MongoDB

LAB MANUAL

IV Semester AI&DS

Designed By,

1. **Prof. Vaibhav Chavan**
2. **Prof. Sagar Birje**

Institute Vision

To become premier institute committed to **academic excellence** and **global competence** for the **holistic development** of students.

Key words: academic excellence, global competence, holistic development.

Institute Mission

M1: Develop competent human resources, adopt outcome based education (OBE) and implement cognitive assessment of students.

M2: Inculcate the traits of global competencies amongst the students.

M3: Nurture and train our students to have domain knowledge, develop the qualities of global professionals and to have social consciousness for holistic development.

Department Vision

To deliver a **quality and responsive** education in the field of artificial intelligence and data science emphasizing **professional skills** to face **global challenges** in the evolving IT paradigm.

Key words: quality and responsive, professional skills, global challenges.

Department Mission

M1: Leverage multiple pedagogical approaches to impart knowledge on the current and emerging AI technologies.

M2: Develop an inclusive and holistic ambiance that bolsters problem solving, cognitive abilities and critical thinking.

M3: Enable students to develop trust worthiness, team spirit, understanding law-of-the-land, social behavior to be a global stake holder.

Program Specific Outcomes (PSOs):

PSO1: To apply core knowledge of Artificial Intelligence, Machine Learning, Deep Learning, Data Science, Big Data Analytics and Statistical Learning to develop effective solutions for real-world problems.

PSO2: To demonstrate proficiency in specialized and emerging technologies such as Natural Language Processing, Cloud Computing, Robotic Process Automation, Storage Area Networks and the Internet of Things to meet the stringent and diverse professional challenges.

PSO3: To imbibe managerial skills, social responsibility, ethical and moral values through courses in Management and Entrepreneurship, Software Engineering Principles, Universal Human Values and Ability Enhancement Programs to meet the industry and societal expectations.

Program Educational Objectives (PEOs)

PEO 1: Build a strong foundation in mathematics, core programming, artificial intelligence, machine learning, and data science to enable graduates to analyze, design, and implement intelligent systems for solving complex real-world problems.

PEO 2: Foster creativity, cognitive and research skills to analyze the requirements and technical specifications of software to articulate novel engineering solutions for an efficient product design.

PEO 3: Prepare graduates for dynamic career opportunities in AI and Data Science by equipping them with interdisciplinary knowledge, adaptability, and practical exposure to tools and techniques required for industry and research.

PEO 4: Instill a strong sense of ethics, professional responsibility, and human values, empowering graduates to contribute positively to society and lead with integrity in their professional domains.

PEO 5: Encourage graduates to pursue higher education, certification program, entrepreneurial ventures, etc. by nurturing a mindset of continuous learning and awareness of global trends and challenges.

INDEX

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

```
//To create the new database as well as switch the database if not existing
use ProgrammingBooks

//To create the collection inside the database
db.createCollection("BookDetails")

//To insert the single value or document
db.BookDetails.insertOne({
  _id: 1,
  title: "Clean Code",
  author: "Robert C. Martin",
  category: "Software Development",
  year: 2008
})

//To insert the multiple values or documents
db.BookDetails.insertMany([
{_id: 1, title: "Clean Code", author: "Robert C. Martin", category: "Software Development", year: 2008 },
{_id: 2,  title: "JavaScript: The Good Parts", author: "Douglas Crockford", category: "JavaScript", year: 2008 },
{_id: 3,  title: "Design Patterns", author: "Erich Gamma", category: "Software Design", year: 1994 },
{_id: 4,  title: "Introduction to Algorithms", author: "Thomas H. Cormen", category: "Algorithms", year: 2009 },
{_id: 5,  title: "Python Crash Course", author: "Eric Matthes", category: "Python", year: 2015 }
]);
```

```
//Condition statement for where operator
db.BookDetails.find({ year: 2008 }).pretty()
```

OUTPUT

```
[
  {
    _id: 1,
    title: 'Clean Code',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]
```

```
//Condition statement for $and operator
db.BookDetails.find({
$and: [
  { category: "Software Development" },
  { year: 2008 }
 ]
 }).pretty()
```

OUTPUT

```
[
  {
    _id: 1,
    title: 'Clean Code',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  }
]
```

