison:-

- Check is equal- **db.product.find({price:{$eq:100}})**

  (this means that we are finding a data from product collection of price is equal to 100)

- Check is not equal- **db.product.find({price:{$ne:100}})**

  (this means that we are finding a data from product collection of price is notequal to 100)

- Check is greater than-

db.product.find({price:{$gt:50}})

(this means that we are finding a data from product collection of price is greater than to 50)

- Check is greater than or equal-

db.product.find({price:{$gte:50}})

(this means that we are finding a data from product collection of price is greater than and equal to 50)

- Check is less than- db.product.find({price:{$lt:50}})

  (this means that we are finding a data from product collection of price is less than 50)

- Check is less than or equal- db.product.find({price:{$lte:50}})

  (this means that we are finding a data from product collection of price is less than and equal to 50)

- Pick data by finding by there price values-
  db.product.find({price:{$in:[50,100,90]}})

- Pick data by finding by there item names-
  db.product.find({item:{$in:[“apple”,”carrot”,”orange”]}})

- Finding by there item name and do not pick this items only pick the balance items-
  db.product.find({item:{$nin:[“apple”,”carrot”,”orange”]}})

# Logical_mongoDB

- db.product.find({
  $and:[
  {price:{$gte:30}},
  {price:{$lte:50}}
  ]
  })

(It’s a condition that get data of price range of 30 to 50 using ‘and’ queries )

- db.product.find({
  $and:[
  {price:{$gte:50}},
  {type:”vegetable”}
  ]
  })

(It’s a condition that get data of price grater than 50 and type will be “vegetable” using ‘and’ queries )

- If we want a product that price range of less than 30 and greater than 90 that time we use “or” operator to find them :-

```
db.product.find({
        $or:[
            {price:{$gt:90}},
             {price:{$lt:40}},
          ]
        })
```

- If we not want a product that price range of less than 30 and greater than 90 that time we use “nor” operator to find them

```
db.product.find({
        $nor:[
            {price:{$gt:90}},
             {price:{$lt:40}}
            ]
      })
```

- If we not need this “item” or any other cases we will use “not” operator:-

db.product.find({item :{$not:{$in:[“apple”]}}})

- Delete first Occurance-

db.product.deleteOne({type:’vegitable’})

- Delete Many: - db.product.deleteMany({type:”fruits”})

- Delete all documents in collection: -

  db.product.deleteMany({})

- Delete collection:- db.product.drop()

- Delete database:- db.dropDatabase()

# Evaluation:-

```
db.customer.insertMany([
        {_id:1,name:”Sreejith ca”},
        {_id:2,name:”arun b”},
        {_id:3,name:”kichu binoy”},
        {_id:4,name:”bincy ba”},
        {_id:5,name:”anu aby”},
        {_id:6,name:”anna sunil”}
])
```

We have many data in customer collection we want to evaluate this collection

- Find by Name start from “a”:-

```
db.customer.find({name:{$regex:/^a/}})
```

- Find by Name ends from “a”:-

```
db.customer.find({name:{$regex:/a$/}})
```

- Name contains with “an”:-

```
db.customer.find({name:{$regex:/an/}})
```

- Name contains with “AN” with ignore case sensitive:-
  db.customer.find({name:{$regex:/An/i}})

- Find by second letter is “n” :-
  db.customer.find({name:{$regex:/^.n/i}})

- Find by third letter is “n” :-
  db.customer.find({name:{$regex:/^..n/i}})

- Find last third letter is “n” :-

```
db.customer.find({name:{$regex:/^n..$/i}})
```

- Name starts with ‘a’ or ‘b’:-

```
db.customer.find({name:{$regex:/^[a,b]/i}})
```

- Find the name that second letter between ‘A’ to ‘D’:-

```
db.customer.find({name:{$regex:/^..[A-D]/i}})
```

# Find by Text:-

- Create index in documents:-

  db.customer.createIndex({name:”text”})

