RAJARAJESWARI COLLEGE OF ENGINEERING

Approved by AICTE, New Delhi, Govt. of Karnataka, Affiliated to Visvesvaraya Technological University, Ramohalli Cross, Kumbalagodu Post, Bengaluru-74

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

MongoDB (BDSL456B)

LAB MANUAL

4TH SEMESTER

HOD
Dr.Rajesh K S
Prof., Dept. of AIML

PREPARED BY:
Dr.Prakasha P K
Dept. of AI&ML

Course Details:

Course Name: MongoDB

Course Code: BDSL456B

Course Objectives:

● Understand basic MongoDB functions, operators and types of operations in MongoDB.

● Demonstrate the use of Indexing, Advanced Indexing in MongoDB.

● Apply the aggregation and Map Reduction in MongoDB.

● Demonstrate text searching on collections in MongoDB.

Template for Practical Course and if AEC is a practical Course Annexure-V

MongoDB		Semester	4
Course Code	**BDSL456B**	CIE Marks	50
Teaching Hours/Week (L: T:P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	01		

Course objectives:

- Understand basic MongoDB functions, operators and types of operations in MongoDB.
- Demonstrate the use of Indexing, Advanced Indexing in MongoDB.
- Apply the aggregation and Map Reduction in MongoDB.
- Demonstrate text searching on collections in MongoDB.

Sl.NO	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) [Refer: Book 1 chapter 4].
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. [use of limit and find] [Refe: Book1 Chapter 4, book 2: chapter 5]
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 [Refer: Book 3 Chapter 13]
4	Create and demonstrate how projection operators ($, $elematch and $slice) would be used in the MondoDB. [Refer: Book 3 Chapter 14]
5	Execute Aggregation operations ($avg, $min,$max, $push, $addToSet etc.). students encourage to execute several queries to demonstrate various aggregation operators) [Refer: Book 3 Chapter 15]
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) [refer book 2: chapter 6]
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. [refer Book2: chapter 6]
8	a. Demonstrate creation of different types of indexes on collection (unique, sparse, compound and multikey indexes) b. Demonstrate optimization of queries using indexes. Refer: Book 2: Chapter 8 and Book 3: Chapter 12]
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 Refer: Book 2: Chapter 9]

10	Develop an aggregation pipeline to illustrate Text search on Catalog data collection. Refer: Book 2 :Chapter 9]

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

1. Make use of MangoDB commands and queries.
2. Illustrate the role of aggregate pipelines to extract data.
3. Demonstrate optimization of queries by creating indexes.
4. Develop aggregate pipelines for text search in collections.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.

- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

Suggested Learning Resources:

- **BOOK 1:** “MongoDB: The Definitive Guide”, Kristina chodorow, 2nd ed O’REILLY, 2013.
- **BOOK 2:** “*MongoDB in Action*” by KYLE BANKER et. al. 2nd ed, Manning publication, 2016
- **BOOK 3:** “MongoDB Complete Guide” by Manu Sharma 1st ed, bpb publication, 2023.
- **installation of MongoDB Video:** https://www.youtube.com/watch?v=dEm2AS5amyA
- **video on Aggregation:** https://www.youtube.com/watch?v=vx1C8EyTa7Y
- **MongoDB in action book Code download URL: h**ttps://www.manning.com/downloads/529
- **MongoDB Exercise URL:** https://www.w3resource.com/mongodb-exercises/

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)

To insert this data into MongoDB, you can use the following commands in the **MongoDB shell** or a MongoDB client like **MongoDB Compass**:

1. **Connect to MongoDB:**

```
> mongosh
```

2. **Create a database and a collection:**

```
>use school
>db.createCollection("students");
```

