S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 5

Aim: Implement and Demonstrate No-SQL database operation: CRUD (create, read, update and delete) using MongoDB.

Name of Student: Shrutika Pradeep Bagdi

Roll No.: **CS22130**

Semester/Year: **7th / 4th**

Academic Session: 2025-2026

Date of Performance: _____

Date of Submission: _____

AIM: Implement and Demonstrate No-SQL database operation: CRUD (create, read, update and delete) using MongoDB.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- In a number of applications, the primary objective of the application is to allow a user to perform create, read, update, and delete (CRUD) operations on the underlying data
- MongoDB provides the different ways to perform aggregation operations on the data like aggregation pipeline, map-reduce or single objective aggregation commands.
- MongoDB can store any type of file which can be any size without effecting our stack.

HARDWARE AND SOFTWARE REQUIRMENTS:

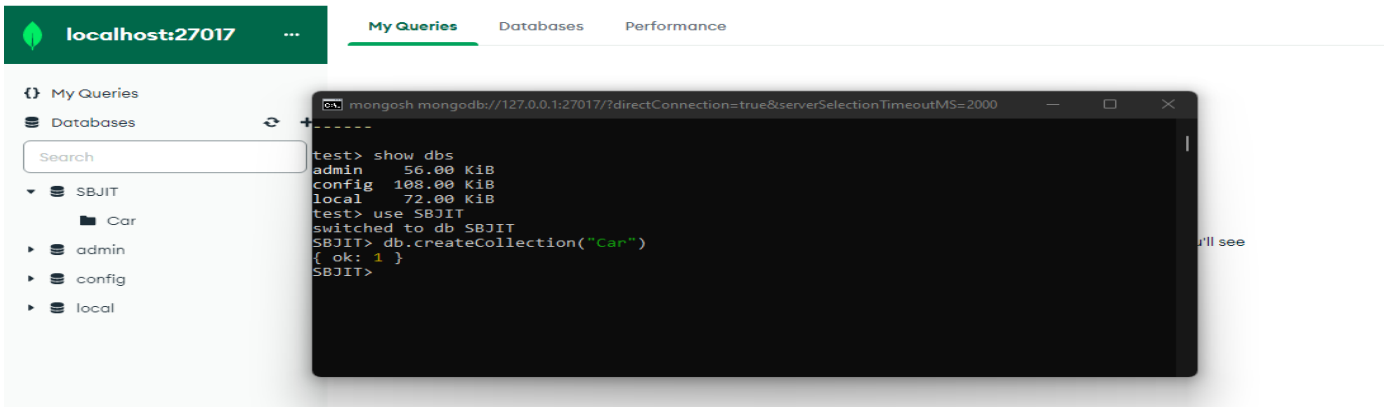
Hardware Requirement: High Configuration computer

Software Requirement: MongoDB-8.0

THEORY:

Mongodb is a document-oriented database program widely classified as a NoSQL database program.

In MongoDB, the CRUD operation refers to the creating, reading, updating, and deleting documents. Here is an explanation of the operations in detail:



1)C--→ Create

Create (or insert) operations add new documents to a collection. There are two ways to add new documents to a collection:

