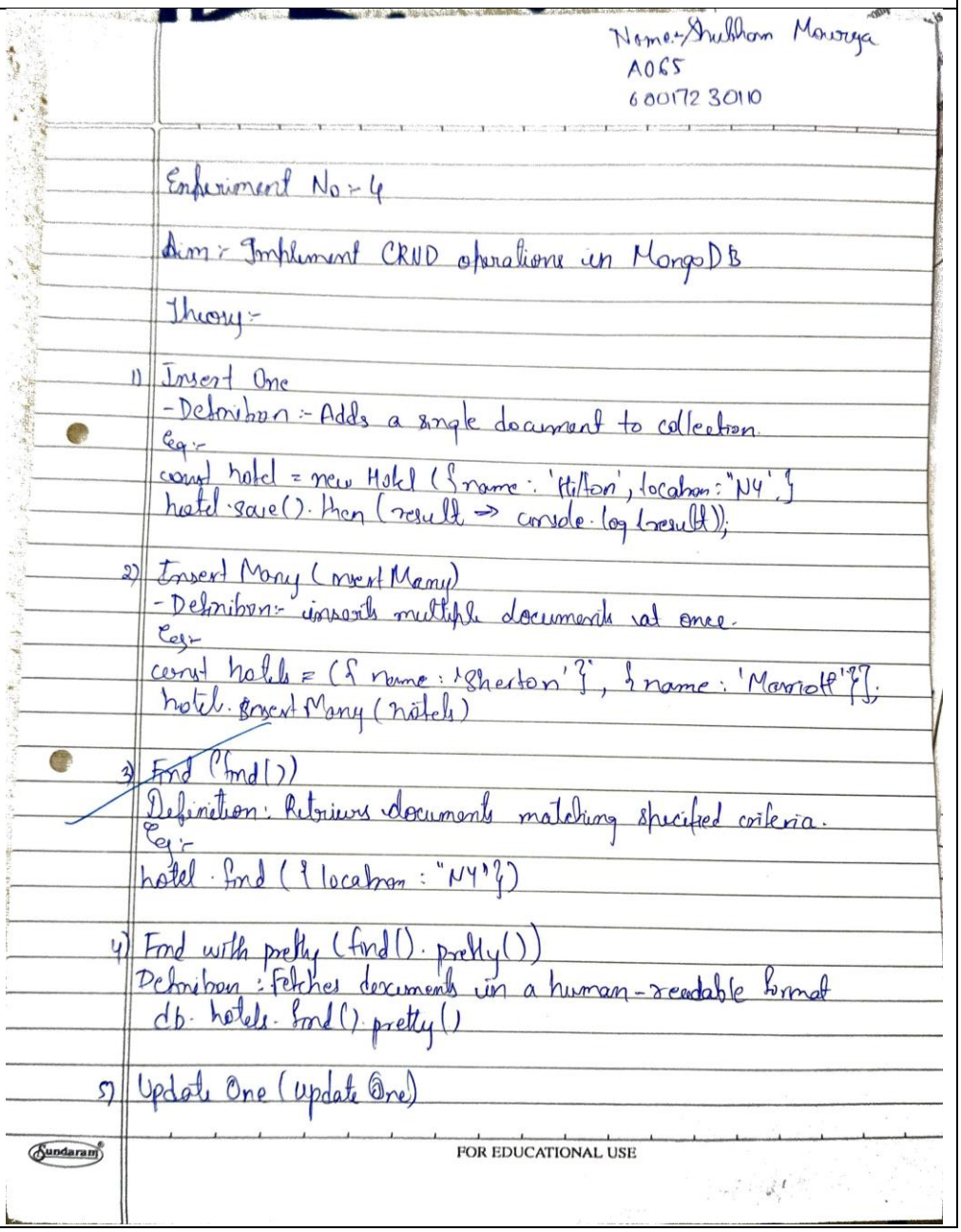
Department of Artificial Intelligence and Machine Learning
B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