3 . **Insert the data into the collection:**

```
>db.students.insertMany([
  { "student_id": 1, "name": "Alice Johnson", "age": 20, "gender": "Female", "courses": ["Math", "Science"], "grade": "A", "address": { "street": "123 Maple St", "city": "Springfield", "state": "IL", "zip": "62701" } },
  { "student_id": 2, "name": "Bob Smith", "age": 22, "gender": "Male", "courses": ["English", "History"], "grade": "B", "address": { "street": "456 Oak St", "city": "Springfield", "state": "IL", "zip": "62702" } },
  { "student_id": 3, "name": "Charlie Brown", "age": 23, "gender": "Male", "courses": ["Math", "Art"], "grade": "C", "address": { "street": "789 Pine St", "city": "Springfield", "state": "IL", "zip": "62703" } },
  { "student_id": 4, "name": "Diana Prince", "age": 21, "gender": "Female", "courses": ["Science", "History"], "grade": "B", "address": { "street": "321 Birch St", "city": "Springfield", "state": "IL", "zip": "62704" } },
  { "student_id": 5, "name": "Eve Adams", "age": 20, "gender": "Female", "courses": ["Math", "Science"], "grade": "A", "address": { "street": "654 Elm St", "city": "Springfield", "state": "IL", "zip": "62705" } },
  { "student_id": 6, "name": "Frank Wright", "age": 22, "gender": "Male", "courses": ["English", "Science"], "grade": "B", "address": { "street": "987 Cedar St", "city": "Springfield", "state": "IL", "zip": "62706" } },
  { "student_id": 7, "name": "Grace Hall", "age": 23, "gender": "Female", "courses": ["History", "Art"], "grade": "C", "address": { "street": "147 Oak St", "city": "Springfield", "state": "IL", "zip": "62707" } },
  { "student_id": 8, "name": "Henry Carter", "age": 21, "gender": "Male", "courses": ["Math", "History"], "grade": "B", "address": { "street": "258 Pine St", "city": "Springfield", "state": "IL", "zip": "62708" } },
  { "student_id": 9, "name": "Isla Turner", "age": 20, "gender": "Female", "courses": ["Science", "English"], "grade": "A", "address": { "street": "369 Birch St", "city": "Springfield", "state": "IL", "zip": "62709" } },
  { "student_id": 10, "name": "Jack White", "age": 22, "gender": "Male", "courses": ["Math", "Science"], "grade": "B", "address": { "street": "741 Maple St", "city": "Springfield", "state": "IL", "zip": "62710" } },
  { "student_id": 11, "name": "Kara Green", "age": 23, "gender": "Female", "courses":
["English", "Art"], "grade": "C", "address": { "street": "852 Oak St", "city": "Springfield",
"state": "IL", "zip": "62711" } },
  { "student_id": 12, "name": "Liam Hill", "age": 21, "gender": "Male", "courses": ["History",
"Science"], "grade": "B", "address": { "street": "963 Cedar St", "city": "Springfield", "state":
"IL", "zip": "62712" } },
  { "student_id": 13, "name": "Mia King", "age": 20, "gender": "Female", "courses": ["Math",
"Art"], "grade": "A", "address": { "street": "123 Pine St", "city": "Springfield", "state": "IL",
"zip": "62713" } },
  { "student_id": 14, "name": "Noah Wright", "age": 22, "gender": "Male", "courses":
["Science", "English"], "grade": "B", "address": { "street": "456 Birch St", "city": "Springfield",
"state": "IL", "zip": "62714" } },
  { "student_id": 15, "name": "Olivia Brown", "age": 23, "gender": "Female", "courses":
["History", "Math"], "grade": "C", "address": { "street": "789 Oak St", "city": "Springfield",
"state": "IL", "zip": "62715" } },
  { "student_id": 16, "name": "Peter Johnson", "age": 21, "gender": "Male", "courses":
["Math", "Science"], "grade": "B", "address": { "street": "321 Cedar St", "city": "Springfield",
"state": "IL", "zip": "62716" } },
  { "student_id": 17, "name": "Quinn Adams", "age": 20, "gender": "Female", "courses":
["English", "History"], "grade": "A", "address": { "street": "654 Maple St", "city":
"Springfield", "state": "IL", "zip": "62717" } },
  { "student_id": 18, "name": "Ryan Clark", "age": 22, "gender": "Male", "courses":
["Science", "Art"], "grade": "B", "address": { "street": "987 Pine St", "city": "Springfield",
"state": "IL", "zip": "62718" } },
  { "student_id": 19, "name": "Sophie Evans", "age": 23, "gender": "Female", "courses":
["Math", "History"], "grade": "C", "address": { "street": "147 Birch St", "city": "Springfield",
"state": "IL", "zip": "62719" } },
  { "student_id": 20, "name": "Thomas Moore", "age": 21, "gender": "Male", "courses":
["Science", "English"], "grade": "B", "address": { "street": "258 Oak St", "city": "Springfield",
"state": "IL", "zip": "62720" } },
  { "student_id": 21, "name": "Uma Harris", "age": 20, "gender": "Female", "courses":
["Math", "Art"], "grade": "A", "address": { "street": "369 Cedar St", "city": "Springfield",
"state": "IL", "zip": "62721" } },
  { "student_id": 22, "name": "Victor Lewis", "age": 22, "gender": "Male", "courses":
["History", "Science"], "grade": "B", "address": { "street": "741 Maple St", "city":
"Springfield", "state": "IL", "zip": "62722" } },
  { "student_id": 23, "name": "Wendy Scott", "age": 23, "gender": "Female", "courses":
["English", "Math"], "grade": "C", "address": { "street": "852 Pine St", "city": "Springfield",
"state": "IL", "zip": "62723" } },
  { "student_id": 24, "name": "Xander Young", "age": 21, "gender": "Male", "courses":
["Science", "History"], "grade": "B", "address": { "street": "963 Birch St", "city": "Springfield",
"state": "IL", "zip": "62724" } },
  { "student_id": 25, "name": "Yara Bell", "age": 20, "gender": "Female", "courses": ["Math",
"Science"], "grade": "A", "address": { "street": "123 Oak St", "city": "Springfield", "state": "IL",
"zip": "62725" } }]);
```

