EXPERIMENT – 8

Aim:

To implement CRUD operations in MongoDB using shell commands on a college database.

Description:

MongoDB is a NoSQL (non-relational) database that stores data in JSON format instead of tables (like SQL databases). It is highly scalable, flexible, and schema-less, making it suitable for handling large amounts of data in applications like e-commerce, social media, and IoT.

MongoDB Shell

MongoDB shell (mongosh) is a command-line interface (CLI) used to interact with MongoDB databases. It allows users to:

- Execute CRUD operations (Create, Read, Update, Delete).
- Run JavaScript-based queries on MongoDB.
- Manage databases and collections.
- Create indexes and execute aggregations for efficient data retrieval.

To start the MongoDB shell, use:

Mongosh

To enable the mongosh command in your system and connect MongoDB, follow these steps:

Step 1: Install MongoDB Community Edition

Windows:

- Download MongoDB from the official website: MongoDB Download Center.
- Install MongoDB, ensuring the MongoDB Database Tools and MongoDB Shell (mongosh) options are selected.
- During installation, check the option "Install MongoDB as a Service".
- After installation, MongoDB will be available as a system service.

Step 2: Verify MongoDB Installation

- To check if MongoDB is running, use:

mongod –version

Step 3: Install MongoDB Shell (mongosh)

- If mongosh is not recognized, install it separately from MongoDB Shell

Step 4: Test mongosh

- Open a new terminal and run:

  mongosh

- **If MongoDB is running, you will see a shell prompt like:**

  Current Mongosh Log ID: ...

  Connecting to: mongodb://localhost:27017

```
C:\Users\HP>mongosh
Current Mongosh Log ID: 67c677d487338fdd2fcc8987
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&ser
verSelectionTimeoutMS=2000&appName=mongosh+2.2.10
Using MongoDB:          7.0.12
Using Mongosh:          2.2.10
mongosh 2.4.0 is available for download: https://www.mongodb.com/try/downloa
d/shell

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

------
   The server generated these startup warnings when booting
   2025-02-23T20:30:36.999+05:30: Access control is not enabled for the data
base. Read and write access to data and configuration is unrestricted
------
```

CRUD Operations in MongoDB

CRUD stands for Create, Read, Update, and Delete – the basic operations to manage data in a database.

Operation	MongoDB Shell Command	Description
Create	db.students.insertOne({ name: "Alice", age: 21 })	Adds a new document to a collection
Read	db.students.find()	Retrieves data from a collection
Update	db.students.updateOne({ name: "Alice" }, { $set: { age: 22 } })	Modifies existing documents
Delete	db.students.deleteOne({ name: "Alice" })	Removes documents from a collection

Programs:

1. Create a new database as college database

- The **use college** command is used to create and switch to the college database. If it doesn't exist, it will be created automatically.

```
C:\Users\HP>mongosh
Current Mongosh Log ID: 67c677d487338fdd2fcc8987
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&ser
verSelectionTimeoutMS=2000&appName=mongosh+2.2.10
Using MongoDB:          7.0.12
Using Mongosh:          2.2.10
mongosh 2.4.0 is available for download: https://www.mongodb.com/try/downloa
d/shell

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

------
   The server generated these startup warnings when booting
   2025-02-23T20:30:36.999+05:30: Access control is not enabled for the data
base. Read and write access to data and configuration is unrestricted
------

test> use college
switched to db college
```

Fig 1: creating database

2. Create students and faculty collections under the college database

- Collections in MongoDB store documents. They are automatically created when data is inserted, but can also be created explicitly using **createCollection()**.

```
college> db.createCollection("students")
{ ok: 1 }
college> db.createCollection("faculty")
{ ok: 1 }
```

Fig 2: creating collections

3. Insert documents into students and faculty collections

- This command adds sample records (documents) into both collections.
- **insertMany()** is used to insert multiple documents at once
- **insertOne()** is used to insert a single document:

```
college> db.students.insertMany([
...     { _id: 1, name: "Alice", surname: "Johnson", marks: { math: 45, physphysics: 38, chemistry: 50 } },
...     { _id: 2, name: "Bob", surname: "Smith", marks: { math: 28, physics: 30, chemistry: 25 } },
...     { _id: 3, name: "Charlie", surname: "Brown", marks: { math: 80, physphysics: 75, chemistry: 85 } }
... ])
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3 } }
college>

college> db.faculty.insertMany([
...     { _id: 1, name: "Dr. Miller", surname: "Anderson", department: "Physics" },
...     { _id: 2, name: "Prof. Lee", surname: "Chang", department: "Mathematics" }
... ])
{ acknowledged: true, insertedIds: { '0': 1, '1': 2 } }
```