- Name contains (word) :-

  db.customer.find({$text:{$search:”sunil”}})

- Name contains (word) case senstive :-

  db.customer.find({$text:{$search:”sunil”,caseSensitive:true}})

# Update Operators:-

- Create a collection that want to update –

```
use (‘blog’)
 db.posts.insertMany([
        {_id:1, title:”post-1”, likes:0, pdate:Date() },
        {_id:2, title:”post-2” ,likes:1, pdate:Date() },
        {_id:3, title:”post-3”, likes:2, pdate:Date() },
         {_id:4, title:”post-4”, likes:3, pdate:Date() },
         {_id:5, title:”post-5”, likes:4, pdate:Date() }
])
db.posts.find()
```

- Searching , updations :-

```
db.posts.updateOne({},{
        $currentDate:{pdate:{$type:”date”}}
        })
```

- Searching , updations by condition:-

```
db.posts.updateOne({_id:2},{
        $currentDate:{pdate:{$type:”timestamp”}}
        })
```

- Searching , update many:-

```
db.posts.updateMany({},{$set:{pdate:Date()}})
```

- Add new field:-

```
db.posts.updateMany({},{$set:{subject:”news”}})
```

- Remove field:-

```
db.posts.updateMany({},{$unset:{subject:””}})
```

- Rename the field:-

```
db.posts.updateMany({},{$rename:{ pdate:”postdate”}})
```

- Update first likes :-

```
db.posts.updateOne({},{$set:{like:5}})
```

- Update likes with condition:-

```
db.posts.updateOne({title:’post-2’},{$set:{like:5}})
```

- Increment method :-

```
db.posts.updateMany({},{$inc:{ like:1} })
```

- decerment method :-

```
db.posts.updateMany({},{$inc:{ like:-1} })
```

- Add an array:-

```
db.posts.updateMany({},{$set:{ like_by:[“anu”,”manu”]} })
```

Array methods:-

Add new datas to array:-

```
db.posts.updateOne({_id:1},{ $addToSet:{ like_by:”sanu”} })
```

Upadate using push:-

```
db.posts.updateOne({_id:1},{$push:{like_by:’meenu’}})
```

- Remove first from array:-

```
db.posts.updateOne({_id:1},{ $pop:{ like_by:-1}})
```

- Remove last from array:-

```
db.posts.updateOne({_id:1},{ $pop:{ like_by:1}})
```

- Remove Matched value :-

```
db.posts.updateOne({_id:1},{$pull:{like_by:’sanu’}})
```

# Aggregates:-

- To count the items in product :-

```
db.product.aggregate({$count:”item” })
```

- Total price :-

```
db.product.aggregate([
                        { $group:{ _id:null,
                        TotalPrice:{
                         $sum:”$price”
                                    }
                         }}
                    ])
```

- Average price :- db.product.aggregate([
  {$group:{ _id:null,
  TotalPrice:{
  $avg:”$price”
  }
  }}
  ])
- Average price :- db.product.aggregate([
  {$group:{ _id:“$type”,
  TotalPrice:{
  $avg:”$price”
  }
  }}
  ])

- maximum price :- db.product.aggregate([
  {$group:{ _id:“$type”,
  MaximumPrice:{
  $max:”$price”
  }
  }}
])
- minimum price :- db.product.aggregate([
  {$group:{ _id:“$type”,
  MinimumPrice:{
  $min:”$price”
  }
  }}
])

- Add quantity to collection:-

```
db.product.updateMany({},{ $set:{quantity:10}})
```

- Create sum by multiply with quantity:-

```
db.product.aggregate([
        {$group:{ _id:'$type',
         TotalPrice:{
           $sum:{
                   $multiply:["$price","$quantity"]}
       }
      }}
])
```

Show all: - `db.product.aggregate([])`

Limit: - `db.product.aggregate([{ $limit:5}])`

- Projection:-

  db.product.aggregate([{ $project:{ _id:0,item:1}}])