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

i). Using the Where Clause

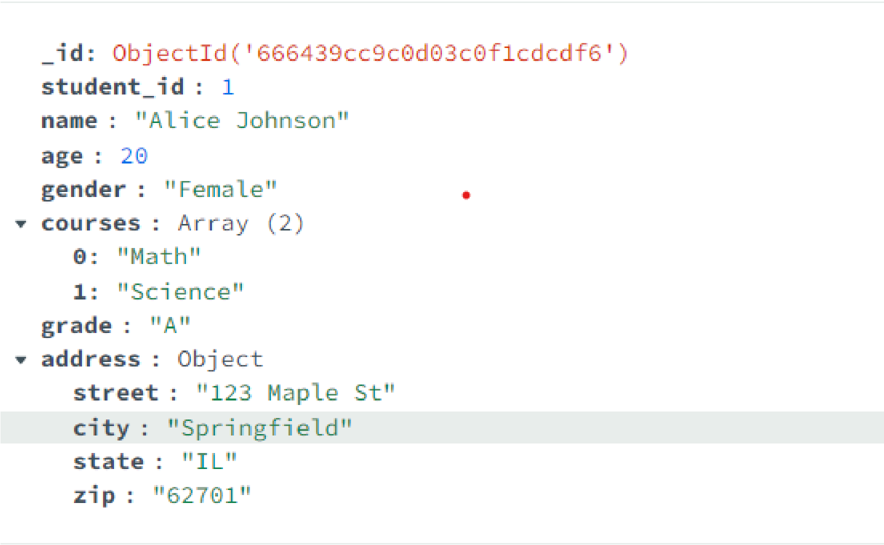
The **`where` clause** in MongoDB is used to filter documents that match a specified condition.

Example: Find students who are 20 years old.

Query:

```
> db.students.find({ age: 20 })
```

Sample output:



```
  _id: ObjectId('666439cc9c0d03c0f1cdcdf6')
  student_id : 1
  name : "Alice Johnson"
  age : 20
  gender : "Female"
▾ courses : Array (2)
    0: "Math"
    1: "Science"
  grade : "A"
▾ address : Object
    street : "123 Maple St"
    city : "Springfield"
    state : "IL"
    zip : "62701"
```

ii). Using the AND Operation

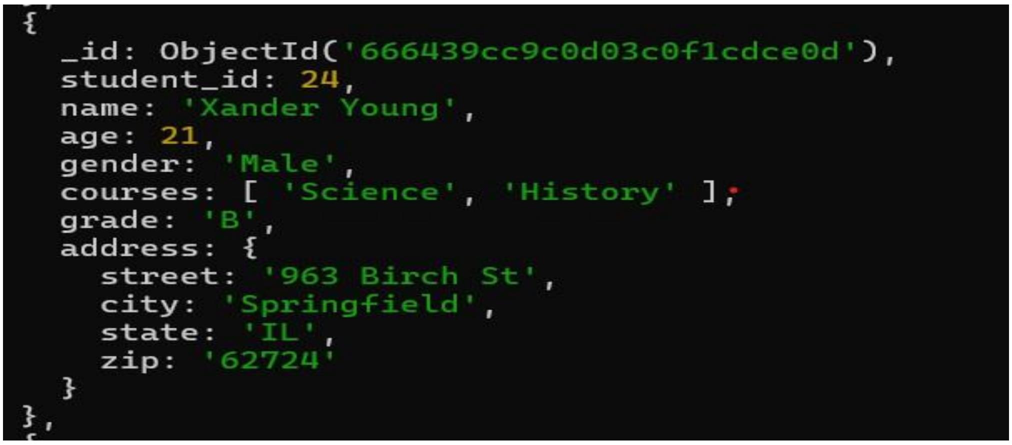
The **AND operation** can be achieved by specifying multiple conditions within the same query object. MongoDB treats all key-value pairs within a single query object as an implicit AND.

Example: Find students who are 21 years old and have a grade of 'B'.

Query:

```
> db.students.find({ age: 21, grade: 'B' })
```

Sample output:



```
{
  _id: ObjectId('666439cc9c0d03c0f1cdce0d'),
  student_id: 24,
  name: 'Xander Young',
  age: 21,
  gender: 'Male',
  courses: [ 'Science', 'History' ],
  grade: 'B',
  address: {
    street: '963 Birch St',
    city: 'Springfield',
    state: 'IL',
    zip: '62724'
  }
},
```