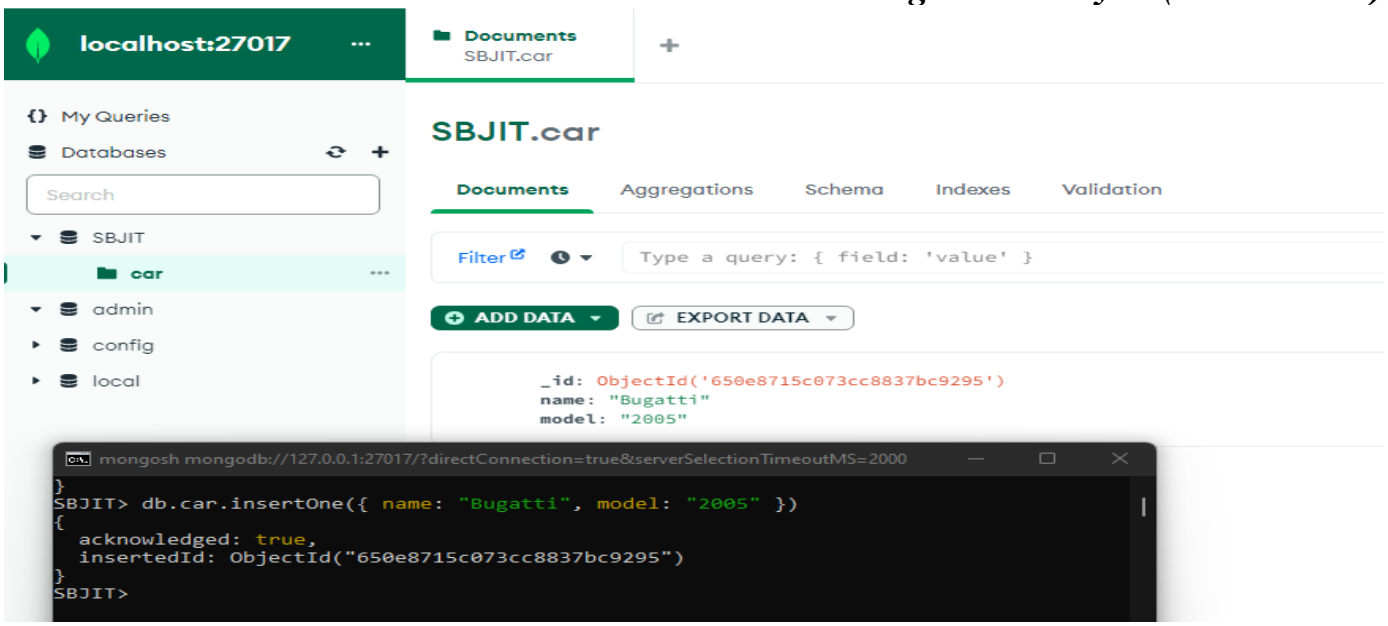
- db.collection.insertOne():

insertOne() operation allows us to create individual documents in a collection.