- Match(filtering/searching) only show the matching (fruits)document in this case:-

  db.product.aggregate([{ $match:{ type:”fruits”}}])

- Add fields:-

  db.product.aggregate([{ $addFields:{Total:{$multiply:[“$price”,”$quantity”]}
  }}])

(This case we created a new field that total price of per item in using multiply the price and quantity )

- Out (Create a sibling database&collection)

```
db.product.aggregate([{ $out:{db:”shop_1”,coll:’stock’}}
                                ])
```

- How to subtract:-

```
db.product.aggregate([{ $addFields:{
 disount:5,
  subtracted:{ $subtract:[“$price”,5] } } } ])
```

- How to subtract price precantage:-

```
db.product.aggregate([{ $addFields:{
disount:”5%”,
subtracted:{ $subtract:[“$price”,{$multiply:[“$price”,0.05]}] } } } ])
```

- Document Relations(OneToOne,OneToMany,ManyToMany)
- OneToOne Relation(using Reference):-

```
use('people')
db.person.insertMany([
    {_id:1,name:"alen",email:alen@gmail.com},
    {_id:2,name:"aleena",email:aleena@gmail.com},
   {_id:3,name:"manu",email:manu@gmail.com},
   {_id:4,name:"shyam",email:shyam@gmail.com},
   {_id:5,name:"kiran",email:kiran@gmail.com}
                                ])
```

- db.adhar.insertMany([
  {_id:11,adharno:”abcd1234”,imag:”y”,biometrics:”y”,person_id:1},
  {_id:12,adharno:”efghi5678”,imag:”y”,biometrics:”y”,person_id:2},
  {_id:13,adharno:”jklm6855”,imag:”y”,biometrics:”y”,person_id:3},
  {_id:14,adharno:”nopq7849”,imag:”y”,biometrics:”y”,person_id:4},
  {_id:15,adharno:”ixyz2255”,imag:”y”,biometrics:”y”,person_id:5},
  ])

db.person.fnd()

db.adhar.find()

- Show persons with adhar-details:- db.person.aggregate([

```
{
    $lookup:{
        from:’adhar’,
        localField:’_id’,
        foreignField:’person_id’,
         as:’adhar-details’
      }
}])
```

- Show adhar with person-details:- db.adhar.aggregate([

```
{
    $lookup:{
        from:’person’,
        localField:’person_id’,
        foreignField:’_id’,
         as:’person-details’
      }
}])
```

- Show persons with adhar(only adhar no) using pipeline and projection:-

```
db.person.aggregate([
                {
                   $lookup:{
                       from:’adhar’,
                       localField:’_id’,
                       foreignField:’person_id’,
                       pipeline:[{$project:{_id:0,adharno:1}}],
                        as:’adhar’
                      } }
                ])
```

- Show persons with adhar(only adhar no) without using pinpeline :-

```
db.person.aggregate([
                {
                   $lookup:{
                       from:’adhar’,
                       localField:’_id’,
                       foreignField:’person_id’,
                       as:’adhar’
                     } },
            {$project:{_id:0,name:1,” adhar.adharno”:1}}
                ])
```

- Show persons with adhar(only adhar no) using pinpeline , projection and match:-

```
db.person.aggregate([
                    { $match:{ name:’alen’}},
                    {
                        $lookup:{
                            from:’adhar’,
                            localField:’_id’,
                            foreignField:’person_id’,
                            pipeline:[{$project:{_id:0,adharno:1}}],
                             as:’adhar’
                          } }
                    ])
```

- Implement relation using Emdedded Document:-

- db.dropDatabase()

- db.person.insertMany([
  { _id:1,name:’alen’,email:’alen@gmail.com’,
  adhar:{adharno:’ofggjlj’,imag:’y’,biometrics:’y’}},
  {_id:2 , name:’aleena’ , email:’aleena@gmail.com’,
  adhar:{adharno:’avbhdjd12’,imag:’X’,biometrics:’X’}}
  ])