iii). Using the OR Operation

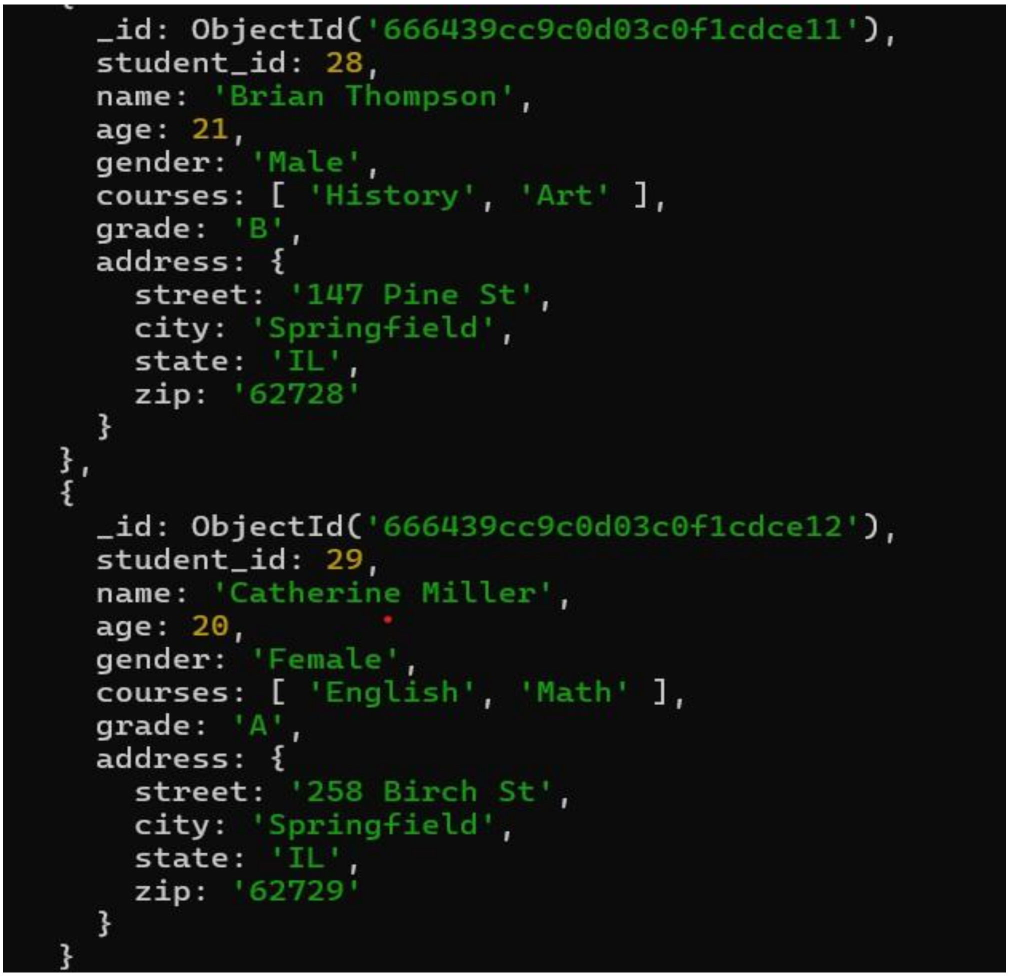
The **OR operation** is performed using the `$or` operator. This operator takes an array of query conditions and matches documents that satisfy at least one of them.

Example: Find students who are either 21 years old or have a grade of 'A'.

<u>Query:</u>

```
> db.students.find({ age: 21,

   $or: [

  { grade: 'A' },

  { courses: 'Math' } ]

   })
```

Sample output:



```
    _id: ObjectId('666439cc9c0d03c0f1cdce11'),
    student_id: 28,
    name: 'Brian Thompson',
    age: 21,
    gender: 'Male',
    courses: [ 'History', 'Art' ],
    grade: 'B',
    address: {
      street: '147 Pine St',
      city: 'Springfield',
      state: 'IL',
      zip: '62728'
    }
  },
  {
    _id: ObjectId('666439cc9c0d03c0f1cdce12'),
    student_id: 29,
    name: 'Catherine Miller',
    age: 20,
    gender: 'Female',
    courses: [ 'English', 'Math' ],
    grade: 'A',
    address: {
      street: '258 Birch St',
      city: 'Springfield',
      state: 'IL',
      zip: '62729'
    }
  }
```

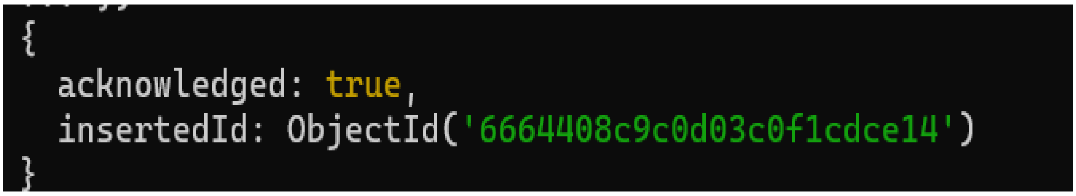
b. Execute the Commands of MongoDB and operations in MongoDB: Insert, Query, Update, Delete and Projection.

1. Insert

Command to insert a single document:

```
> db.students.insertOne({
 student_id: 51,
 name: "Yasmin Brooks",
 age: 24,
 gender: "Female",
 courses: ["Physics", "Chemistry"],
 grade: "B",
 address: {
    street: "987 Willow St",
    city: "Springfield",
    state: "IL",
    zip: "62751"
  } } )
```

