Sl No **PROGRAMS**

1

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

```
// Find documents where age is greater than 25 and gender is "male"

db.collection.find({ $and: [{ age: { $gt: 25 } }, { gender: "male" }] })

// Find documents where age is greater than 30 or gender is "female"

db.collection.find({ $or: [{ age: { $gt: 30 } }, { gender: "female" }] })
```

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

db.createcollection("studentdetails")

To insert one document:

```
db.studentdetails.insertOne({title: "Post Title 1", body: "Body of post.", category: "News", likes: 1,tags: ["news", "events"],  date: Date()})
```

To insert multiple documents:

```
db.posts.insertMany([
 {
  title: "Post Title 2",
  body: "Body of post.",
  category: "Event",
  likes: 2,
  tags: ["news", "events"],
  date: Date()
 },
 {
  title: "Post Title 3",
  body: "Body of post.",
  category: "Technology",
  likes: 3,
  tags: ["news", "events"],
  date: Date()
 },
 {
  title: "Post Title 4",
  body: "Body of post.",
  category: "Event",
  likes: 4,
  tags: ["news", "events"],
  date: Date()
 }
])
```

Update Document

To update an existing document we can use the updateOne() or updateMany() methods.
The first parameter is a query object to define which document or documents should be updated.
The second parameter is an object defining the updated data.

updateOne()

The updateOne() method will update the first document that is found matching the provided query.
Let's see what the "like" count for the post with the title of "Post Title 1":

```
db.posts.find( { title: "Post Title 1" } )
db.posts.updateOne( { title: "Post Title 1" }, { $set: { likes: 2 } } )
```

The updateMany() method will update all documents that match the provided query.

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

Delete One Document

To delete a single document that matches a specified condition, use deleteOne.

```
db.collectionName.deleteOne({ key: value })
```

```
db.student.deleteOne({Anna:101})
```

Delete Multiple Documents

To delete multiple documents that match a specified condition, use deleteMany.

```
db.collectionName.deleteOne({ key: value })
```

```
db.student.deleteMany({ age: { $gt: 30 } })
```

Projection

Both find methods accept a second parameter called projection.
This parameter is an object that describes which fields to include in the results.

```
db.posts.find({}, {title: 1, date: 1})
```

2	a. Develop a MongoDB query to select certain fields and ignore some fields of the documents from any collection.
	
	// Select certain fields and ignore some fields
	db.collection.find({}, { name: 1, age: 1, _id: 0 })
	b. Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find]
	
	// Display the first 5 documents
	db.collection.find().limit(5)

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

A.1	**comparison selectors**
	
	// Comparison selectors
	
	db.collection.find({ age: { $gt: 30 } })
	
	// Logical selectors
	
	db.collection.find({ $and: [{ age: { $gt: 25 } }, { gender: "male" }] })
	
	Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection

B.	**Geospatial selectors:** 1. **Create a Database and Collection**:

B.1	use geospatialDB
	db.createCollection("locations")
	
	2. **Create a Geospatial Index:**
	
	db.locations.createIndex({ location: "2dsphere" })
	3. Insert Geospatial Data**:**
	
	db.locations.insertMany([
	{
	name: "Location A",
	location: {
	type: "Point",
	coordinates: [-73.97, 40.77]
	}

B.2

```
 }
 {
   name: "Location B",
   location: {
     type: "Point",
     coordinates: [ -73.88, 40.78 ]
   }
 }
])
```

Find Nearest Locations:

```
db.locations.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [ -73.95, 40.75 ] // Replace with your coordinates
      },
      $maxDistance: 5000 // Optional: max distance in meters
    }}})
```

Bitwise selectors:

```
db.collection.insertMany([
  { "_id": 1, "field": 5 }, // binary: 0101
  { "_id": 2, "field": 9 }, // binary: 1001
  { "_id": 3, "field": 12 } // binary: 1100
])
```