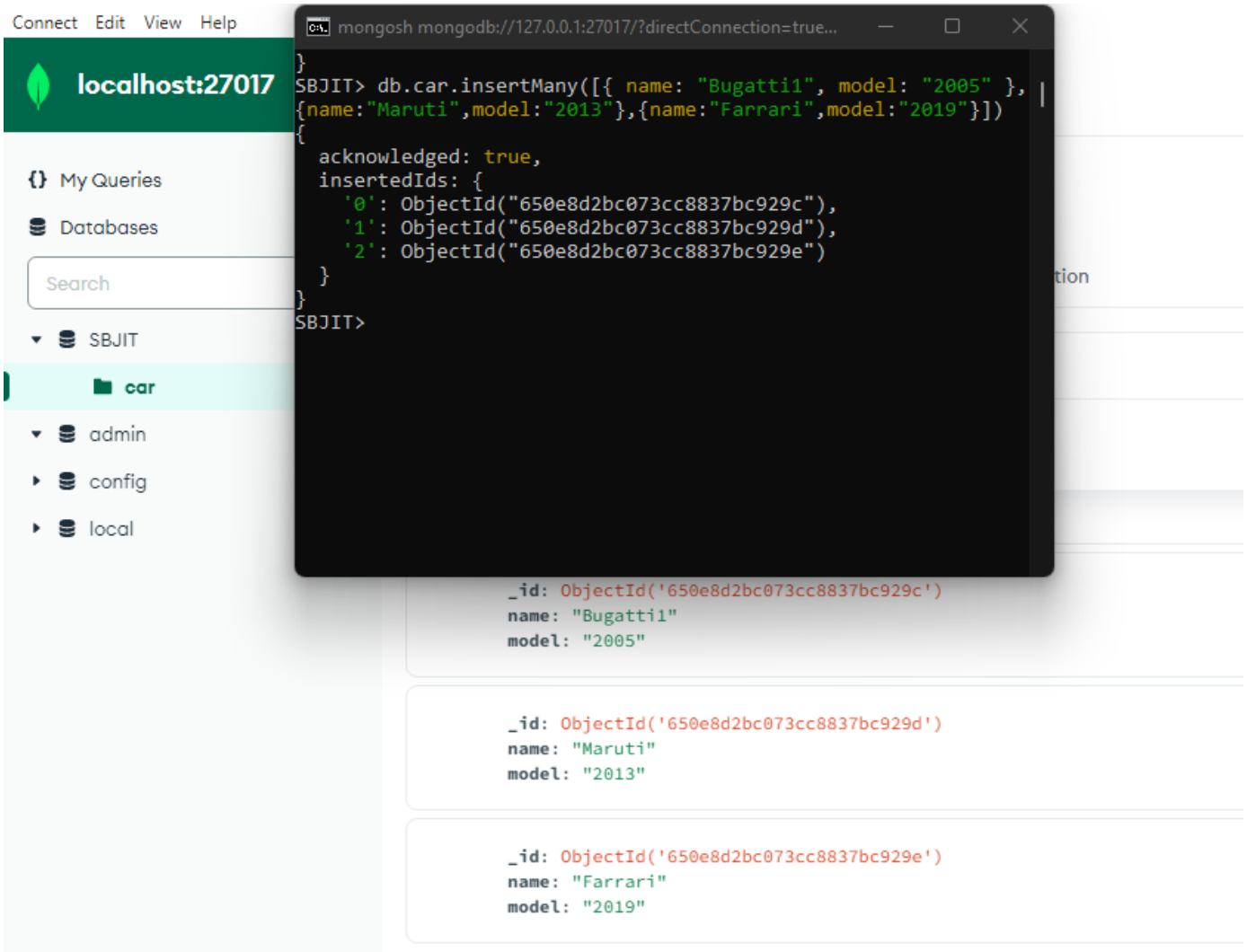
- db.collection.insertMany():

insertMany() operation is used to create multiple documents in a single operation.

Here is an example of how we can add a single car to the cars collection using the insertOne() operation:



Add info of multiple cars to the cars collection with a single operation using insertMany().



2) R--→ Read

Read operations retrieve documents from a collection. Here is the method in Mongodb to retrieve information:

- db.collection.find()

find() operation will return everything from a collection if you call it without any parameters. On the other hand, we can specify any filter or criteria to retrieve information from a collection using:

- db.collection.find(query)

Example

Here is an example of how we can read information about all cars from the cars collection:

```
SBJIT> db.car.find()
[
  {
    _id: ObjectId("650e8cc8c073cc8837bc9299"),
    name: 'Bugatti1',
    model: '2005'
  },
  {
    _id: ObjectId("650e8cc8c073cc8837bc929a"),
    name: 'Maruti',
    model: '2013'
  },
  {
    _id: ObjectId("650e8cc8c073cc8837bc929b"),
    name: 'Farrari',
    model: '2019'
```

Example

Now we will see how we can read information about those cars from the cars collection whose model is 2005:

```
SBJIT> db.car.find({"model":"2005"})
[
  {
    _id: ObjectId("650e8d2bc073cc8837bc929c"),
    name: 'Bugatti1',
    model: '2005'
  }
]
SBJIT>
```

3) U--→ Update

Update operations modify existing documents in a collection. There are three ways to update documents of a collection:

- **db.collection.updateOne()**

Updates one field in the document where the given criteria or filter meets the condition. Updating a field will not remove the old field instead a new field will be added to the document.

```
SBJIT> db.car.updateOne({_id:ObjectId("650e8d2bc073cc8837bc929c")},{$set:{model:'2050'}})
{BJIT> db.car.updateOne({"_id":ObjectId("650e8d2bc073cc8837bc929c")},{$set:{model:'2050'}})
  acknowledged: true,
  insertedId: null,cted token, expected "," (1:81)
  matchedCount: 1,
  modifiedCount: 1,One({"_id":ObjectId("650e8d2bc073cc8837bc929c")},{$set:{model:'2050'}})
  upsertedCount: 0                                                              ^
} 2 |
SBJIT> db.car.find({"_id":ObjectId("650e8d2bc073cc8837bc929c")})
[BJIT>
  {
    _id: ObjectId("650e8d2bc073cc8837bc929c"),
    name: 'Bugatti1',
    model: '2050'
  }
]
```