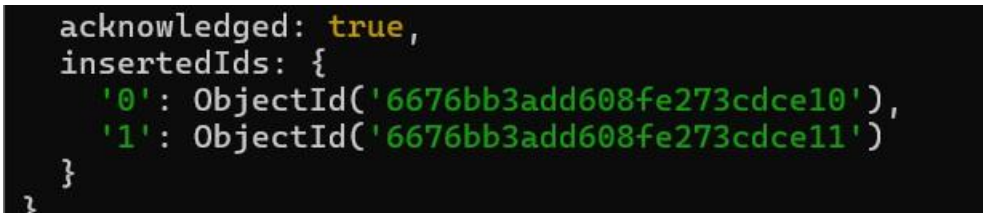
Sample output:



```
{
  acknowledged: true,
  insertedId: ObjectId('6664408c9c0d03c0f1cdce14')
}
```

```
// to insert multiple documents:
> db.students.insertMany([
  {
    student_id: 52,
    name: "Zoe Taylor",
    age: 22,
    gender: "Female",
    courses: ["Math", "History"],
    grade: "A",
    address: {
      street: "123 Palm St",
      city: "Springfield",
      state: "IL",
      zip: "62752"
    } },
  {
    student_id: 53,
    name: "Liam Martinez",
    age: 23,
    gender: "Male",

    courses: ["Biology", "Art"],

    grade: "C",

    address: {

      street: "456 Fir St",

      city: "Springfield",

      state: "IL",

      zip: "62753"

    }}])
```

output:



```
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6676bb3add608fe273cdce10'),
    '1': ObjectId('6676bb3add608fe273cdce11')
  }
}
```

2. Query

Find all documents:

```
> db.students.find()
```

Find students who are 22 years old:

```
> db.students.find({ age: 22 })
```

Find a student by student id

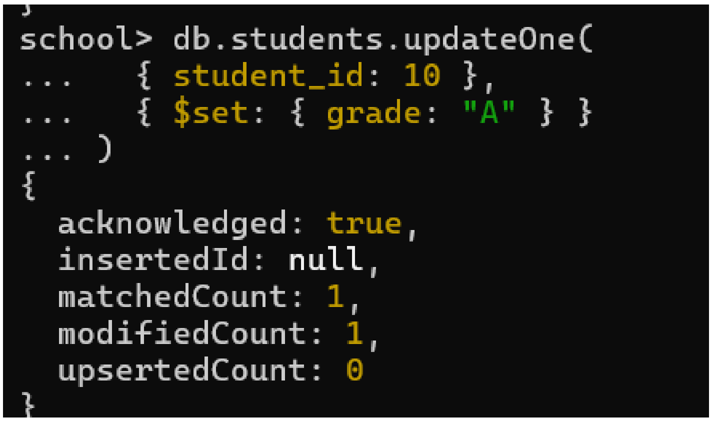
```
> db.students.findOne({ student_id: 10 })
```

3. Update

Update a student's grade by `student_id`:

```
> db.students.updateOne(

  { student_id: 10 },

  { $set: { grade: "A" } } )
```

Output:



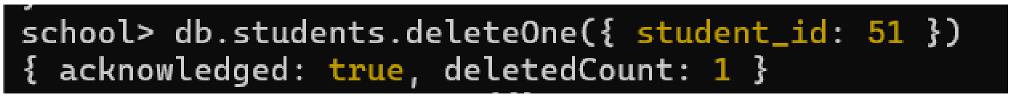
```
school> db.students.updateOne(
...   { student_id: 10 },
...   { $set: { grade: "A" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

4. Delete

Delete a student by student id:

> db.students.deleteOne({ student_id: 51 })

output:



```
school> db.students.deleteOne({ student_id: 51 })
{ acknowledged: true, deletedCount: 1 }
```

5. Projection

Find specific fields (name and age) of all students:

> db.students.find({}, { name: 1, age: 1, _id: 0 })

Find students with specific fields and apply conditions:

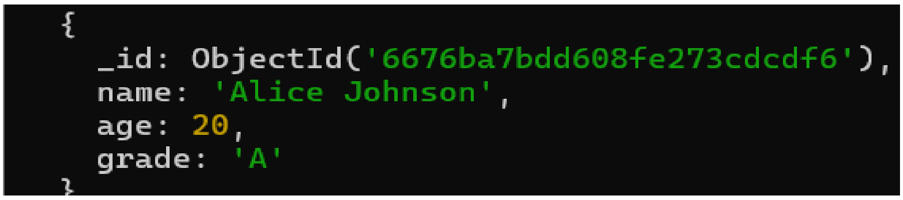
> db.students.find(

{ age: { $gte: 22 } },

{ name: 1, age: 1, grade: 1, _id: 0 }

)

Output:



```
  {
    _id: ObjectId('6676ba7bdd608fe273cdcdf6'),
    name: 'Alice Johnson',
    age: 20,
    grade: 'A'
  }
```

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

Here's how you can do it using the '`students'` **collection**:

Example Query: Include and Exclude Fields

Assume we want to include the `name` and `age` fields but exclude the _`id` and `address` fields from the documents.

Command:

```
> db.students.find({},

  { name: 1, age: 1 }                              // Include name and age, exclude _id and address

  )
```

Example:Output:

```
[

  { "name": "John Doe", "age": 21 },

  { "name": "Jane Smith", "age": 22 },

  { "name": "Michael Johnson", "age": 23 },

]
```

Example 1: Include `name, age,` and `grade`, but exclude `_id`.

