ACS COLLEGE OF ENGINEERING

DEPARTMENT OF CSE-Data Science

MongoDB BDS456B

Manual

(Effective from the academic year 2023-2024)

Semester –IV

Prepared By:

Mrs. Indumathi K

Tutor

Department of CSE- Data Science

ACSCE, Bengaluru

List Of Experiments

1 a. Illustration of Where Clause, AND,OR operations in MongoDB.

b. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection)

2 a. Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.

b. Develop a MongoDB query to display the first 5 documents from the results obtained in a.

3 a. Execute query selectors (comparison selectors, logical selectors) and list out the results on any collection

b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

4 Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MongoDB.

5 Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)

6 Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project,$skip etc. students encourage to execute several queries to demonstrate various aggregation operators)

7 a. Find all listings with listing_url, name, address, host_picture_url in the listings And Reviews collection that have a host with a picture url

b. Using E-commerce collection write a query to display reviews summary.

8 a. Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes)

b. Demonstrate optimization of queries using indexes.

9 a. Develop a query to demonstrate Text search using catalog data collection for a given word

b. Develop queries to illustrate excluding documents with certain words and phrases

10 Develop an aggregation pipeline to illustrate Text search on Catalog data collection.

Experiment 1: a. Illustration of Where Clause, AND, OR operations in MongoDB.

C:\Users\Indu>mongosh

Current Mongosh Log ID: 662007b444c5203fe1117b7a

Connecting to:
mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.4

Using MongoDB: 7.0.8

Using Mongosh: 2.2.4

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

The server generated these startup warnings when booting

2024-04-16T10:13:50.721+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted

---

test> show dbs

admin 40.00 KiB

config 108.00 KiB

deptofCSE 31.43 MiB

local 40.00 KiB

products 80.00 KiB

test> use products

switched to db products

products> show collections

customer_details

product_details

```
products> db.product_details.find({name:'AC7 Phone'})
[
 {
  _id: 'ac7',
  name: 'AC7 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 320,
  rating: 4,
  warranty_years: 1,
  available: false
 }
]
products> db.product_details.find({$and:[{name:'AC7 Phone'},{brand:'ACME'}]})
[
 {
  _id: 'ac7',
  name: 'AC7 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 320,
  rating: 4,
  warranty_years: 1,
  available: false
 }
]
```

products> db.product_details.find({$or:[{name:'AC7 Phone'},{brand:'ACME'}]})

```
[
 {
  _id: 'ac3',
  name: 'AC3 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 200,
  rating: 3.8,
  warranty_years: 1,
  available: true
 },
 {
  _id: 'ac7',
  name: 'AC7 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 320,
  rating: 4,
  warranty_years: 1,
  available: false
 }
]
```

b. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection)

//Insert Query

products> db.product_details.insertOne({_id:'ac8',name:'AC8
Phone',brand:'ACME',type:'phone',price:2000,rating:4.0,warranty_years:2,available:true})

{ acknowledged: true, insertedId: 'ac8' }

products> db.product_details.find({_id:'ac8'})

```
[
 {
  _id: 'ac8',
  name: 'AC8 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 2000,
  rating: 4,
  warranty_years: 2,
  available: true
 }
]
```

//Update Query

products> db.product_details.updateOne({name:'AC8 Phone'},{$set:{price:4000}})

```
{
 acknowledged: true,
 insertedId: null,
 matchedCount: 1,
 modifiedCount: 1,
 upsertedCount: 0
}
```

products> db.product_details.find({_id:'ac8'})

```
[
 {
  _id: 'ac8',
  name: 'AC8 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 4000,
  rating: 4,
  warranty_years: 2,
  available: true
 }
]
```

//Deletion

products> db.product_details.deleteOne({_id:'ac8'})

{ acknowledged: true, deletedCount: 1 }

products> db.product_details.find({type:'tv'})

```
[
 {
  _id: ObjectId('507d95d5719dbef170f15c01'),
  name: 'Cable TV Basic Service Package',
  type: 'tv',
  monthly_price: 50,
  rating: 3.9,
  term_years: 2,
  cancel_penalty: 25,
  sales_tax: true,
```

```
   additional_tarriffs: [
    { kind: 'federal tarriff', amount: { percent_of_service:  0.06 } },
    { kind: 'misc tarriff', amount: 2.25 }
   ]
  }
]
```