Using the $or Operator:

```
////Condition statement for $or operator
db.BookDetails.find({
$or: [
    { category: "JavaScript" },
    { year: 2015 }
  ]
  }).pretty()
```

OUTPUT

```
[
  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
```

```
  },
  {
    _id: 5,
    title: 'Python Crash Course',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

1b. Execute the Commands of MongoDB and operations in MongoDB : Insert, Query, Update, Delete and Projection. (Note: use any collection)

```
//To create the new database as well as switch the database if not existing
use Books

//To create the collection inside the database
db.createCollection("BookDetails")

//To insert the single value or document
db.BookDetails.insertOne({
  _id: 1,
  title: "The Pragmatic Programmer: Your Journey to Mastery",
  author: "David Thomas, Andrew Hunt",
  category: "Software Development",
  year: 1999
})

//To insert the multiple values or documents
db.BookDetails.insertMany([
  {
```

```
  _id: 1,
  title: "Clean Code: A Handbook of Agile Software Craftsmanship",
  author: "Robert C. Martin",
  category: "Software Development",
  year: 2008
},

{
  _id: 2,
  title: "JavaScript: The Good Parts",
  author: "Douglas Crockford",
  category: "JavaScript",
  year: 2008
},

{
  _id: 3,
  title: "Design Patterns: Elements of Reusable Object-Oriented Software",
  author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
  category: "Software Design",
  year: 1994
},

{
  _id: 4,
  title: "Introduction to Algorithms",
  author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein",
  category: "Algorithms",
  year: 1990
},
```

```
  {
    _id: 5,
    title: "Python Crash Course: A Hands-On, Project-Based Introduction to Programming",
    author: "Eric Matthes",
    category: "Python",
    year: 2015
  }
])
```

1. Find All Documents command…
   db.BookDetails.find().pretty()

```
[
  {
    _id: 1,
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: 3,
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Software Design',
    year: 1994
  },
  {
    _id: 4,
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford
Stein',
    category: 'Algorithms',
```

```
    year: 1990
  },
  {
    _id: 5,
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

2. Find Documents Matching a Condition:

```
db.BookDetails.find({ year: { $gt: 2000 } }).pretty()

[
  {
    _id: 1,
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: 5,
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
```

Update Operations:

```
//To insert the single value or document
db.BookDetails.updateOne(
  { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
```

```
  { $set: { author: "vtucode" } }
)

//To see the updated result
db.BookDetails.find({ year: { $eq: 2008 } }).pretty()

[
  {
    _id: 1,
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'vtucode',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  }
]

//To insert the multiple values or documents
db.BookDetails.updateMany(
  { year: { $lt: 2010 } },
  { $set: { category: "vtucode website" } }
)

//To see the updated result
db.BookDetails.find({ year: { $lt: 2010 } }).pretty()

[
  {
    _id: 1,
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'vtucode',
    category: 'vtucode website',
    year: 2008
  },
  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'vtucode website',
    year: 2008
```

```
  },
  {
    _id: 3,
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'vtucode website',
    year: 1994
  },
  {
    _id: 4,
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford
Stein',
    category: 'vtucode website',
    year: 1990
  }
]
```

Delete Operations:

```
//To delete the single value or document
db.BookDetails.deleteOne({ _id: 2 })

//To verify the deleted document
db.BookDetails.find({ _id: 2 }).pretty()

//To delete the multiple values or documents
db.BookDetails.deleteMany({ year: { $lt: 1995 } })

//To verify the deleted document
db.BookDetails.find().pretty()
```

```
[
  {
    _id: 1,
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'vtucode',
    category: 'vtucode website',
    year: 2008
  },
  {
    _id: 5,
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
```

```
  }
]

//To delete the all values or document
db.BookDetails.deleteMany({ })

//To verify the deleted document
db.BookDetails.find().pretty()
```

Projection Operations:

```
//To retrieve specific include field values or document
db.ProgrammingBooks.find({}, { title: 1, author: 1 })

[
  {
    _id: 1,
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin'
  },

  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford'
  },

  {
    _id: 3,
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides'
  },

  {
    _id: 4,
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein'
  },

  {
    _id: 5,
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes'
  }
]
```

```
//To retrieve specific exclude field values or document
db.BookDetails.find({}, { year:0 } )

[
  {
    _id: 1,
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development'
  },

  {
    _id: 2,
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript'
  },

  {
    _id: 3,
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Software Design'
  },

  {
    _id: 4,
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford
Stein',
    category: 'Algorithms'
  },

  {
    _id: 5,
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to
Programming',
    author: 'Eric Matthes',
    category: 'Python'
  }
]
```