```
> db.students.find( { },

  { name: 1, age: 1, grade: 1, }

  )
```

Example 2: Include only `name` and `courses`

```
> db.students.find( {},

  { name: 1, courses: 1}

)
```

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

To display the first 5 documents from the results obtained in the previous query, you can simply use the `limit` function in MongoDB. Here's how you can do it:

Example Query: Using Limit to Display First 5 Documents

```
db.students.find( { },

 { name: 1, age: 1 }

).limit(5)

.limit(5)
```

Example output:

This function limits the number of documents returned by the query to 5 (specified limit)

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

In MongoDB, query selectors are used to filter documents based on various conditions. These include **comparison selectors (like `$gt`, `$lt`, `$eq`, etc.)** and **logical selectors (like `$and`, `$or`, `$not`, etc.).**

Comparison Selectors

1). `$gt` (greater than): Find students who are older than 21.

<u>Query</u>

```
> db.students.find({ age: { $gt: 21 } })
```

Example output:

```
[{ "student_id": 10, "name": "Jack White", "age": 22, ... },

 { "student_id": 12, "name": "Michael Brown", "age": 22, ... },

]
```

2. `$lt` (less than): Find students who are younger than 22.

<u>Query</u>

```
> db.students.find({ age: { $lt: 22 } })
```

Example Output:

[{ "student_id": 4, "name": "Diana Prince", "age": 21, ... },

{ "student_id": 8, "name": "Henry Carter", "age": 21, ... },

]

3. `$eq` (equal): Find students who are exactly 23 years old.

<u>Query</u>

> **db.students.find({ age: { $eq: 23 } })**

Example output:

[{ "student_id": 11, "name": "Anna Davis", "age": 23, ... },

{ "student_id": 31, "name": "Ella Martinez", "age": 23, ... },]

4. `$ne` (not equal): Find students who are not 22 years old.

<u>Query</u>

> **db.students.find({ age: { $ne: 22 } })**

Example Output:

[{ "student_id": 4, "name": "Diana Prince", "age": 21, ... },

{ "student_id": 11, "name": "Anna Davis", "age": 23, ... },

]

5. `$in`: Find students who are either 21 or 23 years old.

<u>Query</u>

```
> db.students.find({ age: { $in: [21, 23] } })
```

Example Output:

```
[
  { "student_id": 4, "name": "Diana Prince", "age": 21, ... },
  { "student_id": 11, "name": "Anna Davis", "age": 23, ... },
  ...
]
```

6. `$nin`: Find students who are neither 21 nor 23 years old.

Query

```
> db.students.find({ age: { $nin: [21, 23] } })
```

Example Output:

```
[
  { "student_id": 10, "name": "Jack White", "age": 22, ... },
  { "student_id": 12, "name": "Michael Brown", "age": 22, ... },
  ...
]
```

Logical Selectors

1. `$and`: Find students who are 22 years old and have a grade of 'B'.

Query

```
> db.students.find({ $and: [{ age: 22 }, { grade: 'B' }] })
```

Example Output:

```
[
  { "student_id": 10, "name": "Jack White", "age": 22, "grade": "B",
... },
  { "student_id": 12, "name": "Michael Brown", "age": 22, "grade":
"B", ... },
]
```

2. `$or`: Find students who are either 21 years old or have a grade of 'A'.

Query

```
> db.students.find({ $or: [{ age: 21 }, { grade: 'A' }] })
```

Example Output:

```
[
  { "student_id": 4, "name": "Diana Prince", "age": 21, "grade":
"B", ... },
  { "student_id": 21, "name": "Uma Harris", "age": 21, "grade": "A",
... },
]
```

3. `$not`: Find students who do not have a grade of 'A'.

Query

```
> db.students.find({ grade: { $not: { $eq: 'A' } } })
```

Example Output:

```
[
  { "student_id": 10, "name": "Jack White", "age": 22, "grade": "B", ... },
  { "student_id": 12, "name": "Michael Brown", "age": 22, "grade": "B", ...
},
  ...
]
```

4. `$nor`: Find students who are neither 21 years old nor have a grade of 'B'.

Query

```
> db.students.find({ $nor: [{ age: 21 }, { grade: 'B' }] })
```

Example Output:

```
[
  { "student_id": 11, "name": "Anna Davis", "age": 23, "grade": "C",
... },
  { "student_id": 30, "name": "Daniel Thomas", "age": 22, "grade":
"A", ... },
  ...
]
```

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

Geospatial Selectors

Sample Data with Location

```
> db.students.insertMany( [

 {

   student_id: 51,

   name: "Yasmin Brooks",

   age: 24,

   location: { type: "Point", coordinates: [ -73.97, 40.77 ] }        // New York City coordinates

 },

 {
   student_id: 52,

   name: "Zoe Taylor",
   age: 22,

   location: { type: "Point", coordinates: [ -118.25, 34.05 ] }         // Los Angeles coordinates

 },

 {

   student_id: 53,

   name: "Liam Martinez",

   age: 23,

   location: { type: "Point", coordinates: [ -87.62, 41.88 ] } }])          // Chicago coordinates
```

Geospatial Query Example: Find Students Near a Location

```
> db.students.find({

  location: {

   $near: {

     $geometry: { type: "Point", coordinates: [ -73.97, 40.77 ] },       // New York City coordinates