Experiment 4

Name: Shubham Mourya **SAP ID: 60017230110**

Date:	**26 – 10 - 2024**
Aim	**Implementing CRUD Operations in MongoDB**
Software	MongoDB
Pre-requisite	Active internet connection
Theory	

Name:- Shubham Mourya
A065
60017230110

Experiment No:- 4

Aim:- Implement CRUD operations in MongoDB

Theory:-

1) Insert One
- Definition:- Adds a single document to collection
Eg:-
```
const hotel = new Hotel ({ name: 'Hilton', location: "N4", })
hotel.save().then (result => console.log (result));
```

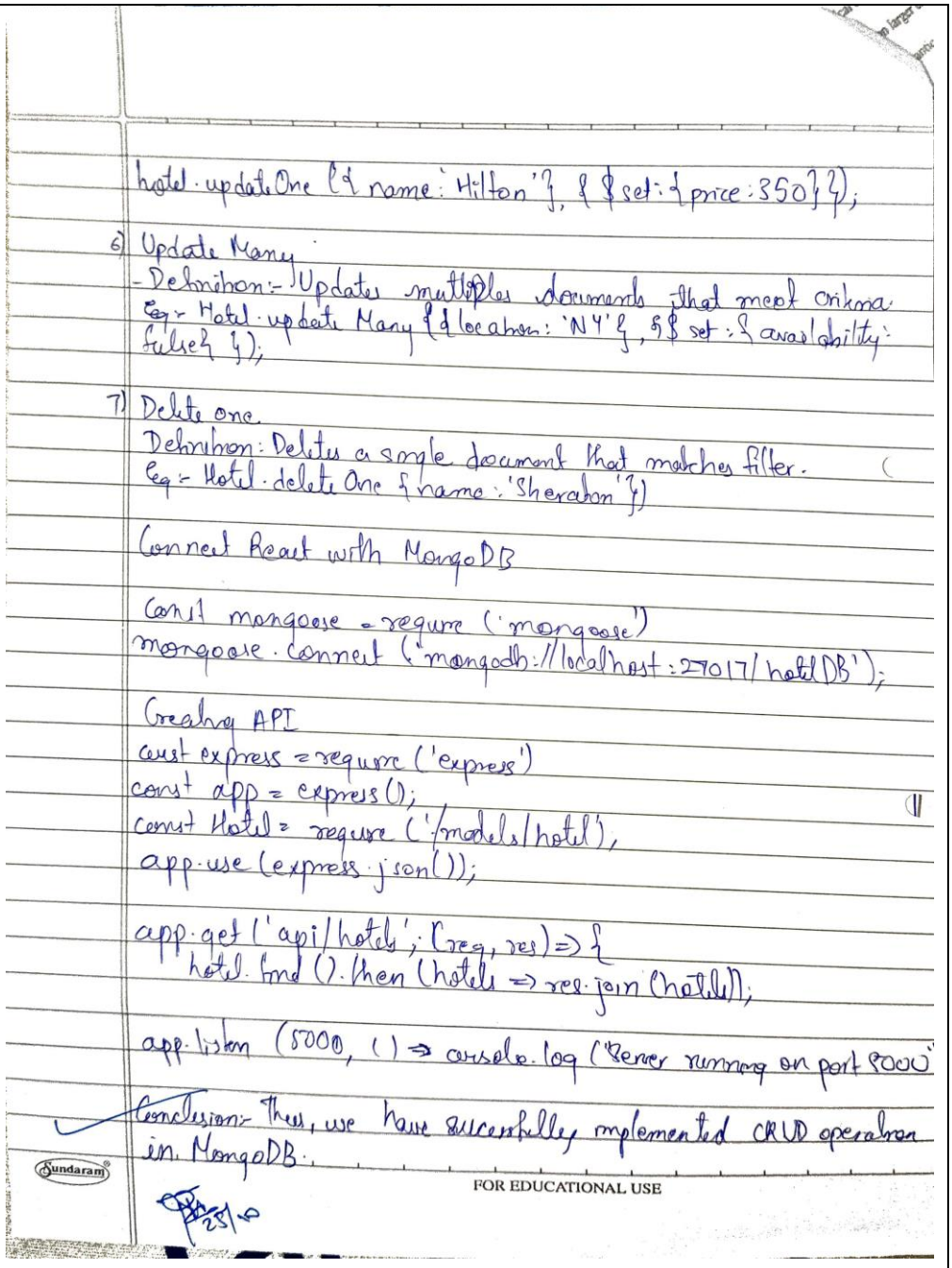
2) Insert Many (insertMany)
- Definition:- inserts multiple documents at once.
Eg:-
```
const hotels = ({ name: 'Sheraton'}, { name: 'Marriott'});
hotel.insertMany (hotels)
```