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

```
//To create the new database as well as switch the database if not existing
use Movies

//To create the collection inside the database
db.createCollection("MovieDetails")

//To insert the multiple values or documents
db.MovieDetails.insertMany([
  { _id: 1, title: "Inception", director: "Christopher Nolan", genre: "Science Fiction", year: 2010, ratings: { imdb: 8.8, rottenTomatoes: 87 } },
  { _id: 2, title: "The Matrix", director: "Wachowskis", genre: "Science Fiction", year: 1999, ratings: { imdb: 8.7, rottenTomatoes: 87 } },
  { _id: 3, title: "The Godfather", director: "Francis Ford Coppola", genre: "Crime", year: 1972, ratings: { imdb: 9.2, rottenTomatoes: 97 } }
]);
```

Using projection to perform the query:

1. Including Specific Fields:

```
//To retrieve specific include field values or document
db.MovieDetails.find({}, { title: 1, director: 1 })

[
  { _id: 1, title: 'Inception', director: 'Christopher Nolan' },
  { _id: 2, title: 'The Matrix', director: 'Wachowskis' },
  { _id: 3, title: 'The Godfather', director: 'Francis Ford Coppola' }
]
3
```

2. Excluding Specific Fields:

```
//To retrieve specific exclude field values or document
db.MovieDetails.find({}, { ratings: 0 })
```

```
[
  {
    _id: 1,
    title: 'Inception',
    director: 'Christopher Nolan',
    genre: 'Science Fiction',
    year: 2010
  },
  {
    _id: 2,
    title: 'The Matrix',
    director: 'Wachowskis',
    genre: 'Science Fiction',
    year: 1999
  },
  {
    _id: 3,
    title: 'The Godfather',
    director: 'Francis Ford Coppola',
    genre: 'Crime',
    year: 1972
  }
]
```

```
// Combine query filter with a projection...
```

```
db.MovieDetails.find({ director: "Christopher Nolan" }, { title: 1, year: 1, _id: 0 })
```

```
[ { title: 'Inception', year: 2010 } ]
```

2b. Develop a MongoDB query to display the first 5 documents from the results obtained in a. [use of limit and find] Using Above same Database

```
db.MovieDetails.insertMany([
  {_id: 4, title: "Pulp Fiction", director: "Quentin Tarantino", genre: "Crime", year: 1994, ratings: { imdb: 8.9, rottenTomatoes: 92 } },
  {_id: 5, title: "The Shawshank Redemption", director: "Frank Darabont", genre: "Drama", year: 1994, ratings: { imdb: 9.3, rottenTomatoes: 91 } },
  {_id: 6, title: "The Dark Knight", director: "Christopher Nolan", genre: "Action", year: 2008, ratings: { imdb: 9.0, rottenTomatoes: 94 } },
  {_id: 7, title: "Fight Club", director: "David Fincher", genre: "Drama", year: 1999, ratings: { imdb: 8.8, rottenTomatoes: 79 } }
]);

//Query with Projection and Limit command...
db.MovieDetails.find({}, { title: 1, director: 1, year: 1, _id: 0 }).limit(5)
```

```
[
  { title: 'Inception', director: 'Christopher Nolan', year: 2010 },
  { title: 'The Matrix', director: 'Wachowskis', year: 1999 },
  { title: 'The Godfather', director: 'Francis Ford Coppola', year: 1972 },
  { title: 'Pulp Fiction', director: 'Quentin Tarantino', year: 1994 },
  { title: 'The Shawshank Redemption', director: 'Frank Darabont', year: 1994 }
]
```