Experiment 2:

a. Develop a MongoDB query to select certain fields and ignore some fields of the documents:

products> db.product_details.find({brand:'ACME'},{name:0,_id:0})

```
[
 {
  brand: 'ACME',
  type: 'phone',
  price: 200,
  rating: 3.8,
  warranty_years: 1,
  available: true
 },
 {
  brand: 'ACME',
  type: 'phone',
  price: 320,
  rating: 4,
  warranty_years: 1,
  available: false
 },
 {
  brand: 'ACME',
  type: 'phone',
  price: 2000,
  rating: 4,
```

```
  warranty_years: 2,
  available: true

 }
]
```

b. Develop a MongoDB query to display the first 5 documents:

```
products> db.product_details.find().limit(5)
[
 {
  _id: 'ac3',
  name: 'AC3 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 200,
  rating: 3.8,
  warranty_years: 1,
  available: true
 },
 {
  _id: 'ac7',
  name: 'AC7 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 320,
  rating: 4,
```

```
  warranty_years: 1,
  available: false
 },
 {
  _id: ObjectId('507d95d5719dbef170f15bf9'),
  name: 'AC3 Series Charger',
  type: [ 'accessory', 'charger' ],
  price: 19,
  rating: 2.8,
  warranty_years: 0.25,
  for: [ 'ac3', 'ac7', 'ac9' ]
 },
 {
  _id: ObjectId('507d95d5719dbef170f15bfa'),
  name: 'AC3 Case Green',
  type: [ 'accessory', 'case' ],
  color: 'green',
  price: 12,
  rating: 1,
  warranty_years: 0
 },
 {
  _id: ObjectId('507d95d5719dbef170f15bfb'),
  name: 'Phone Extended Warranty',
  type: 'warranty',
  price: 38,
```

```
    rating: 5,
    warranty_years: 2,
    for: [ 'ac3', 'ac7', 'ac9', 'qp7', 'qp8', 'qp9' ]
  }
]
```

Experiment 3:

a. Execute query selectors (comparison selectors, logical selectors):

products> db.product_details.find({price:{$gt:200}})

```
[
 {
  _id: 'ac7',
  name: 'AC7 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 320,
  rating: 4,
  warranty_years: 1,
  available: false
 },
 {
  _id: ObjectId('66290e738a958f3edf117b7b'),
  name: 'AC3 Phone',
  brand: 'ACME',
  type: 'phone',
  price: 2000,
  rating: 4,
  warranty_years: 2,
  available: true
 }
]
```

b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

products> use geoDatabase

switched to db geoDatabase

geoDatabase> db.Places.insertMany([

{ name: "Kengeri", location: { type: "Point", coordinates: [77.4827,12.8997] } },

{ name: "RR Nagar", location: { type: "Point", coordinates: [77.5206, 12.9149] } },

{ name: "JayaNagar", location: { type: "Point", coordinates: [77.5839, 12.9308] } },

{ name: "Kempegowda International Airport", location: { type: "Point", coordinates: [77.710136, 13.199379] } },

{ name: "vidhana soudha", location: { type: "Point", coordinates: [77.5908, 12.9796] } }

])

```
 acknowledged: true,
 insertedIds: {
   '0': ObjectId('6672aa3c7106ca43c3117b7b'),
   '1': ObjectId('6672aa3c7106ca43c3117b7c'),
   '2': ObjectId('6672aa3c7106ca43c3117b7d'),
   '3': ObjectId('6672aa3c7106ca43c3117b7e'),
'4': ObjectId('6672aa3c7106ca43c3117b7f')
 }
}
```

//create Index to apply geospatial queries

geoDatabase> db.Places.createIndex({ location: "2dsphere" })

location_2dsphere

// distance in meters

geoDatabase> db.Places.find({location: {$near: {$geometry: {type: "Point",coordinates: [77.4827,12.8997]},$maxDistance: 5000}} })

```
[
 {
  _id: ObjectId('6672aa3c7106ca43c3117b7b'),
  name: 'Kengeri',
  location: { type: 'Point', coordinates: [ 77.4827, 12.8997 ] }
 },
 {
  _id: ObjectId('6672aa3c7106ca43c3117b7c'),
  name: 'RR Nagar',
  location: { type: 'Point', coordinates: [ 77.5206, 12.9149 ] }
 }
]
```