1. db.collection.find({ field: { $bitsAllSet: [0, 2] } })
2. db.collection.find({ field: { $bitsAnySet: [1] } })
3. db.collection.find({ field: { $bitsAllClear: [0, 2] } })
4. db.collection.find({ field: { $bitsAnyClear: [0, 2] } })

4

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

```
db.marklist.insertMany([
{ "_id": 1, "semester": 1, "grades": [70, 87, 90] },
{ "_id": 2, "semester": 1, "grades": [90, 88, 92] },
 { "_id": 3, "semester": 1, "grades": [85, 100, 90] },
 { "_id": 4, "semester": 2, "grades": [79, 85, 80] },
{ "_id": 5, "semester": 2, "grades": [88, 88, 92] },
 { "_id": 6, "semester": 2, "grades": [95, 90, 96] }])
```

1. db.marklist.find({ semester: 1, grades: { $gte: 85 } }, { "grades.$": 1 })
2. db.marklist.find({"grades": {"$elemMatch": {"$gte": 95} }})

3. db.marklist.find({},{"grades": {"$slice": 1 } })

5

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

```
db.sales.insertMany([
{ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" :
ISODate("2014-01-01T08:00:00Z") }
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "date" :
ISODate("2014-02-03T09:00:00Z") }
{ "_id" : 3, "item" : "xyz", "price" : 5, "quantity" : 5, "date" :
ISODate("2014-02-03T09:05:00Z") }
{ "_id" : 4, "item" : "abc", "price" : 10, "quantity" : 10, "date" :
ISODate("2014-02-15T08:00:00Z") }
{ "_id" : 5, "item" : "xyz", "price" : 5, "quantity" : 10, "date" :
ISODate("2014-02-15T09:05:00Z") }
{ "_id" : 6, "item" : "xyz", "price" : 5, "quantity" : 5, "date" :
ISODate("2014-02-15T12:05:10Z") }
{ "_id" : 7, "item" : "xyz", "price" : 5, "quantity" : 10, "date" :
ISODate("2014-02-15T14:12:12Z") }])
```

1.$avg:

```
db.sales.aggregate([
  {
    $group : {
       _id : null,
       averageQuantity: { $avg: "$quantity" },
       count: { $sum: 1 }
    }
  }
 ])
```

2.$min.

```
db.sales.aggregate([{ $group: { _id: null, minqty: { $min: "$quantity" } } }])
```

3.$max:

```
db.sales.aggregate([{ $group: { _id: null, maxqty: { $max: "$quantity" } } }])
```

4.$push:

```
db.sales.aggregate(
  [
    {
      $group:
        {
          _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
          itemsSold: { $push:  { item: "$item", quantity: "$quantity" } }
        }
    }
  ]
```

5.)$addToSet

```
db.sales.aggregate(
  [
    {
      $group:
        {
          _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
          itemsSold: { $addToSet: "$item" }
        }
    }
  ]
)
```

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)

```
db.orders.insertMany( [
   { _id: 0, name: "Pepperoni", size: "small", price: 19,  quantity: 10, date: ISODate(
"2021-03-13T08:14:30Z" ) },
   { _id: 1, name: "Pepperoni", size: "medium", price: 20, quantity: 20, date : ISODate(
"2021-03-13T09:13:24Z" ) },
   { _id: 2, name: "Pepperoni", size: "large", price: 21, quantity: 30, date : ISODate(
"2021-03-17T09:22:12Z" ) },
   { _id: 3, name: "Cheese", size: "small", price: 12, quantity: 15, date : ISODate(
"2021-03-13T11:21:39.736Z" ) },
   { _id: 4, name: "Cheese", size: "medium", price: 13, quantity:50, date : ISODate(
"2022-01-12T21:23:13.331Z" ) },
   { _id: 5, name: "Cheese", size: "large", price: 14, quantity: 10, date : ISODate(
"2022-01-12T05:08:13Z" ) },
   { _id: 6, name: "Vegan", size: "small", price: 17, quantity: 10, date : ISODate(
 "2021-01-13T05:08:13Z" ) },
   { _id: 7, name: "Vegan", size: "medium", price: 18, quantity: 10, date : ISODate(
 "2021-01-13T05:10:13Z" ) }] )
```