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

//To create the new database as well as switch the database if not existing

use companyDB

//To insert the multiple values or documents

```
db.Employees.insertMany([
  { name: "Alice", age: 30, department: "HR", salary: 50000, joinDate: new Date("2015-01-15") },
  { name: "Bob", age: 24, department: "Engineering", salary: 70000, joinDate: new Date("2019-03-10") },
  { name: "Charlie", age: 29, department: "Engineering", salary: 75000, joinDate: new Date("2017-06-23") },
  { name: "David", age: 35, department: "Marketing", salary: 60000, joinDate: new Date("2014-11-01") },
  { name: "Eve", age: 28, department: "Finance", salary: 80000, joinDate: new Date("2018-08-19") }
])
```

1. $eq (Equal): Find employees in the “Engineering” department.
   db.Employees.find({ department: { $eq: "Engineering" } }).pretty()

```
[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf7'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf8'),
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  }
```

```
]
```

2. $ne (Not Equal): Find employees who are not in the “HR” department.
db.Employees.find({ department: { $ne: "HR" } }).pretty()

```
[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf7'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf8'),
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf9'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdfa'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
  }
]
```

3. $gt (Greater Than): Find employees who are older than 30.

```
db.Employees.find({ age: { $gt: 30 } }).pretty()
```

```
[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf9'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  }
]
```

4.$lt (Less Than): Find employees with a salary less than 70000.

```
db.Employees.find({ salary: { $lt: 70000 } }).pretty()
```

```
[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf6'),
    name: 'Alice',
    age: 30,
    department: 'HR',
    salary: 50000,
    joinDate: ISODate('2015-01-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf9'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  }
]
```

5. $gte (Greater Than or Equal): Find employees who joined on or after January 1, 2018.

```
db.Employees.find({ joinDate: { $gte: new Date("2018-01-01") } }).pretty()
```

```
[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf7'),
    name: 'Bob',
    age: 24,
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdfa'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
  }
]
```

6. $lte (Less Than or Equal): Find employees who are 28 years old or younger.

```
db.Employees.find({ age: { $lte: 28 } }).pretty()
```

```
[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf7'),
    name: 'Bob',
    age: 24,
```

```
    department: 'Engineering',
    salary: 70000,
    joinDate: ISODate('2019-03-10T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdfa'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
  }
]
```

Queries Using Logical Selectors:

1. $and (Logical AND): Find employees who are in the “Engineering” department and have a salary greater than 70000.

```
db.Employees.find({
  $and: [
    { department: "Engineering" },
    { salary: { $gt: 70000 } }
  ]
}).pretty()
```

```
[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf8'),
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
```

```
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  }
]
```

2.$or (Logical OR): Find employees who are either in the “HR” department or have a salary less than 60000.

```
db.Employees.find({
  $or: [
    { department: "HR" },
    { salary: { $lt: 60000 } }
  ]
}).pretty()

[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf6'),
    name: 'Alice',
    age: 30,
    department: 'HR',
    salary: 50000,
    joinDate: ISODate('2015-01-15T00:00:00.000Z')
  }
]
```

3. $not (Logical NOT): Find employees who are not in the “Engineering” department.

```
db.Employees.find({
  department: {
    $not: { $eq: "Engineering" }
  }
}).pretty()

[
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf6'),
    name: 'Alice',
    age: 30,
    department: 'HR',
    salary: 50000,
    joinDate: ISODate('2015-01-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf9'),
    name: 'David',
```

```
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
   },
   {
    _id: ObjectId('666c18217d3bfa1feacdcdfa'),
    name: 'Eve',
    age: 28,
    department: 'Finance',
    salary: 80000,
    joinDate: ISODate('2018-08-19T00:00:00.000Z')
   }
  ]
```

3. $nor (Logical NOR): Find employees who are neither in the “HR” department nor have a salary greater than 75000.

```
 db.Employees.find({
 $nor: [
  { department: "HR" },
  { salary: { $gt: 75000 } }
 ]
}).pretty()
```

```
[
 {
  _id: ObjectId('666c18217d3bfa1feacdcdf7'),
  name: 'Bob',
  age: 24,
  department: 'Engineering',
  salary: 70000,
  joinDate: ISODate('2019-03-10T00:00:00.000Z')
 },
 {
  _id: ObjectId('666c18217d3bfa1feacdcdf8'),
```

```
    name: 'Charlie',
    age: 29,
    department: 'Engineering',
    salary: 75000,
    joinDate: ISODate('2017-06-23T00:00:00.000Z')
  },
  {
    _id: ObjectId('666c18217d3bfa1feacdcdf9'),
    name: 'David',
    age: 35,
    department: 'Marketing',
    salary: 60000,
    joinDate: ISODate('2014-11-01T00:00:00.000Z')
  }
]
```

3b. Execute query selectors (Geospatial selectors, Bitwise selectors) and list out the results on any collection.