Execute the following find() method to see the updated data.

```
SBJIT> db.car.updateOne({model:"2050"},{$set:{name:'Aniket'}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
SBJIT> db.car.find({"_id":ObjectId("650e8d2bc073cc8837bc929c")})
[
  {
    _id: ObjectId("650e8d2bc073cc8837bc929c"),
    name: 'Aniket',
    model: '2050'
  }
]
SBJIT>
```

- **db.collection.updateMany()**

Updates all fields in the document where the given criteria or filter meets the condition.

```
updateMany({"_id" : ObjectId("1")}, {$set: { "name" : "NewName"}, $set: { "new_model" : 2020
}})
```

- **db.collection.replaceOne()**

Replace the entire document. It will replace the old fields and values with new ones.

db.collection.replaceOne(<filter>, <replacementDocument>,

{

upsert: <boolean>,

```
    writeConcern: <document>,
    collation: <document>,
    hint: <document|string>
  }
)
```

<filter>: This is a document that specifies the filter criteria to identify the document you want to replace. It determines which document(s) will be replaced. Only the first document that matches the filter will be replaced.

<replacement>: This is the new document that will replace the existing document. It should be a complete document with the same _id value as the document you want to replace. If the _id field is omitted in the replacement document, a new _id will be generated. The replacement document should have the same structure as the existing document, or you can modify it as needed.

upsert (optional): A boolean flag that specifies whether to perform an upsert operation if the document specified in the filter is not found. If upsert is set to true, and no matching document is found, a new document will be inserted based on the replacement document.

writeConcern (optional): A document that specifies the write concern for the operation, including options such as w (the number of nodes that must acknowledge the write), j (wait for the write to be journaled), and wtimeout (a timeout for the write operation).

```
> db.employee.replaceOne({},
... { name: "Anu", age: 30, branch: "EEE", department: "HR", joiningYear: 2018})

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.employee.find().pretty()
{
        "_id" : ObjectId("5e4138d692e6dfa3fc48dd6f"),
        "name" : "Anu",
        "age" : 30,
        "branch" : "EEE",
        "department" : "HR",
        "joiningYear" : 2018
}
{
        "_id" : ObjectId("5e4138d692e6dfa3fc48dd70"),
        "name" : "Mohit",
        "age" : 26,
        "branch" : "ECE",
        "department" : "HR"
}
{
        "_id" : ObjectId("5e4138d692e6dfa3fc48dd71"),
        "name" : "Sonu",
        "age" : 25,
        "branch" : "CSE",
        "department" : "Development"
}
```

Replacing single document that matches the filter:

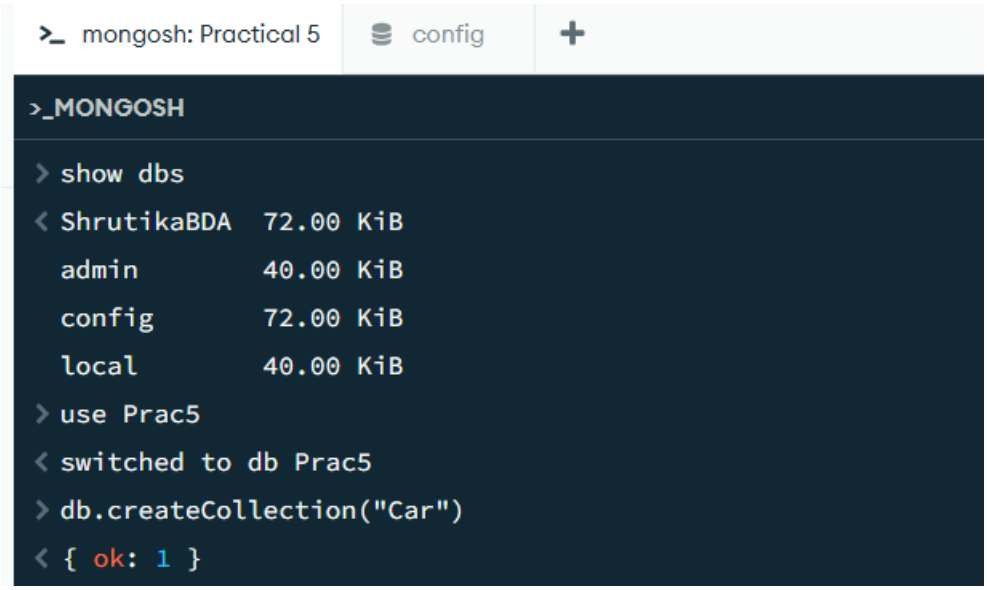
```
> db.employee.replaceOne({name: "Sonu"}, {name: "Sonu", age: 25, branch: "CSE", department: "Designing"})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.employee.find().pretty()
{
        "_id" : ObjectId("5e4138d692e6dfa3fc48dd6f"),
        "name" : "Anu",
        "age" : 30,
        "branch" : "EEE",
        "department" : "HR",
        "joiningYear" : 2018
}
{
        "_id" : ObjectId("5e4138d692e6dfa3fc48dd70"),
        "name" : "Mohit",
        "age" : 26,
        "branch" : "ECE",
        "department" : "HR"
}
{
        "_id" : ObjectId("5e4138d692e6dfa3fc48dd71"),
        "name" : "Sonu",
        "age" : 25,
        "branch" : "CSE",
        "department" : "Designing"
}
>
```

4) D--→ Delete

Here are the most popular read methods in MongoDB.

- db.collection.deleteOne() – Deletes a single document in the collection.
- db.collection.deleteMany() – Deletes multiple document in the collection.
- db.collection.remove() – Removes older documents from the collection.

INPUT / OUTPUT (SCREENSHOTS):



```
> db.Car.insertOne({ name: "Bugatti", model: "2005" })
< {
    acknowledged: true,
    insertedId: ObjectId('68c3e0cf2d2fcdeb3dc36869')
  }
```

Type a query: { field: 'value' } or Generate query

ADD DATA | EXPORT DATA | UPDATE | DELETE

```
_id: ObjectId('68c3e0cf2d2fcdeb3dc36869')
name : "Bugatti"
model : "2005"
```

```
> db.Car.insertMany([{ name: "Bugatti1", model: "2005"}, {name:"Maruti", model:"2013"}, {name:"Farrari", model:"2019"}])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68c3e27e2d2fcdeb3dc3686b'),
      '1': ObjectId('68c3e27e2d2fcdeb3dc3686c'),
      '2': ObjectId('68c3e27e2d2fcdeb3dc3686d')
    }
  }
Prac5>
```

ADD DATA | EXPORT DATA | UPDATE | DELETE

```
_id: ObjectId('68c3e0cf2d2fcdeb3dc36869')
name : "Bugatti"
model : "2005"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686b')
name : "Bugatti1"
model : "2005"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686c')
name : "Maruti"
model : "2013"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686d')
name : "Farrari"
model : "2019"
```

```
> db.Car.find()
< {
    _id: ObjectId('68c3e0cf2d2fcdeb3dc36869'),
    name: 'Bugatti',
    model: '2005'
  }
  {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686b'),
    name: 'Bugatti1',
    model: '2005'
  }
  {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686c'),
    name: 'Maruti',
    model: '2013'
  }
  {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686d'),
    name: 'Farrari',
    model: '2019'
  }
Prac5 >
```

```
> db.Car.find({"model":"2005"})
< {
    _id: ObjectId('68c3e0cf2d2fcdeb3dc36869'),
    name: 'Bugatti',
    model: '2005'
  }
  {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686b'),
    name: 'Bugatti1',
    model: '2005'
  }
Prac5 >
```