- Implement one to many Relation using Reference:-
- use('people')
- db.dropDatabase()

- db.person.insertMany([
  { _id: 1, name: "Alen", email: "alen@mail.com" },
  { _id: 2, name: "Aleena", email: "aleena@mail.com" },
  { _id: 3, name: "Manu", email: "manu@mail.com" },
  { _id: 4, name: "Shyam", email: "shyam@mail.com" },
  { _id: 5, name: "Kiran", email: "kiran@mail.com" }])

```
db.vehicle.insertMany([
   { _id: 5001, vtype: "Two Wheeler", reg: "KL42 7788", person_id: 1 },
   { _id: 5002, vtype: "Four Wheeler", reg: "KL42 8788",person_id: 1 },
   { _id: 5003, vtype: "Four Wheeler", reg: "KL43 12788", person_id: 3 },
   { _id: 5004, vtype: “Two Wheeler", reg: "KL43 23788", person_id: 3 },
  { _id: 5005, vtype: "Four Wheeler", reg: "KL69D1317",person_id: 2 },
  { _id: 5005, vtype: "Four Wheeler", reg: "KL69D1317",person_id: 2 },
        ])
```

Show Persons:- db.person.find()

Show Vehicles:- db.vehicle.find()

Show Persons with Vehicle details:-

```
db.person.aggregate([
  {
    $lookup: {
      from: 'vehicle',
      localField: '_id',
      foreignField: 'person_id',
      as: 'VehicleDetails'
    }
  }
])
```

Show Vehicles with Person details:-

```
db.vehicle.aggregate([
    {
        $lookup: {
            from: 'person',
            localField: 'person_id',
            foreignField: '_id',
            as: 'PersonDetails'
        }
    }
])
```

Show Persons with Vehicle details (contains 77 in "reg" )

hide _id from vehicle details:-

using pipeline:-

```
db.person.aggregate([
   {
      $lookup: {
         from: 'vehicle',
         localField: '_id',
         foreignField: 'person_id',
```

```
pipeline: [
            {
                $match: {
                    reg: { $regex: /77/ }
                }},{
                $project: {
                    _id: 0, person_id: 0
                }}],
            as: 'VehicleDetails'
        }])
```

Show Persons with Vehicle details (contains 77 in "reg" )

hide ids

without pipeline

```
db.person.aggregate([
   {       $lookup: {
          from: 'vehicle',
           localField: '_id',
           foreignField: 'person_id',
           as: 'VehicleDetails'
       }
   },
```

```
{
        $match: {
            'VehicleDetails.reg': { $regex: /77/ }
        }
    },
    {
        $project: {
            _id: 0, 'VehicleDetails._id': 0, 'VehicleDetails.person_id': 0
        }
    }
])
```

Show Vehicle Reg-No with Person Name (contains 77 in "reg"  )
hide ids

```
db.vehicle.aggregate([
   {
      $match: {reg: { $regex: /77/ }}
   },{$lookup: {
         from: 'person',
         localField: 'person_id',
         foreignField: '_id',
         as: 'PersonDetails'
      }},{
      $project: {
         reg: 1, 'PersonDetails.name': 1, _id: 0
      }
   }
])
```

Implement Relation using Embedded Document:-

```
db.dropDatabase()
db.person.insertMany([
   {
      _id: 1, name: "Alen", email: "alen@mail.com", vehicle: [
         { vtype: "Two Wheeler", reg: "KL42 7788" },
         { vtype: "Four Wheeler", reg: "KL42 8788" }]
   },
   { _id: 2, name: "Aleena", email: "aleena@mail.com",
vehicle: [] },
```

```
 {
        _id: 3, name: "Manu", email: "manu@mail.com", vehicle: [
           { vtype: "Four Wheeler", reg: "KL43 12788" }]
    },
    {
        _id: 4, name: "Shyam", email: "shyam@mail.com", vehicle: [
           { vtype: "Four Wheeler", reg: "KL43 23788" }]
    },
    { _id: 5, name: "Kiran", email: "kiran@mail.com", vehicle: [] }
])
db.person.find()
```