Geospatial Selectors:

```
use geodatabase
db.Places.insertMany([
  { name: "Central Park", location: { type: "Point", coordinates: [-73.9654, 40.7829] } },
  { name: "Times Square", location: { type: "Point", coordinates: [-73.9851, 40.7580] } },
  { name: "Brooklyn Bridge", location: { type: "Point", coordinates: [-73.9969, 40.7061] } },
  { name: "Empire State Building", location: { type: "Point", coordinates: [-73.9857, 40.7488] } },
  { name: "Statue of Liberty", location: { type: "Point", coordinates: [-74.0445, 40.6892] } }
])

// Create a geospatial index
db.Places.createIndex({ location: "2dsphere" })
```

Geospatial Queries:

1. $near (Find places near a certain point): Find places near a specific coordinate, for example, near Times Square.

```
db.Places.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [-73.9851, 40.7580]
      },
      $maxDistance: 5000 // distance in meters
    }
  }
}).pretty()
```

```
[
  {
    _id: ObjectId('666c25eb7d3bfa1feacdcdfc'),
    name: 'Times Square',
    location: { type: 'Point', coordinates: [ -73.9851, 40.758 ] }
  },
  {
    _id: ObjectId('666c25eb7d3bfa1feacdcdfe'),
    name: 'Empire State Building',
    location: { type: 'Point', coordinates: [ -73.9857, 40.7488 ] }
  },
  {
    _id: ObjectId('666c25eb7d3bfa1feacdcdfb'),
    name: 'Central Park',
    location: { type: 'Point', coordinates: [ -73.9654, 40.7829 ] }
  }
]
```

2. $geoWithin (Find places within a specific area): Find places within a specific polygon, for example, an area covering part of Manhattan.

```
db.Places.find({
  location: {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [
```

```
        [
          [-70.016, 35.715],
          [-74.014, 40.717],
          [-73.990, 40.730],
          [-73.990, 40.715],
          [-70.016, 35.715]
        ]
      ]
    }
  }
 }
}).pretty()

[
  {
    _id: ObjectId('666c25eb7d3bfa1feacdcdfd'),
    name: 'Brooklyn Bridge',
    location: { type: 'Point', coordinates: [ -73.9969, 40.7061 ] }
  }
]
```

Bitwise Selectors:

```
use techDB
db.Devices.insertMany([
  { name: "Device A", status: 5 }, // Binary: 0101
  { name: "Device B", status: 3 }, // Binary: 0011
  { name: "Device C", status: 12 }, // Binary: 1100
  { name: "Device D", status: 10 }, // Binary: 1010
  { name: "Device E", status: 7 }  // Binary: 0111
])
```

Execute Bitwise Queries:

1. $bitsAllSet (Find documents where all bits are set): Find devices where the binary status has both the 1st and 3rd bits set (binary mask 0101, or decimal 5).

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

```
[
  {
    _id: ObjectId('666c28847d3bfa1feacdce00'),
    name: 'Device A',
    status: 5
  },
  {
    _id: ObjectId('666c28847d3bfa1feacdce04'),
    name: 'Device E',
    status: 7
  }
]
```

2.$bitsAnySet (Find documents where any of the bits are set): Find devices where the binary status has at least the 2nd bit set (binary mask 0010, or decimal 2).

```
db.Devices.find({
  status: { $bitsAnySet: [1] }
}).pretty()
```

```
[
  {
    _id: ObjectId('666c28847d3bfa1feacdce01'),
    name: 'Device B',
    status: 3
  },
```

```
  {
    _id: ObjectId('666c28847d3bfa1feacdce03'),
    name: 'Device D',
    status: 10
  },
  {
    _id: ObjectId('666c28847d3bfa1feacdce04'),
    name: 'Device E',
    status: 7
  }
]
```