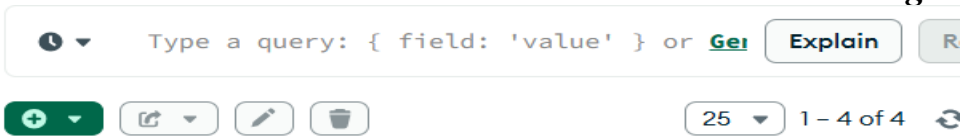
```
> db.Car.updateOne({"_id":ObjectId("68c3e27e2d2fcdeb3dc3686b")},{$set:{model:'2050'}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
Prac5 >
```

```
> db.Car.find({"_id":ObjectId("68c3e27e2d2fcdeb3dc3686b")})
< {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686b'),
    name: 'Bugatti1',
    model: '2050'
  }
Prac5 >
```

```
> db.Car.updateOne({model:"2050"},{$set:{name:'Shrutika'}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
Prac5 >
```

```
> db.Car.find({"_id":ObjectId("68c3e27e2d2fcdeb3dc3686b")})
< {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686b'),
    name: 'Shrutika',
    model: '2050'
  }
Prac5 >
```

```
> db.Car.deleteOne({ name: "Shrutika "})
< {
    acknowledged: true,
    deletedCount: 0
  }
Prac5 >
```




```
_id: ObjectId('68c3e0cf2d2fcdeb3dc36869')
name : "Bugatti"
model : "2005"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686b')
name : "Bugatti1"
model : "2005"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686c')
name : "Maruti"
model : "2013"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686d')
name : "Farrari"
model : "2019"
```

```
> db.Car.deleteMany({ model: "2005" })
< {
    acknowledged: true,
    deletedCount: 1
  }
Prac5>
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686b')
name : "Shrutika"
model : "2050"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686c')
name : "Maruti"
model : "2013"
```

```
_id: ObjectId('68c3e27e2d2fcdeb3dc3686d')
name : "Farrari"
model : "2019"
```

```
> db.Car.find()
< {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686b'),
    name: 'Shrutika',
    model: '2050'
  }
  {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686c'),
    name: 'Maruti',
    model: '2013'
  }
  {
    _id: ObjectId('68c3e27e2d2fcdeb3dc3686d'),
    name: 'Farrari',
    model: '2019'
  }
Prac5>
```

```
> db.Car.deleteMany({})
< {
    acknowledged: true,
    deletedCount: 3
  }
Prac5>
```

```
> db.Car.find()
<
Prac5>
```

CONCLUSION:

CRUD operations are at the core of database interactions in MongoDB, and they are used in various applications, from simple data storage to complex web applications and analytics platforms. Understanding how to perform these operations and their associated concepts is essential for working effectively with MongoDB databases. Additionally, developers should consider data modeling, indexing, and query optimization techniques to maximize the efficiency and performance of CRUD operations in MongoDB.

DISCUSSION AND VIVA VOCE:

- Explain the $set operator in the context of updating documents. How is it used?
- Explain the difference between find() and findOne() methods in MongoDB.

- What is the significance of the **multi** option when updating documents using updateMany()?
- What is the impact of deleting documents on the size of a MongoDB collection and its indexes?
- How can you increment or decrement a specific field in a document using the **$inc** operator during an update operation?

REFERENCE:

- https://blog.sqlauthority.com/2020/05/22/mongodb-fundamentals-crud-deleting-objects-day-5-of-6/
- https://www.educative.io/answers/moperations?utm_campaign=brand_educative&utm_source=google&utm_medium=ppc&utm_content=performance_max_india&eid=5082902844932096&utm_term=&utm_campaign=Cj0KCQjw9rSoBhCiARIsAFOipln7xt0eaYuBMHeHVs58wQUc5xbMNbpI6VM_NUBquZgz8taUH1CErVwaAlFrEALw_wcB.
- https://www.youtube.com/watch?v=oSIv-E60NiU

Observation book: (3)	Viva-Voce (3)	Quality of Submission and timely Evaluation (4)
Total:		**Sign with date:**