Fig 3: inserting documents

4. Show the collections data

- **find()** is used to retrieve and display all the documents stored in the collections.

```
college> db.students.find()
[
  {
    _id: 1,
    name: 'Alice',
    surname: 'Johnson',
    marks: { math: 45, physics: 38, chemistry: 50 }
  },
  {
    _id: 2,
    name: 'Bob',
    surname: 'Smith',
    marks: { math: 28, physics: 30, chemistry: 25 }
  },
  {
    _id: 3,
    name: 'Charlie',
    surname: 'Brown',
    marks: { math: 80, physics: 75, chemistry: 85 }
  }
]
college> db.faculty.find().pretty()
[
  {
    _id: 1,
    name: 'Dr. Miller',
    surname: 'Anderson',
    department: 'Physics'
  },
  {
    _id: 2,
    name: 'Prof. Lee',
    surname: 'Chang',
    department: 'Mathematics'
  }
]
```

Fig 4: displaying collections

5. Create your own indexes for the documents

- Indexing improves query performance. Here, we create indexes using **createIndex()** on the surname field.

```
college> db.studnets.createIndex({surname:1})
surname_1
college> db.faculty.createIndex({surname:1})
surname_1
```

Fig 5: creating indexes

6. Count documents in a collection

- Counts the number of documents present in a collection.

```
college> db.students.countDocuments()
3
college> db.faculty.countDocuments()
2
```

Fig 6: count documents

7. Find the first document in a collection

- **findOne()** retrieves the first document in a collection.

```
college> db.students.findOne()
{
  _id: 1,
  name: 'Alice',
  surname: 'Johnson',
  marks: { math: 45, physics: 38, chemistry: 50 }
}
college> db.faculty.findOne()
{
  _id: 1,
  name: 'Dr. Miller',
  surname: 'Anderson',
  department: 'Physics'
}
```

Fig 7: display first document

8. Find document by _id

- Searches for a document using the _id field, which uniquely identifies each record.

```
college> db.students.find({_id:3})
[
  {
    _id: 3,
    name: 'Charlie',
    surname: 'Brown',
    marks: { math: 80, physics: 75, chemistry: 85 }
  }
]
college> db.faculty.find({_id:2})
[
  {
    _id: 2,
    name: 'Prof. Lee',
    surname: 'Chang',
    department: 'Mathematics'
  }
]
```

Fig 8: finding with id

9. Find student and faculty by surname

- Fetches records where the surname matches a specific value.

```
college> db.students.find({surname:"Smith"})
[
  {
    _id: 2,
    name: 'Bob',
    surname: 'Smith',
    marks: { math: 28, physics: 30, chemistry: 25 }
  }
]
college> db.faculty.find({surname:"Anderson"})
[
  {
    _id: 1,
    name: 'Dr. Miller',
    surname: 'Anderson',
    department: 'Physics'
  }
]
```

Fig 9: finding with surname

10. Display the top 5 documents in the collection

- Limits the number of documents returned to 5 for better performance.

```
college> db.students.find().limit(5)
[
  {
    _id: 1,
    name: 'Alice',
    surname: 'Johnson',
    marks: { math: 45, physics: 38, chemistry: 50 }
  },
  {
    _id: 2,
    name: 'Bob',
    surname: 'Smith',
    marks: { math: 28, physics: 30, chemistry: 25 }
  },
  {
    _id: 3,
    name: 'Charlie',
    surname: 'Brown',
    marks: { math: 80, physics: 75, chemistry: 85 }
  },
  {
    _id: 4,
    name: 'David',
    surname: 'Williams',
    marks: { math: 55, physics: 60, chemistry: 65 }
  },
  {
    _id: 5,
    name: 'Emma',
    surname: 'Johnson',
    marks: { math: 70, physics: 75, chemistry: 80 }
  }
]
```

Fig 10: displaying top 5 documents

11. Find students who scored less than 40 marks in different subjects

- This query retrieves students whose marks in **any** subject are below 40 using the $lt (less than) and $or (logical OR) operators.

```
college> db.students.find({ $or: [ { "marks.math": { $lt: 40 } },
{ "marks.physics": { $lt: 40 } }, { "marks.chemistry": { $lt: 40 }
 }] })
[
  {
    _id: 1,
    name: 'Alice',
    surname: 'Johnson',
    marks: { math: 45, physics: 38, chemistry: 50 }
  },
  {
    _id: 2,
    name: 'Bob',
    surname: 'Smith',
    marks: { math: 28, physics: 30, chemistry: 25 }
  },
  {
    _id: 6,
    name: 'Frank',
    surname: 'Miller',
    marks: { math: 35, physics: 40, chemistry: 45 }
  },
  {
    _id: 7,
    name: 'Grace',
    surname: 'Davis',
    marks: { math: 25, physics: 30, chemistry: 20 }
  },
  {
    _id: 11,
    name: 'Katherine',
    surname: 'Lopez',
    marks: { math: 33, physics: 38, chemistry: 42 }
  }
]
```