$match & $group

```
db.orders.aggregate( [
   // Stage 1: Filter pizza order documents by pizza size
   {
      $match: { size: "medium" }
   },
   // Stage 2: Group remaining documents by pizza name and calculate total quantity
   {
      $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
   }
] )
```

$sort:

```
db.restaurants.insertMany( [
   { "_id" : 1, "name" : "Central Park Cafe", "borough" : "Manhattan"},
   { "_id" : 2, "name" : "Rock A Feller Bar and Grill", "borough" : "Queens"},
   { "_id" : 3, "name" : "Empire State Pub", "borough" : "Brooklyn"},
   { "_id" : 4, "name" : "Stan's Pizzaria", "borough" : "Manhattan"},
   { "_id" : 5, "name" : "Jane's Deli", "borough" : "Brooklyn"},
] )

db.restaurants.aggregate(    [ { $sort : { borough : 1 } }    ] )
```

$project:

```
db.restaurants.aggregate([{$project:{name:1}}])
```

$skip:

```
db.restaurants.aggregate([{ $skip : 2 }]);
```

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

Inserting values:

```
db.listings_and_reviews.insertMany([
    {
        listing_url: "https://example.com/listing/2",
        name: "Cozy Cabin",
        address: "456 Forest Rd, City, Country",
        host: {
            picture_url: "https://example.com/host/2.jpg"
      }
    },

{
        listing_url: "https://example.com/listing/3",
        name: "Modern Loft",
        address: "789 Skyline Dr, City, Country",
        host: {
            picture_url: "https://example.com/host/3.jpg"
        }
    }
])
```

QUERY:

```
db.listings_and_reviews.find(
    { "host.picture_url": { $exists: true, $ne: null, $ne: "" } },
    {
        "listing_url": 1,
        "name": 1,
        "address": 1,
        "host.picture_url": 1
    }
)
```

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

Insert the values:

```
db.ecommerce.insertMany([
      {
        product_name: "Product A",
        reviews: [
            {
                rating: 3,
                comment: "Average product.",
                date: new Date("2023-01-15")
            },
            {
                rating: 2,
                comment: "Not satisfied with the quality.",
                date: new Date("2023-07-05")
            }
        ]
    },

    {
        product_name: "Product B",
        reviews: [
            {
                rating: 3,
                comment: "Average product.",
                date: new Date("2023-02-20")
            },
            {
                rating: 2,
                comment: "Not satisfied with the quality.",
                date: new Date("2023-04-05")
            }
        ]
    },
    {
        product_name: "Product C",
        reviews: [
            {
                rating: 5,
                comment: "Highly recommend!",
                date: new Date("2023-05-18")
            }
        ]
    }
])
```

Query:

```
db.ecommerce.aggregate([
    { $unwind: "$reviews" }, // Deconstruct the reviews array
    {
        $group: {
            _id: null,
            totalReviews: { $sum: 1 }, // Count total number of reviews
            averageRating: { $avg: "$reviews.rating" }, // Calculate average rating
            latestReview: { $max: "$reviews.date" }, // Find the most recent review date
            oldestReview: { $min: "$reviews.date" } // Find the oldest review date
        }
    },
    {
        $project: {
            _id: 0, // Exclude the _id field from the results
            totalReviews: 1,
            averageRating: 1,
            latestReview: 1,
            oldestReview: 1
        }
    }
])
```

8

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

1. Unique Index

A unique index ensures that the indexed field does not contain duplicate values.

synatax:

```
db.exampleCollection.createIndex({ "fieldName": 1 }, { unique: true })
```