//Bitwise selectors

bitdevice> db.Devices.insertMany([

{ name: "Device A", status: 5 },

{ name: "Device B", status: 3 },

{ name: "Device C", status: 12 },

{ name: "Device D", status: 10 },

{ name: "Device E", status: 7 }

])

```
{
 acknowledged: true,
 insertedIds: {
  '0': ObjectId('667bb0f17106ca43c3117b80'),
```

```
    '1': ObjectId('667bb0f17106ca43c3117b81'),
    '2': ObjectId('667bb0f17106ca43c3117b82'),
    '3': ObjectId('667bb0f17106ca43c3117b83'),
    '4': ObjectId('667bb0f17106ca43c3117b84')
  }
}
```

bitdevice> db.Devices.find({status:{$bitsAllSet:[0,2]}})

```
[
  {
    _id: ObjectId('667bb0f17106ca43c3117b80'),
    name: 'Device A',
    status: 5
  },
  {
    _id: ObjectId('667bb0f17106ca43c3117b84'),
    name: 'Device E',
    status: 7
  }
]
```

Experiment 4:

Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MongoDB.

products> db.product_details.find({},{"name.$":1})

```
[
 { _id: 'ac3', name: 'AC3 Phone' },
 { _id: 'ac7', name: 'AC7 Phone' },
 {
  _id: ObjectId('507d95d5719dbef170f15bf9'),
  name: 'AC3 Series Charger'
 },
 { _id: ObjectId('507d95d5719dbef170f15bfa'), name: 'AC3 Case Green' },
 {
  _id: ObjectId('507d95d5719dbef170f15bfb'),
  name: 'Phone Extended Warranty'
 },



 { _id: ObjectId('507d95d5719dbef170f15bfc'), name: 'AC3 Case Black' },
 { _id: ObjectId('507d95d5719dbef170f15bfd'), name: 'AC3 Case Red' },
 {
  _id: ObjectId('507d95d5719dbef170f15bfe'),
  name: 'Phone Service  Basic Plan'
 },
 {
  _id: ObjectId('507d95d5719dbef170f15bff'),
```

```
  name: 'Phone Service Core Plan'
 },
 {
  _id: ObjectId('507d95d5719dbef170f15c00'),
  name: 'Phone Service Family Plan'
 },
 {
  _id: ObjectId('507d95d5719dbef170f15c01'),
  name: 'Cable TV Basic Service Package'
 },
 { _id: ObjectId('66290e738a958f3edf117b7b'), name: 'AC3 Phone' }
]
```

products>
db.product_details.find({},{additional_tarriffs:{$elemMatch:{amount:2.25}}})

```
[
 { _id: 'ac3' },
 { _id: 'ac7' },
 { _id: ObjectId('507d95d5719dbef170f15bf9') },
 { _id: ObjectId('507d95d5719dbef170f15bfa') },
 { _id: ObjectId('507d95d5719dbef170f15bfb') },

 { _id: ObjectId('507d95d5719dbef170f15bfc') },
 { _id: ObjectId('507d95d5719dbef170f15bfd') },
 { _id: ObjectId('507d95d5719dbef170f15bfe') },
 { _id: ObjectId('507d95d5719dbef170f15bff') },
 { _id: ObjectId('507d95d5719dbef170f15c00') },
```

```
  {
    _id: ObjectId('507d95d5719dbef170f15c01'),
    additional_tarriffs: [ { kind: 'misc tarriff', amount: 2.25 } ]
  },
  { _id: ObjectId('66290e738a958f3edf117b7b') }
]
```

// $slice projection operator

```
products> db.product_details.find({},{name:1,brand:1,for:{$slice:1},_id:0})
[
  { name: 'AC3 Phone', brand: 'ACME' },
  { name: 'AC7 Phone', brand: 'ACME' },
  { name: 'AC3 Series Charger', for: [ 'ac3' ] },
  { name: 'AC3 Case Green' },
  { name: 'Phone Extended Warranty', for: [ 'ac3' ] },
  { name: 'AC3 Case Black', for: 'ac3' },
  { name: 'AC3 Case Red', for: 'ac3' },
  { name: 'Phone Service Basic Plan' },
  { name: 'Phone Service Core Plan' },
  { name: 'Phone Service Family Plan' },
  { name: 'Cable TV Basic Service Package' },
  { name: 'AC3 Phone', brand: 'ACME' }
]
```

5. Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators)

//find $avg

products> db.product_details.aggregate([{$group:{_id:0,avgprice:{$avg:'$price'}}}])

[{ _id: 0, avgprice: 326.6875 }]

//find $min

products> db.product_details.aggregate([{$group:{_id:0,minprice:{$min:'$price'}}}])

[{ _id: 0, minprice: 12 }]

//find $max

products> db.product_details.aggregate([{$group:{_id:0,maxprice:{$max:'$price'}}}])

[{ _id: 0, maxprice: 2000 }]

// $push

products> db.product_details.aggregate([{$group:{_id:0,allnames:{$push:'$name'}}}])

```
[
 {
  _id: 0,
  allnames: [
   'AC3 Phone',
   'AC7 Phone',
   'AC3 Series Charger',
   'AC3 Case Green',
   'Phone Extended Warranty',
   'AC3 Case Black',
   'AC3 Case Red',
   'Phone Service Basic Plan',
   'Phone Service Core Plan',
```

```
    'Phone Service  Family Plan',
    'Cable TV Basic Service Package',
    'AC3 Phone'
   ]
  }
]
```

6.Execute Aggregation Pipeline and its operations (pipeline must contain $match, $group, $sort, $project, $skip etc. students encourage to execute several queries to demonstrate various aggregation operators)

```
bitdevice> use resturant
```

switched to db resturant

```
resturant> db.restaurants.insertMany([
 {
  name: "Biryani House",
  cuisine: "Indian",
  location: "Jayanagar",
  reviews: [
   { user: "Aarav", rating: 5, comment: "Amazing biryani!" },
   { user: "Bhavana", rating: 4, comment: "Great place!" }
  ]
 },
 {
  name: "Burger Joint",
  cuisine: "American",
  location: "Koramangala",
  reviews: [
   { user: "Chirag", rating: 3, comment: "Average burger" },
   { user: "Devika", rating: 4, comment: "Good value" }
  ]

 },
 {
  name: "Pasta House",
  cuisine: "Italian",
```

```
    location: "Rajajinagar",
    reviews: [
      { user: "Esha", rating: 5, comment: "Delicious pasta!" },
      { user: "Farhan", rating: 4, comment: "Nice ambiance" }
    ]
  },
  {
    name: "Curry Palace",
    cuisine: "Indian",
    location: "Jayanagar",
    reviews: [
      { user: "Gaurav", rating: 4, comment: "Spicy and tasty!" },
      { user: "Harini", rating: 5, comment: "Best curry in town!" }
    ]
  },
  {
    name: "Taco Stand",
    cuisine: "Mexican",
    location: "Jayanagar",
    reviews: [
      { user: "Ishaan", rating: 5, comment: "Fantastic tacos!" },
      { user: "Jaya", rating: 4, comment: "Very authentic" }
    ]
  }
])
{
  acknowledged: true,
```

```
  insertedIds: {
    '0': ObjectId('667bb9f17106ca43c3117b85'),
    '1': ObjectId('667bb9f17106ca43c3117b86'),
    '2': ObjectId('667bb9f17106ca43c3117b87'),
    '3': ObjectId('667bb9f17106ca43c3117b88'),
    '4': ObjectId('667bb9f17106ca43c3117b89')
  }
}
```

db.restaurants.aggregate([{ $match: {location: "Jayanagar" } },

{ $unwind: "$reviews" }, { $group: {_id:"$name", averageRating: { $avg: "$reviews.rating" },totalReviews: { $sum: 1 } } },{$sort: {averageRating: -1 } },{$project: { _id: 0,restaurant: "$_id", averageRating: 1,totalReviews: 1} }, { $skip: 1 }])

```
[
  { averageRating: 4.5, totalReviews: 2, restaurant: 'Biryani House' },
  { averageRating: 4.5, totalReviews: 2, restaurant: 'Curry Palace' }
]
```