     $maxDistance: 10000 // 10 km

   } } } )
```

Explanation:

- **`$near`**: Finds documents near the specified location.
- **`$geometry`**: Specifies the type of geometry and coordinates.
- **`$maxDistance`:** Limits the results to within a specified distance (in meters).

Example Output:

```
[
  { "student_id": 51, "name": "Yasmin Brooks", "age": 24,
"location": { "type": "Point", "coordinates": [ -73.97, 40.77 ] } }
]
```

Bitwise Selectors

Bitwise query operators allow you to query documents based on bitwise comparisons of integer values.

Sample Data with Bitwise Fields

```
> db.students.insertMany([
  {
     student_id: 54,
    name: "Eve Adams",
    age: 20,
    bitwiseFlag: 5                                  // Binary 101
  },
  {
    student_id: 55,
    name: "Sam Brown",
    age: 25,
    bitwiseFlag: 6                                   // Binary 110
    }  ] )
```

Bitwise Query Example: Find Documents Where Bitwise AND with a Mask is Non-zero

Command:

```
>  db.students.find( {
  bitwiseFlag: { $bitsAllSet: 1 }          // Find documents where the bitwise AND with 1 (binary 001) is non-zero
})
```

Explaination:

- **`$bitsAllSet`:** Matches documents where all the specified bit positions are set to 1

Example Output:

```
[
  { "student_id": 54, "name": "Eve Adams", "age": 20, "bitwiseFlag": 5 },
  { "student_id": 55, "name": "Sam Brown", "age": 25, "bitwiseFlag": 6 }
]
```

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

   MongoDB's projection operators: `$`, `$elemMatch`, and `$slice`. We'll use the `students` collection for these examples.

   > **`$` Projection Operator**

Example: Find a student and project the first course that matches a condition

Sample document:

```
> db.students.insertMany([{  "student_id": 60,

  "name": "Alice Johnson",

  "age": 23,

  "courses": [

    { "name": "Math", "grade": "A" },

    { "name": "Physics", "grade": "B" },

    { "name": "Chemistry", "grade": "A" }

  ] ]})
```

Query:

```
> db.students.find(
  { "student_id": 60, "courses.grade": "A" },
  { "courses.$": 1} )
```

Explanation:

- `{ "student_id": 60, "courses.grade": "A" }`: Find the student with `student_id` 60 and courses with grade "A".
-
- `{ "courses.$": 1, _id: 0 }`: Project only the first course that matches the grade "A", and exclude the `_id` field.

Example Output:

```
[
  { "courses": [{ "name": "Math", "grade": "A" }] }
]
```

> **$elemMatch** Projection Operator

The $elemMatch projection operator is used to project only the first array element that matches the specified $elemMatch condition.

Example: Find a student and project the course that matches a specific grade

Query:

```
> db.students.find(
  { "student_id": 60 },
  { "courses": { $elemMatch: { "grade": "A" } }}
)
```

Explanation:

- { "student_id": 60 }: Find the student with student_id 60.
- { "courses": { $elemMatch: { "grade": "A" } }, _id: 0 }: Project only the course element that matches the grade "A", and exclude the _id field.

Example Output:

```
[
  { "courses": { "name": "Math", "grade": "A" } }
]
```

> **$slice** Projection Operator

The $slice projection operator is used to return a subset of array elements. You can specify the number of elements to return, and optionally, the starting position.

Example: Find a student and project the first 2 courses

Query:

```
> db.students.find(
  { "student_id": 60 },
  { "courses": { $slice: 2 } })
```

Explanation:

- { "student_id": 60 }: Find the student with student_id 60.
- { "courses": { $slice: 2 }, _id: 0 }: Project only the first 2 elements of the courses array, and exclude the _id field.

Example Output:

```
[   { "courses": [
    { "name": "Math", "grade": "A" },
    { "name": "Physics", "grade": "B" }
  ]}]
```

Example: Find a student and project 2 courses starting from the second course

Query:

```
> db.students.find(
  { "student_id": 60 },
  { "courses": { $slice: [1, 2] } }
)
```

Explanation:

- `{ "student_id": 60 }`: Find the student with `student_id` 60.
- `{ "courses": { $slice: [1, 2] }, _id: 0 }`: Project 2 elements of the `courses` array starting from the second element, and exclude the `_id` field.

Example Output:

```
[
  { "courses": [
    { "name": "Physics", "grade": "B" },
    { "name": "Chemistry", "grade": "A" }
  ]}
]
```

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

Sample data:

```
> db.students.insertMany([

 { student_id: 1, name: "Alice Johnson", age: 23, grade: "A", scores: [85, 90, 95] },

 { student_id: 2, name: "Bob Smith", age: 22, grade: "B", scores: [75, 80, 85] },

 { student_id: 3, name: "Charlie Brown", age: 24, grade: "A", scores: [90, 95, 100] },

 { student_id: 4, name: "Diana Prince", age: 21, grade: "C", scores: [65, 70, 75] },

 { student_id: 5, name: "Eve Adams", age: 23, grade: "B", scores: [80, 85, 90] },

 { student_id: 6, name: "Frank White", age: 22, grade: "A", scores: [88, 92, 96] },