- Many to many :-

```
use('people')
db.dropDatabase()
```

Implement Relation using many to many Reference:-

```
db.person.insertMany([
    { _id: 1, name: "Alen", email: "alen@mail.com" },
    { _id: 2, name: "Aleena", email: "aleena@mail.com" },
    { _id: 3, name: "Manu", email: "manu@mail.com" },
    { _id: 4, name: "Shyam", email: "shyam@mail.com" },
    { _id: 5, name: "Kiran", email: "kiran@mail.com" }])
```

```
db.address.insertMany([
   { _id: 1001, house_no: "YU-345", place: "Kochi", pin: 4582361, person_id: [1, 2] },
   { _id: 1002, house_no: "YU-145", place: "Kochi", pin: 4582361, person_id: [3, 4] },
   { _id: 1003, house_no: "FG-435", place: "Kollam", pin: 8982361, person_id: [1] }
])
```

Show Persons :- db.person.find()

Show Address:- db.address.find()

Show Persons & Address:-

```
db.person.aggregate([
   {
      $lookup: {
         from: 'address',
         localField: '_id',
         foreignField: 'person_id',
         as: 'AddressDetails'
      }
   }
])
```

Show Address & Persons :-

```
db.address.aggregate([
    {
        $lookup: {
            from: 'person',
            localField: 'person_id',
            foreignField: '_id',
            as: 'PersonalDetails'
        }
    }
])
```

Address & Persons living in address_id: 1001,hide id :-

```
db.address.aggregate([
   {
      $match: {
         _id: 1001
      }},
   {$lookup: {
         from: 'person',
         localField: 'person_id',
         foreignField: '_id',
         as: "Persons"
      }},
```

```
{
        $project: {
          _id: 0, person_id: 0,'Persons._id': 0
        }
    }
])
```

• Implement Relation using Embedded Document:-

```
db.dropDatabase()
db.person.insertMany([
   {
      _id: 1, name: "Alen", email: "alen@mail.com", address: [
         { _id: 1001, hose_no: "YU-345", place: "Kochi",
 pin: 4582361 },
         { _id: 1003, hose_no: "FG-435", place: "Kollam",
 pin: 8982361 }
      ]  },{
      _id: 2, name: "Aleena", email: "aleena@mail.com", address: [
         { _id: 1001, hose_no: "YU-345", place: "Kochi",
pin: 4582361 }
      ] },
```

```
{
    _id: 3, name: "Manu", email: "manu@mail.com", address: [
        { _id: 1002, hose_no: "YU-145", place: "Kochi", pin: 4582361 }
    ] }, {
    _id: 4, name: "Shyam", email: "shyam@mail.com", address: [
        { _id: 1002, hose_no: "YU-145", place: "Kochi", pin: 4582361 }
    ] },{
    _id: 5, name: "Kiran", email: "kiran@mail.com", address: [
        { _id: 1002, hose_no: "YU-145", place: "Kochi", pin: 4582361 }
    ]   }])
```

Show all:-

```
db.person.find()
```

Show Manu's details:-

```
db.person.find({ name: 'Manu' })
```

Show Persons from Kollam:-

```
db.person.find(
{ 'address.place': 'Kollam' },
{ name: 1, email: 1, _id: 0 })
```

Find by address text:-

```
db.person.find(
{ 'address.place': {$regex : /kol/i} },
{ name: 1, email: 1, _id: 0 })
```