7 a. Find all listings with listing_url, name, address, host_picture_url in the listings And Reviews collection that have a host with a picture url

```
test> use vacation
```

switched to db vacation

```
vacation> db.listingsAndReviews.insertMany([
 {
  listing_url: "http://www.example.com/listing/123456",
  name: "Beautiful Apartment",
  address: {
   street: "123 Main Street",
   suburb: "Central",
   city: "Metropolis",
   country: "Wonderland"
  },
  host: {
   name: "Alice",
   picture_url: "http://www.example.com/images/host/host123.jpg"
  }
 },
 {
  listing_url: "http://www.example.com/listing/654321",
  name: "Cozy Cottage",
  address: {
   street: "456 Another St",
   suburb: "North",
   city: "Smallville",
```

```
    country: "Wonderland"
   },
   host: {
    name: "Bob",
    picture_url: ""
   }
  },
  {
   listing_url: "http://www.example.com/listing/789012",
   name: "Modern Condo",
   address: {
    street: "789 Side Road",
    suburb: "East",
    city: "Gotham",
    country: "Wonderland"
   },
   host: {
    name: "Charlie",
    picture_url: "http://www.example.com/images/host/host789.jpg"
   }
  }
])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('668be647efc7cdb5d3117b7b'),
```

```
    '1': ObjectId('668be647efc7cdb5d3117b7c'),
    '2': ObjectId('668be647efc7cdb5d3117b7d')
  }
}
vacation> db.listingsAndReviews.find(
 {
  "host.picture_url": { $exists: true, $ne: "" }
 },
 {
  listing_url: 1,
  name: 1,
  address: 1,
  "host.picture_url": 1
 }
)
[
 {
  _id: ObjectId('668be647efc7cdb5d3117b7b'),
  listing_url: 'http://www.example.com/listing/123456',
  name: 'Beautiful Apartment',
  address: {
   street: '123 Main Street',
   suburb: 'Central',
   city: 'Metropolis',
   country: 'Wonderland'
  },
```

```
  host: { picture_url: 'http://www.example.com/images/host/host123.jpg' }
 },
 {
  _id: ObjectId('668be647efc7cdb5d3117b7d'),
  listing_url: 'http://www.example.com/listing/789012',
  name: 'Modern Condo',
  address: {
   street: '789 Side Road',
   suburb: 'East',
   city: 'Gotham',
   country: 'Wonderland'
  },
  host: { picture_url: 'http://www.example.com/images/host/host789.jpg' }
 }
]
```

b. Using E-commerce collection write a query to display reviews summary.

```
products> use ecommerce
switched to db ecommerce
ecommerce> db.products.insertMany([
 {
  product_id: 1,
  name: "Laptop",
  category: "Electronics",
  price: 1200,
  reviews: [
   { user: "Alice", rating: 5, comment: "Excellent!" },
```

```
    { user: "Bob", rating: 4, comment: "Very good" },
    { user: "Charlie", rating: 3, comment: "Average" }
  ]
 },
 {
  product_id: 2,
  name: "Smartphone",
  category: "Electronics",
  price: 800,
  reviews: [
    { user: "Dave", rating: 4, comment: "Good phone" },
    { user: "Eve", rating: 2, comment: "Not satisfied" },
    { user: "Frank", rating: 5, comment: "Amazing!" }
  ]
 },
 {
  product_id: 3,
  name: "Headphones",
  category: "Accessories",
  price: 150,
  reviews: [
    { user: "Grace", rating: 5, comment: "Great sound" },
    { user: "Heidi", rating: 3, comment: "Okay" }
  ]
 }
])
```

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('668be8f2efc7cdb5d3117b7e'),
    '1': ObjectId('668be8f2efc7cdb5d3117b7f'),
    '2': ObjectId('668be8f2efc7cdb5d3117b80')
  }
}
ecommerce> db.products.aggregate([
  {
    $unwind: "$reviews"
  },
  {
    $group: {
      _id: "$name",
      totalReviews: { $sum: 1 },
      averageRating: { $avg: "$reviews.rating" },
      comments: { $push: "$reviews.comment" }
    }
  },
  {
    $project: {
      _id: 0,
      product: "$_id",
      totalReviews: 1,
      averageRating: 1,
```

```
    comments: 1
   }
  }
])
```

```
[
 {
  totalReviews: 3,
  averageRating: 4,
  comments: [ 'Excellent!', 'Very good', 'Average' ],
  product: 'Laptop'
 },
 {
  totalReviews: 3,
  averageRating: 3.6666666666666665,
  comments: [ 'Good phone', 'Not satisfied', 'Amazing!' ],
  product: 'Smartphone'
 },
 {
  totalReviews: 2,
  averageRating: 4,
  comments: [ 'Great sound', 'Okay' ],
  product: 'Headphones'
 }
]
```

8. a. Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes)