Fig 11: displaying filtered documents

12. Display names of students based on their marks

- This query sorts students based on their **math marks in descending order** using $sort.

```
college> db.students.find({}, { name: 1, marks: 1 }).sort({ "marks.math": -1
 }).limit(5)
[
  {
    _id: 8,
    name: 'Henry',
    marks: { math: 85, physics: 90, chemistry: 88 }
  },
  {
    _id: 3,
    name: 'Charlie',
    marks: { math: 80, physics: 75, chemistry: 85 }
  },
  {
    _id: 5,
    name: 'Emma',
    marks: { math: 70, physics: 75, chemistry: 80 }
  },
  {
    _id: 10,
    name: 'Jack',
    marks: { math: 60, physics: 65, chemistry: 58 }
  },
  {
    _id: 4,
    name: 'David',
    marks: { math: 55, physics: 60, chemistry: 65 }
  }
]
```

Fig 11: displaying sorted documents

13. Update the name of a student

- The $set operator is used to modify the name field of a student whose _id is 2.

```
college> db.students.updateOne({_id:2},{$set:{name:"Robert"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Fig 13: update document

14. Delete documents in students' collection where marks are less than 35

- **deleteMany()** is used to remove multiple documents from collection and selectors are used to filter out students whose marks in any subject are below 35.

```
college> db.students.deleteMany({ $or: [ { "marks.math": { $lt: 35 } },
 { "marks.physics": { $lt: 35 } }, { "marks.chemistry": { $lt: 35 } }]
})
{ acknowledged: true, deletedCount: 3 }
```

Fig 14: deleting documents

15. Display the grades of students

- Aggregation in MongoDB processes data records and returns computed results, similar to SQL's GROUP BY.
- The $project stage modifies fields, and $cond assigns grades based on conditions.

```
college> db.students.aggregate([
...     {
...         $project: {
...             name: 1,
...             marks: 1,
...             grade: {
...                 $cond: { if: { $gte: [ "$marks.math", 75 ] }, then:
 "A", else: "B" }
...             }
...         }
...     }
... ]).pretty()
[
  {
    _id: 1,
    name: 'Alice',
    marks: { math: 45, physics: 38, chemistry: 50 },
    grade: 'B'
  },
  {
    _id: 3,
    name: 'Charlie',
    marks: { math: 80, physics: 75, chemistry: 85 },
    grade: 'A'
  },
  {
    _id: 4,
    name: 'David',
    marks: { math: 55, physics: 60, chemistry: 65 },
    grade: 'B'
  },
```

Fig 15: aggregating grades

16. Drop the faculty collection

- **drop()** deletes the entire faculty collection from the database.

```
college> db.faculty.drop()
true
```

Fig 16: dropping collection

17. Drop the college database

- dropDatabase() completely removes the college database, along with all its collections.

```
college> db.dropDatabase()
{ ok: 1, dropped: 'college' }
```

Fig 17: dropping database

18. Execute query selectors (comparison, logical operators) on any collection

Query operators are used to filter and retrieve specific documents based on conditions. They allow precise searches by applying different criteria to database queries.

1. Comparison Operators

These operators compare field values with specified conditions.

- $gt (greater than) → Finds values greater than a given number.
- $lt (less than) → Finds values less than a given number.
- $gte (greater than or equal to) → Finds values greater than or equal to a given number.
- $lte (less than or equal to) → Finds values less than or equal to a given number.
- $eq (equal to) → Matches exact values.
- $ne (not equal to) → Excludes specified values.

```
college> db.students.find({"marks.math":{$gte:50}})
[
  {
    _id: 3,
    name: 'Charlie',
    surname: 'Brown',
    marks: { math: 80, physics: 75, chemistry: 85 }
  },
  {
    _id: 4,
    name: 'David',
    surname: 'Williams',
    marks: { math: 55, physics: 60, chemistry: 65 }
  },
  {
    _id: 5,
    name: 'Emma',
    surname: 'Johnson',
    marks: { math: 70, physics: 75, chemistry: 80 }
  },
```

Fig 18: comparison selector

2. Logical Operators

Logical operators combine multiple conditions to refine search results.

- $and → Matches documents that meet all specified conditions.
- $or → Matches documents that meet at least one condition.
- $nor → Matches documents that do not satisfy any of the conditions.
- $not → Negates a condition, returning documents that do not match the specified criteria.

```
college> db.students.find({ $or: [ { "marks.math": { $lt: 40 } },
{ "marks.physics": { $lt: 40 } }, { "marks.chemistry": { $lt: 40 }
 }] })
[
  {
    _id: 1,
    name: 'Alice',
    surname: 'Johnson',
    marks: { math: 45, physics: 38, chemistry: 50 }
  },
  {
    _id: 2,
    name: 'Bob',
    surname: 'Smith',
    marks: { math: 28, physics: 30, chemistry: 25 }
  },
  {
    _id: 6,
    name: 'Frank',
    surname: 'Miller',
    marks: { math: 35, physics: 40, chemistry: 45 }
  },
  {
    _id: 7,
    name: 'Grace',
    surname: 'Davis',
    marks: { math: 25, physics: 30, chemistry: 20 }
  },
  {
    _id: 11,
    name: 'Katherine',
    surname: 'Lopez',
    marks: { math: 33, physics: 38, chemistry: 42 }
  }
]
```

Fig 19: logical selector

Result:

CRUD operations in MongoDB using shell commands on a college database are implemented successfully.