- All Relations :-

```
use('people’)

db.dropDatabase();
```

Implement Relation using Reference:-

```
db.person.insertMany([
    { _id: 1, name: "Alen", email: "alen@mail.com" },
    { _id: 2, name: "Aleena", email: "aleena@mail.com" },
    { _id: 3, name: "Manu", email: "manu@mail.com" },
    { _id: 4, name: "Shyam", email: "shyam@mail.com" },
    { _id: 5, name: "Kiran", email: "kiran@mail.com" }]);
```

- one to one (adhar to person):-

```
db.adhar.insertMany([
   { _id: 11, adharno: "ABCD12345", image: 'Y', biometrics: 'Y’,
    person_id: 1 },
   { _id: 12, adharno: "XYZT34566", image: 'Y', biometrics: 'Y’,
    person_id: 2 },
   { _id: 13, adharno: "ERFTY52345", image: 'Y', biometrics: 'Y’,
   person_id: 3 },
   { _id: 14, adharno: "RTERG12345", image: 'Y', biometrics: 'Y’,
   person_id: 4 },
   { _id: 15, adharno: "ABCD78345", image: 'Y', biometrics: 'Y’,
   person_id: 5 }]);
```

```
db.vehicle.insertMany([
    { _id: 5001, vtype: "Two Wheeler", reg: "KL42 7788", person_id: 1 },
    { _id: 5002, vtype: "Four Wheeler", reg: "KL42 8788",person_id: 1 },
    { _id: 5003, vtype: "Four Wheeler", reg: "KL43 12788", person_id: 3 },
    { _id: 5004, vtype: “Two Wheeler", reg: "KL43 23788", person_id: 3 },
   { _id: 5005, vtype: "Four Wheeler", reg: "KL69D1317",person_id: 2 },
   { _id: 5005, vtype: "Four Wheeler", reg: "KL69D1317",person_id: 2 },
          ])
```

```
db.address.insertMany([

   { _id: 1001, house_no: "YU-345", place: "Kochi", pin: 4582361,
     person_id: [1, 2] },
   { _id: 1002, house_no: "YU-145", place: "Kochi", pin: 4582361,
     person_id: [3, 4, 5] },
   { _id: 1003, house_no: "FG-435", place: "Kollam", pin: 8982361,
     person_id: [1] }
]);
```

Show Persons:- **db.person.find()**

Show Adhars:- **db.adhar.find()**

Show Vehicles:- **db.vehicle.find()**

Show Address:- **db.address.find()**

Show All:-

```
db.person.aggregate([
   {
      $lookup: {
         from: 'adhar',
         localField: '_id',
         foreignField: 'person_id',
         as: 'AdharDetails’
    }}, {    $lookup: {
         from: 'vehicle',
         localField: '_id',
         foreignField: 'person_id',
         as: 'VehicleDetails'       } },
```

```
{
    $lookup: {
        from: 'address',
        localField: '_id',
        foreignField: 'person_id',
        as: 'AddressDetails'
    }
}
])
```

Show Name of persons having 4 wheeler:-

```
db.vehicle.aggregate([
   {
      $match: {
         vtype: { $regex: /Four/i }
      }},{
      $lookup: {
         from: 'person',
         localField: 'person_id',
         foreignField: '_id',
         as: 'PersonalDetails'
      }},{
      $project: {
         'PersonalDetails.name': 1, _id: 0
      } }  ])
```

Show Address of persons having 4 wheeler:-

```
db.vehicle.aggregate([
   {
      $match: {
         vtype: { $regex: /Four/i }
      }},{
      $lookup: {
         from: 'person',
         localField: 'person_id',
         foreignField: '_id',
         as: 'PersonalDetails'
      }   },
```

```
{
    $lookup: {
        from: 'address',
        localField: 'person_id',
        foreignField: 'person_id',
        as: 'AddressDetails'
    } },{
    $project: {
        'PersonalDetails.name': 1, _id: 0,
        'AddressDetails': 1
    }},{
    $project: {
        'AddressDetails._id': 0, 'AddressDetails.person_id': 0
    }}])
```