```
test> use resturant
switched to db resturant
resturant> db.restaurants.createIndex({ "name": 1 }, { unique: true })
name_1
resturant> db.restaurants.createIndex({ location: 1 }, { sparse:true })
location_1
resturant> db.restaurants.createIndex({ name: 1, location: 1 })
name_1_location_1
resturant> db.restaurants.createIndex({ reviews: 1 })
reviews_1
resturant> db.restaurants.getIndexes()
[
 { v: 2, key: { _id: 1 }, name: '_id_' },
 { v: 2, key: { name: 1 }, name: 'name_1', unique: true },
 { v: 2, key: { location: 1 }, name: 'location_1', sparse: true },
 { v: 2, key: { name: 1, location: 1 }, name: 'name_1_location_1' },
 { v: 2, key: { reviews: 1 }, name: 'reviews_1' }
]
```

b. Demonstrate optimization of queries using indexes.

```
resturant> db.restaurants.find({ location: "Jayanagar" }).hint({ location: 1 })
[
 {
  _id: ObjectId('667bb9f17106ca43c3117b85'),
```

```
  name: 'Biryani House',
  cuisine: 'Indian',
  location: 'Jayanagar',
  reviews: [
   { user: 'Aarav', rating: 5, comment: 'Amazing biryani!' },
   { user: 'Bhavana', rating: 4, comment: 'Great place!' }
  ]
 },
 {
  _id: ObjectId('667bb9f17106ca43c3117b88'),
  name: 'Curry Palace',
  cuisine: 'Indian',
  location: 'Jayanagar',
  reviews: [
   { user: 'Gaurav', rating: 4, comment: 'Spicy and tasty!' },
   { user: 'Harini', rating: 5, comment: 'Best curry in town!' }
  ]
 },
 {
  _id: ObjectId('667bb9f17106ca43c3117b89'),
  name: 'Taco Stand',
  cuisine: 'Mexican',
  location: 'Jayanagar',
  reviews: [
   { user: 'Ishaan', rating: 5, comment: 'Fantastic tacos!' },
   { user: 'Jaya', rating: 4, comment: 'Very authentic' }
```

```
   ]
  }
]
```

resturant> db.restaurants.find({name:"Pasta House"}).explain()

```
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'resturant.restaurants',
    indexFilterSet:  false,
    parsedQuery: { name: { '$eq': 'Pasta House' } },
    queryHash: 'A2F868FD',
    planCacheKey: 'DA04EF84',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { name: 1 },
        indexName: 'name_1',
        isMultiKey: false,
        multiKeyPaths: { name: [] },
        isUnique: true,
        isSparse: false,
```

```
    isPartial: false,
    indexVersion: 2,
    direction: 'forward',
    indexBounds: { name: [ '["Pasta House", "Pasta House"]' ] }
   }
  },
  rejectedPlans: [
   {
    stage: 'FETCH',
    inputStage: {
     stage: 'IXSCAN',
     keyPattern: { name: 1, location: 1 },
     indexName: 'name_1_location_1',
     isMultiKey: false,
     multiKeyPaths: { name: [], location: [] },
     isUnique: false,
     isSparse: false,
     isPartial: false,
     indexVersion: 2,
     direction: 'forward',
     indexBounds: {
      name: [ '["Pasta House", "Pasta House"]' ],
      location: [ '[MinKey, MaxKey]' ]
     }
    }
   }
```

```
  ]
 },
 command: {
  find: 'restaurants',
  filter: { name: 'Pasta House' },
  '$db': 'resturant'
 },
 serverInfo: {
  host: 'DESKTOP-1HKM3JN',
  port: 27017,
  version: '7.0.8',
  gitVersion: 'c5d33e55ba38d98e2f48765ec4e55338d67a4a64'
 },
 serverParameters: {
  internalQueryFacetBufferSizeBytes: 104857600,
  internalQueryFacetMaxOutputDocSizeBytes: 104857600,
  internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
  internalDocumentSourceGroupMaxMemoryBytes: 104857600,
  internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
  internalQueryProhibitBlockingMergeOnMongoS: 0,
  internalQueryMaxAddToSetBytes: 104857600,
  internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
  internalQueryFrameworkControl: 'trySbeRestricted'
 },
 ok: 1
}
```

9.a. Develop a query to demonstrate Text search using catalog data collection for a given word