])
```

Aggregation Framework

Let's explore various aggregation operators like `$avg`, `$min`, `$max`, `$push`, and `$addToSet`.

1. **`$avg`**: Calculate the average age of students

Query

```
> db.students.aggregate([
  {
   $group: {
   _id: null,
   averageAge: { $avg: "$age" }
   }}
  ])
```

Explanation:

- `$group`: Groups the documents by the specified `_id` (null means all documents are in one group).
- `$avg`: Calculates the average age of students.

Example Output:

```
[
  { "_id": null, "averageAge": 22.5 }
]
```

2. **$min** and **$max** : Find the minimum and maximum age of students

Query

```
> db.students.aggregate([
  {
    $group: {
    _id: null,
    minAge: { $min: "$age" },
    maxAge: { $max: "$age" }
  }}
  ])
```

Explanation:

- **$min**: Finds the minimum age of students.
- **$max**: Finds the maximum age of students.

Example Output:

```
[
  { "_id": null, "minAge": 21, "maxAge": 24 }
]
```

3. **$push**: Collect all student names into an array

Query

```
> db.students.aggregate([
  {
   $group: {
   _id: null,
   allNames: { $push: "$name" }}
   }
  ])
```

Explanation:

- **$push**: Collects all student names into an array.

Example Output:

```
[
  { "_id": null, "allNames": ["Alice Johnson", "Bob Smith", "Charlie
Brown", "Diana Prince", "Eve Adams", "Frank White"] }
]
```

4. `$addToSet`: Collect unique grades into an array

Query

```
> db.students.aggregate([
  {
    $group: {
     _id: null,
     uniqueGrades: { $addToSet: "$grade" } }
   }
   ])
```

Explanation:

- **`$addToSet`**: Collects unique grades into an array.

Example Output:

```
[
  { "_id": null, "uniqueGrades": ["A", "B", "C"] }
]
```

5. `$sum`: Calculate the total number of students

Query

```
> db.students.aggregate([
  {
    $group: {
      _id: null,
      totalStudents: { $sum: 1 }}
   }
  ])
```

Explanation:

- **`$sum`**: Calculates the total number of students by summing 1 for each document.

Example Output:

```
[
  { "_id": null, "totalStudents": 6 }
]
```

6. `$avg` with nested fields: Calculate the average score for each student

Query

```
> db.students.aggregate([
  {
    $project: {
    name: 1,
    averageScore: { $avg: "$scores" } }}])
```

Explanation:

- **`$project`**: Projects the student name and calculates the average of the `scores` array for each student.

Example Output:

```
[
  { "name": "Alice Johnson", "averageScore": 90 },
  { "name": "Bob Smith", "averageScore": 80 },
  { "name": "Charlie Brown", "averageScore": 95 },
  { "name": "Diana Prince", "averageScore": 70 },
  { "name": "Eve Adams", "averageScore": 85 },
  { "name": "Frank White", "averageScore": 92 }
]
```

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

Sample Data

Ensure the **`students`** collection is populated with the following sample data:

```
> db.students.insertMany([

  { student_id: 1, name: "Alice Johnson", age: 23, grade: "A", scores: [85, 90, 95] },

  { student_id: 2, name: "Bob Smith", age: 22, grade: "B", scores: [75, 80, 85] },

  { student_id: 3, name: "Charlie Brown", age: 24, grade: "A", scores: [90, 95, 100] },

  { student_id: 4, name: "Diana Prince", age: 21, grade: "C", scores: [65, 70, 75] },

  { student_id: 5, name: "Eve Adams", age: 23, grade: "B", scores: [80, 85, 90] },

  { student_id: 6, name: "Frank White", age: 22, grade: "A", scores: [88, 92, 96] },

])
```

Example1 Pipeline: Calculate the average score for each student with grade "A", sort by average score, and return the top 2 students

<u>QUERY:</u>

```
> db.students.aggregate([

  {   $match: { grade: "A" }  },                              // Filter students with grade "A"
```

```
 {
   $project: {  name: 1,averageScore: { $avg: "$scores" }}     // Calculate the average score for each student
 },
 { $sort: { averageScore: -1 } },                               // Sort by average score in descending order
 {  $limit: 2 }                                                 // Limit the results to the top 2 students
  ])
```

Example Output:

```
[{ "name": "Charlie Brown", "averageScore": 95 },
 { "name": "Frank White", "averageScore": 92 }]
```

Example2 Pipeline: Group by grade, count the number of students in each grade, sort by count, skip the first group, and project the results

```
> db.students.aggregate([
 {
   $group: {                          // Group by grade and count the number of students in each grade
    _id: "$grade",
   studentCount: { $sum: 1 }}
 },
    { $sort: { studentCount: -1 }    },                // Sort by student count in descending order
    { $skip: 1 },                                      // Skip the first group
   {  $project: {                                      // Project the grade and student count fields
   _id: 0,grade: "$_id",studentCount: 1 } }
])
```

Example output:

```
[  { "grade": "B", "studentCount": 2 },
 { "grade": "C", "studentCount": 1 }   ]
```