3) ~~Find~~ (find())
Definition: Retrieves documents matching specified criteria.
Eg:-
```
hotel.find ({ location: "N4"})
```

4) Find with pretty (find().pretty())
Definition: Fetches documents in a human-readable format
```
db.hotels.find().pretty()
```

5) Update One (updateOne)

Sundaram
FOR EDUCATIONAL USE



```
hotel.updateOne({name:'Hilton'},{$set:{price:350}});
```

6) Update Many
- Definition:- Updates multiples documents that meet criteria.
Eg:- Hotel.updateMany({location:'NY'},{$set:{availability: false}});

7) Delete one.
Definition: Deletes a single document that matches filter.
Eg:- Hotel.deleteOne({name:'Sheraton'})

Connect React with MongoDB

```
const mongoose = require('mongoose')
mongoose.connect('mongodb://localhost:27017/hotelDB');
```

Creating API

```
const express = require('express')
const app = express();
const Hotel = require('./models/hotel');
app.use(express.json());

app.get('/api/hotels',(req,res)=>{
  hotel.find().then(hotels => res.json(hotels));

app.listen(5000, () => console.log('Server running on port 5000'
```

Conclusion:- Thus, we have successfully implemented CRUD operation in MongoDB.

Sundaram

FOR EDUCATIONAL USE

25/10

1. Create (Insert)

- **Purpose**: The `Create` operation inserts new documents (records) into a MongoDB collection (equivalent to a table in SQL databases).
- **How it works**: MongoDB stores data in BSON (Binary JSON) format, allowing it to store nested structures and complex data types. Each document is a JSON-like object with key-value pairs, and they are stored in collections. A unique identifier (`_id`) is automatically added if it's not provided, making it easy to retrieve each document individually.
- **Use Cases**: Storing new user profiles, adding transactions, logging events, etc.

2. Read (Query)

- **Purpose**: The `Read` operation fetches data from a MongoDB collection based on specific criteria.
- **How it works**: MongoDB queries use a JSON-like syntax, where you can specify key-value pairs to filter documents. Queries can be simple or complex, allowing conditions, ranges, regular expressions, and projections (to select specific fields to return). You can use the `findOne` method to retrieve a single document or `find` to retrieve multiple documents.
- **Use Cases**: Fetching user data by ID, retrieving all posts by a specific author, filtering records within a date range, etc.

3. Update

- **Purpose**: The `Update` operation modifies existing documents within a collection.
- **How it works**: Updates can either replace an entire document or modify specific fields. MongoDB uses update operators, like `$set`, `$inc`, and `$rename`, which allow partial updates without needing to replace the entire document. You can update a single document with `updateOne` or multiple documents with `updateMany`.
- **Use Cases**: Updating user profile details, incrementing counters, marking notifications as read, and updating multiple records at once for bulk modifications.

4. Delete

- **Purpose**: The `Delete` operation removes documents from a collection.
- **How it works**: Deleting can target a single document (`deleteOne`) or multiple documents (`deleteMany`). A filter condition is specified, and MongoDB deletes documents that match the criteria. This operation is irreversible, so it's essential to carefully construct the filter criteria.
- **Use Cases**: Removing outdated records, deleting a specific user's data, clearing old logs or temporary data, etc.

MongoDB's Advantages with CRUD Operations

MongoDB's flexibility with CRUD operations comes from several factors:

- **Schema Flexibility**: Unlike SQL databases, MongoDB doesn't require a fixed schema, so each document can have a different structure.
- **Scalability**: MongoDB is designed to scale horizontally, meaning it can handle high data volumes and high read/write loads.
- **Embedded Data**: Documents can have nested structures, allowing related data to be stored in a single document rather than in separate tables.
- **Powerful Query Language**: MongoDB's query language supports filtering, projections, aggregations, and joins (through the `$lookup` operator), enabling complex data manipulation and retrieval.

CRUD Operations in Use

These CRUD operations form the core of any database interaction, allowing applications to manage and manipulate data efficiently. MongoDB's design makes it especially suitable for dynamic, data-rich applications that benefit from the flexibility of a document-oriented approach.

In summary, MongoDB's CRUD operations support a wide range of use cases with flexibility and scalability, making it a popular choice for applications requiring complex, nested data handling or high-speed data access.

Result

```
collection> db.students.insertOne({ 'name': 'David', 'age': 24 });
{
  acknowledged: true,
  insertedId: ObjectId('671741d3c8b4b13da986b01e')
}
```

```
collection> db.students.insertMany([{ 'name': 'David', 'age': 24 }, {'name': 'Max', 'age': 21}]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67174212c8b4b13da986b01f'),
    '1': ObjectId('67174212c8b4b13da986b020')
  }
}
```

```
collection> db.students.find().pretty()
[
  { _id: ObjectId('671741a7c8b4b13da986b01d'), name: 'John', age: 26 },
  { _id: ObjectId('671741d3c8b4b13da986b01e'), name: 'David', age: 28 },
  { _id: ObjectId('67174212c8b4b13da986b01f'), name: 'David', age: 24 },
  { _id: ObjectId('67174212c8b4b13da986b020'), name: 'Max', age: 21 },
  { _id: ObjectId('6717423ac8b4b13da986b021'), name: 'Blake', age: 23 },
  {
    _id: ObjectId('6717423ac8b4b13da986b022'),
    name: 'Christen',
    age: 19
  },
  { _id: ObjectId('67174257c8b4b13da986b023'), name: 'Aaron', age: 20 },
  { _id: ObjectId('67174257c8b4b13da986b024'), name: 'Ray', age: 22 }
]
```

```
collection> db.students.update({ 'name': 'David'}, {$set: {'age': 28}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
collection> db.students.remove({name: 'David'})
{ acknowledged: true, deletedCount: 2 }
collection> db.students.find()
[
  { _id: ObjectId('671741a7c8b4b13da986b01d'), name: 'John', age: 26 },
  { _id: ObjectId('67174212c8b4b13da986b020'), name: 'Max', age: 21 },
  { _id: ObjectId('6717423ac8b4b13da986b021'), name: 'Blake', age: 23 },
  {
    _id: ObjectId('6717423ac8b4b13da986b022'),
    name: 'Christen',
    age: 19
  },
  { _id: ObjectId('67174257c8b4b13da986b023'), name: 'Aaron', age: 20 },
  { _id: ObjectId('67174257c8b4b13da986b024'), name: 'Ray', age: 22 }
]
```

```
collection> db.students.insert([{name: 'Shrey', age: 19}, {name: 'Yash', age: 20}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67174accc8b4b13da986b02a'),
    '1': ObjectId('67174accc8b4b13da986b02b')
  }
}
collection> db.students.remove({$or :[{name: 'Shrey'}, {name: 'Yash'}]})
```

```
collection> db.students.insert([{name: 'Shrey', age: 19}, {name: 'Yash', age: 20}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67174aecc8b4b13da986b02c'),
    '1': ObjectId('67174aecc8b4b13da986b02d')
  }
}
collection> db.students.remove({$and :[{name: 'Shrey'}, {age: 19}]})
{ acknowledged: true, deletedCount: 1 }
```

Conclusion	Thus, we have understood and implemented the CRUD operations in MongoDB.