```
test > use movies
```

switched to db movies

```
movies> db.movies.insertMany([
 {
  title: "Bang Bang",
  genres: ["Romance", "Action"],
  runtime: 105,
  rated: "R",
  year: 2018,
  directors: ["Siddhartha Anand"],
  cast: ["Hritik Roshan", "Katrina Kaif"],
  type: "movie"
  },
 {
  title: "Fanaa",
  genres: ["Drama", "Action", "Romance"],
  runtime: 203,
  rated: "R",
  year: 2006,
  directors: ["Kunal Kohli"],
  cast: ["Amir Khan", "Kajol", "Rishi Kapoor"],
  type: "movie"
  },
 {
```

title: "Robin Hood",

genres: ["Action", "Romance"],

runtime: 143,

rated: "R",

year: 1922,

directors: ["Allan Dwan"],

cast: ["Sam De Grasse", "Wallace Beery", "Enid Bennett"],

type: "movie"

}

])

{

acknowledged: true,

insertedIds: {

'0': ObjectId('668c16a0efc7cdb5d3117b81'),

'1': ObjectId('668c16a0efc7cdb5d3117b82'),

'2': ObjectId('668c16a0efc7cdb5d3117b83')

}

}

movies> db.movies.createIndex({

title: "text",

directors: "text",

cast: "text",

genres: "text"

})

title_text_directors_text_cast_text_genres_text

movies> db.movies.find({ $text: { $search: "Hritik Fanaa" } })

```
[
 {
  _id: ObjectId('668c16a0efc7cdb5d3117b82'),
  title: 'Fanaa',
  genres: [ 'Drama', 'Action', 'Romance' ],
  runtime: 203,
  rated: 'R',
  year: 2006,
  directors: [ 'Kunal Kohli' ],
  cast: [ 'Amir Khan', 'Kajol', 'Rishi Kapoor' ],
  type: 'movie'
 },
 {
  _id: ObjectId('668c16a0efc7cdb5d3117b81'),
  title: 'Bang Bang',
  genres: [ 'Romance', 'Action' ],
  runtime: 105,
  rated: 'R',
  year: 2018,
  directors: [ 'Siddhartha Anand' ],
  cast: [ 'Hritik Roshan', 'Katrina Kaif' ],
  type: 'movie'
 }
]
```

//to find exact phrase

movies> db.movies.find({ $text: { $search: "\"Hritik Roshan\"" } })

```
[
 {
  _id: ObjectId('668c16a0efc7cdb5d3117b81'),
  title: 'Bang Bang',
  genres: [ 'Romance', 'Action' ],
  runtime: 105,
  rated: 'R',
  year: 2018,
  directors: [ 'Siddhartha Anand' ],
  cast: [ 'Hritik Roshan', 'Katrina Kaif' ],
  type: 'movie'
 }
]
```

b. Develop queries to illustrate excluding documents with certain words and phrases

```
movies> db.movies.find( { $text: { $search: "Action -Hritik" } } )
[
 {
  _id: ObjectId('668c16a0efc7cdb5d3117b83'),
  title: 'Robin Hood',
  genres: [ 'Action', 'Romance' ],
  runtime: 143,
  rated: 'R',
  year: 1922,
  directors: [ 'Allan Dwan' ],
  cast: [ 'Sam De Grasse', 'Wallace Beery', 'Enid Bennett' ],
  type: 'movie'
```

```
  },
  {
    _id: ObjectId('668c16a0efc7cdb5d3117b82'),
    title: 'Fanaa',
    genres: [ 'Drama', 'Action', 'Romance' ],
    runtime: 203,
    rated: 'R',
    year: 2006,
    directors: [ 'Kunal Kohli' ],
    cast: [ 'Amir Khan', 'Kajol', 'Rishi Kapoor' ],
    type: 'movie'
  }
]
```

10. Develop an aggregation pipeline to illustrate Text search on Catalog data collection.

```
test> use movies
```

switched to db movies

```
movies> db.movies.aggregate(
  [
   { $match: { $text: { $search: "Action" } } },
   {$sort: {title:1}},
   { $project: { title: 1, _id: 0} }
  ]
)
```

[{ title: 'Bang Bang' }, { title: 'Fanaa' }, { title: 'Robin Hood' }]