Query:

```
db.exampleCollection.insertMany([
 { _id: 1, username: "alice", age: 25 },
 { _id: 2, username: "bob", age: 30 }
]);

db.exampleCollection.createIndex({ "username": 1 }, { unique: true });
// This ensures that the "username" field contains unique values.
```

2. Sparse Index

A sparse index only indexes the documents that contain the indexed field, skipping documents that do not have the field.

synatax:

```
db.exampleCollection.createIndex({ "fieldName": 1 }, { sparse: true })
```

Query:

```
db.exampleCollection.insertMany([
   { _id: 3, username: "carol" },
   { _id: 4, username: "dave", email: "dave@example.com" }
]);

db.exampleCollection.createIndex({ "email": 1 }, { sparse: true });
// This indexes only documents where "email" exists.
```

3. Compound Index

A compound index indexes multiple fields within a collection. The order of the fields in the index is significant.

synatax:

```
db.exampleCollection.createIndex({ "field1": 1, "field2": -1 })
```

Query:

```
db.exampleCollection.insertMany([
  { _id: 5, firstName: "eve", lastName: "johnson" },
  { _id: 6, firstName: "frank", lastName: "smith" }
]);

db.exampleCollection.createIndex({ "firstName": 1, "lastName": -1 });
// This creates an index on both "firstName" and "lastName" fields.
```

4. Multikey Index

A multikey index is created on an array field, allowing MongoDB to index the individual elements within the array.

synatax:

```
db.exampleCollection.createIndex({ "arrayField": 1 })
```

Query:

```
db.exampleCollection.insertMany([
  { _id: 7, tags: ["mongodb", "database", "nosql"] },
  { _id: 8, tags: ["indexing", "performance"] }
 ]);

 db.exampleCollection.createIndex({ "tags": 1 });
// This creates an index on the elements of the "tags" array.
```

b. Demonstrate optimization of queries using indexes.

Unique Index:

db.optimization.createIndex({ "name": 1 }, { unique: true });

db.optimization.find({ age: { $gt: 30 } }).explain("executionStats");

Compound Index:

```
db.exampleCollection.createIndex({ name: 1, age: 1 });

db.exampleCollection.find({ name: "Alice", age: 25 }).explain("executionStats");
```

Multiple Index:

```
db.exampleCollection.createIndex({ tags: 1 });

db.exampleCollection.find({ tags: "mongodb" }).explain("executionStats");
```

9

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

```
use catalogDB

db.catalog.insertMany
  ([ { title: "Data Science for Beginners", description: "An introductory book on data
science." },
{ title: "Advanced Machine Learning", description: "A deep dive into machine learning
algorithms." },
{ title: "MongoDB in Action", description: "A comprehensive guide to MongoDB." }, {
title: "Data Analysis with Python", description: "Learn data analysis techniques using
Python." },
 { title: "Introduction to Big Data", description: "Basics of big data technologies." } ])


# Create text index
db.catalog.createIndex({ title: "text", description: "text" })


# Perform text search db.catalog.find({ $text: { $search: "data" } })
```

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

Method:1

```
db.catalog.aggregate([
   { $match: { $text: { $search: "data" } } },
   { $match: { title: { $not: /Python/ }, description: { $not: /Python/ } } }
])
```

Method:2

```
db.catalog.aggregate([
   { $match: { $text: { $search: "data" } } },
   { $match: {
      $and: [
         { title: { $not: /Python/ } },
         { description: { $not: /Python/ } }
      ]
   }}
])
```

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

```
db.catalog.aggregate([
   {
      $match: {
         $text: { $search: "data" }
      }
   },
   {
      $match: {
         $and: [
            { title: { $not: /Python/ } },
            { description: { $not: /Python/ } }
         ]
      }
   },
   {
      $project: {
         title: 1,
         description: 1,
         score: { $meta: "textScore" }
      }
   },
   {
      $sort: {
         score: { $meta: "textScore" }
      }
   }
])
```