3.$bitsAllClear (Find documents where all bits are clear): Find devices where the binary status has both the 2nd and 4th bits clear (binary mask 1010, or decimal 10).

```
db.Devices.find({
  status: { $bitsAllClear: [1, 3] }
}).pretty()
```

```
[
  {
    _id: ObjectId('666c28847d3bfa1feacdce00'),
    name: 'Device A',
    status: 5
  }
]
```

4. $bitsAnyClear (Find documents where any of the bits are clear): Find devices where the binary status has at least the 1st bit clear (binary mask 0001, or decimal 1).

```
db.Devices.find({
  status: { $bitsAnyClear: [0] }
}).pretty()
```

```
[
  {
    _id: ObjectId('666c28847d3bfa1feacdce02'),
    name: 'Device C',
    status: 12
  },
  {
    _id: ObjectId('666c28847d3bfa1feacdce03'),
    name: 'Device D',
    status: 10
  }
]
```

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

```
use retailDB


db.Products.insertMany([
  {
    name: "Laptop",
    brand: "BrandA",
    features: [
      { name: "Processor", value: "Intel i7" },
      { name: "RAM", value: "16GB" },
      { name: "Storage", value: "512GB SSD" }
    ],
    reviews: [
      { user: "Alice", rating: 5, comment: "Excellent!" },
      { user: "Bob", rating: 4, comment: "Very good" },
      { user: "Charlie", rating: 3, comment: "Average" }
    ]
  },
  {
    name: "Smartphone",
    brand: "BrandB",
    features: [
      { name: "Processor", value: "Snapdragon 888" },
      { name: "RAM", value: "8GB" },
      { name: "Storage", value: "256GB" }
    ],
    reviews: [
      { user: "Dave", rating: 4, comment: "Good phone" },
      { user: "Eve", rating: 2, comment: "Not satisfied" }
    ]
```

```
  }
])
```

Use Projection Operators:

1. $ Projection Operator: Find the product named “Laptop” and project the review from the user “Alice”.

```
db.Products.find(
  { name: "Laptop", "reviews.user": "Alice" },
  { "reviews.$": 1 }
).pretty()
```

```
[
  {
    _id: ObjectId('666c2f237d3bfa1feacdce05'),
    reviews: [ { user: 'Alice', rating: 5, comment: 'Excellent!' } ]
  }
]
```

2.$elemMatch Projection Operator: Find the product named “Laptop” and project the review where the rating is greater than 4.

```
db.Products.find(
  { name: "Laptop" },
  { reviews: { $elemMatch: { rating: { $gt: 4 } } } }
).pretty()
```

```
[
  {
    _id: ObjectId('666c2f237d3bfa1feacdce05'),
    reviews: [ { user: 'Alice', rating: 5, comment: 'Excellent!' } ]
```

```
  } ]
```

3. $slice Projection Operator: Find the product named “Smartphone” and project the first review.

```
db.Products.find(
  { name: "Smartphone" },
  { reviews: { $slice: 1 } }
).pretty()
```

```
[
  {
    _id: ObjectId('666c2f237d3bfa1feacdce06'),
    name: 'Smartphone',
    brand: 'BrandB',
    features: [
      { name: 'Processor', value: 'Snapdragon 888' },
      { name: 'RAM', value: '8GB' },
      { name: 'Storage', value: '256GB' }
    ],
    reviews: [ { user: 'Dave', rating: 4, comment: 'Good phone' } ]
  }
]
```

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

```
use salesDB


db.Sales.insertMany([
  { date: new Date("2024-01-01"), product: "Laptop", price: 1200, quantity: 1, customer: "Amar" },
  { date: new Date("2024-01-02"), product: "Laptop", price: 1200, quantity: 2, customer: "Babu" },
  { date: new Date("2024-01-03"), product: "Mouse", price: 25, quantity: 5, customer: "Chandra" },
  { date: new Date("2024-01-04"), product: "Keyboard", price: 45, quantity: 3, customer: "Amar" },
  { date: new Date("2024-01-05"), product: "Monitor", price: 300, quantity: 1, customer: "Babu" },
  { date: new Date("2024-01-06"), product: "Laptop", price: 1200, quantity: 1, customer: "Deva" }
])
```

Execute Aggregation Operations:

1. $avg (Average): Calculate the average price of each product.

```
db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      averagePrice: { $avg: "$price" }
    }
  }
]).pretty()


[
```

```
  { _id: 'Laptop', averagePrice: 1200 },
  { _id: 'Keyboard', averagePrice: 45 },
  { _id: 'Mouse', averagePrice: 25 },
  { _id: 'Monitor', averagePrice: 300 }
]
```

2. $min (Minimum): Find the minimum price of each product.

```
db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      minPrice: { $min: "$price" }
    }
  }
]).pretty()
```

```
[
  { _id: 'Mouse', minPrice: 25 },
  { _id: 'Keyboard', minPrice: 45 },
  { _id: 'Monitor', minPrice: 300 },
  { _id: 'Laptop', minPrice: 1200 }
]
```

3.$max (Maximum): Find the maximum price of each product.

```
db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      maxPrice: { $max: "$price" }
    }
  }
]).pretty()

[
  { _id: 'Mouse', maxPrice: 25 },
```

```
    { _id: 'Keyboard', maxPrice: 45 },
    { _id: 'Monitor', maxPrice: 300 },
    { _id: 'Laptop', maxPrice: 1200 }
  ]
```

4. $push (Push Values to an Array): Group sales by customer and push each purchased product into an array.

```
db.Sales.aggregate([
  {
   $group: {
     _id: "$customer",
     products: { $push: "$product" }
   }
  }
]).pretty()
```

```
[
  { _id: 'Babu', products: [ 'Laptop', 'Monitor' ] },
  { _id: 'Amar', products: [ 'Laptop', 'Keyboard' ] },
  { _id: 'Chandra', products: [ 'Mouse' ] },
  { _id: 'Deva', products: [ 'Laptop' ] }
]
```

5.$addToSet (Add Unique Values to an Array): Group sales by customer and add each unique purchased product to an array.

```
db.Sales.aggregate([
  {
   $group: {
     _id: "$customer",
     uniqueProducts: { $addToSet: "$product" }
   }
  }
]).pretty()
```

```
[
  { _id: 'Amar', uniqueProducts: [ 'Keyboard', 'Laptop' ] },
  { _id: 'Babu', uniqueProducts: [ 'Monitor', 'Laptop' ] },
  { _id: 'Deva', uniqueProducts: [ 'Laptop' ] },
  { _id: 'Chandra', uniqueProducts: [ 'Mouse' ] } ]
```

Combining Aggregation Operations:

1. Calculate the total quantity and total sales amount for each product, and list all customers who purchased each product.

```
db.Sales.aggregate([
  {
    $group: {
      _id: "$product",
      totalQuantity: { $sum: "$quantity" },
      totalSales: { $sum: { $multiply: ["$price", "$quantity"] } },
      customers: { $addToSet: "$customer" }
    }
  }
]).pretty()
```

```
[
  {
    _id: 'Mouse',
    totalQuantity: 5,
    totalSales: 125,
    customers: [ 'Chandra' ]
  },
  {
    _id: 'Keyboard',
    totalQuantity: 3,
    totalSales: 135,
```

```
    customers: [ 'Amar' ]
  },
  {
    _id: 'Monitor',
    totalQuantity: 1,
    totalSales: 300,
    customers: [ 'Babu' ]
  },
  {
    _id: 'Laptop',
    totalQuantity: 4,
    totalSales: 4800,
    customers: [ 'Amar', 'Babu', 'Deva' ]
  }
]
```

Experiment 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)

```
use restaurantDB
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" }
    ]
  }
])
```

Aggregation Pipeline and its operations:

1. Execute Aggregation Pipeline and its operations (pipeline must contain match, group, sort, project, $skip etc.)

```
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
  }
]).pretty()
```

```
[
  { averageRating: 4.5, totalReviews: 2, restaurant: 'Curry Palace' },
  { averageRating: 4.5, totalReviews: 2, restaurant: 'Taco Stand' } ]
```

Experiment 7: 7a. 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.

```
use vacationRentals


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"
    }
  }
])
```

Query to Find Listings with Host Picture URLs:

Now that the collection is set up, you can run the query to find all listings with listing_url, name, address, and host_picture_url where the host has a picture URL.

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

```
  }
).pretty()

[
  {
    _id: ObjectId('666c40ce85a7615d27cdcdfb'),
    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('666c40ce85a7615d27cdcdfd'),
    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' }
  }
]
```

7b. Using E-commerce collection write a query to display reviews summary.

```
use 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" }
    ]
  }
])
```

To display a summary of reviews in an e-commerce collection, we can assume the ecommerce database contains a products collection with documents structured to include reviews. Each product document could have a reviews array with review details such as rating, comment, and user.

```
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
    } } ]).pretty()
```

```
[
```

```
  {
    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'
  }
]
```

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

```
// Switch to the restaurantDB database
use restaurantDB


// Insert sample documents into the restaurants collection
db.restaurants.insertMany([
  {
    name: "Biryani House",
    cuisine: "Indian",
    location: "Downtown",
    reviews: [
      { user: "Aarav", rating: 5, comment: "Amazing biryani!" },
      { user: "Bhavana", rating: 4, comment: "Great place!" }
    ],
    contact: { phone: "1234567890", email: "contact@biryanihouse.com" }
  },
  {
    name: "Curry Palace",
    cuisine: "Indian",
    location: "Downtown",
    reviews: [
      { user: "Gaurav", rating: 4, comment: "Spicy and tasty!" },
      { user: "Harini", rating: 5, comment: "Best curry in town!" }
    ],
    contact: { phone: "0987654321", email: "contact@currypalace.com" }
  },
  {
    name: "Taco Stand",
    cuisine: "Mexican",
    location: "Downtown",
```

```
    reviews: [
      { user: "Ishaan", rating: 5, comment: "Fantastic tacos!" },
      { user: "Jaya", rating: 4, comment: "Very authentic" }
    ],
    contact: { phone: "1122334455", email: "contact@tacostand.com" }
  }
])

// Create a unique index on the contact.email field
db.restaurants.createIndex({ "contact.email": 1 }, { unique: true })

// Create a sparse index on the location field
db.restaurants.createIndex({ location: 1 }, { sparse: true })

// Create a compound index on the name and location fields
db.restaurants.createIndex({ name: 1, location: 1 })

// Create a multikey index on the reviews field
db.restaurants.createIndex({ reviews: 1 })

// Verify the created indexes
db.restaurants.getIndexes()
```

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

Create a text index on the fields you want to search

```
{
  "_id": 1,
  "title": "Wireless Bluetooth Headphones",
  "description": "High-quality sound with noise cancellation"
}
```

```
db.catalog.createIndex({ title: "text", description: "text" })
```

Perform a text search query for a given word

```
db.catalog.find(
  { $text: { $search: "Bluetooth" } },
  { score: { $meta: "textScore" } }  // To get relevance score
).sort({ score: { $meta: "textScore" } })
```

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

MongoDB Text Search (Exclude documents containing a word)

```
db.collection.find({
  $text: { $search: "some keyword" },
  "field": { $not: /excludedWord/i }
})
```

```
db.collection.find({
  $text: { $search: "some keyword" },
  "field": { $not: /apple/i }
})
```

Elasticsearch Query DSL

```
{
  "query": {
    "bool": {
      "must": {
        "match": { "content": "searchTerm" }
      },
      "must_not": {
        "match": { "content": "excludedWord" }
      }
    }
  }
}
```

MongoDB Aggregation with Text Search Excluding

If you want to exclude documents containing "spam" in the text indexed

```
db.collection.aggregate([
  {
    $match: {
      $text: { $search: "yourSearchTerm" }
    }
  },
  {
    $match: {
      "content": { $not: /spam/i }
    }
  } ])
```

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

Aggregation Pipeline for Text Search on catalog Collection:

```
db.catalog.aggregate([
  {
   $match: {
     $text: { $search: "your search keywords" }
   }
  },
  {
   $addFields: {
     score: { $meta: "textScore" }
   }
  },
  {
   $sort: { score: -1 }
  },
  {
   $project: {
     title: 1,
     description: 1,
     price: 1,
     score: 1
   }
  }
])


db.catalog.aggregate([
  {
   $match: {
     $text: { $search: "wireless headphones" }
```

```
    }
  },
  {
    $addFields: {
      score: { $meta: "textScore" }
    }
  },
  {
    $sort: { score: -1 }
  },
  {
    $project: {
      title: 1,
      description: 1,
      price: 1,
      score: 1
    }
  }
])
```

Experiment 10: