PRACTICAL NO :- 1

MONGO DB BASICS

MongoDB is a popular NoSQL database designed for flexibility, scalability, and performance. Here are some of the basic concepts and theories behind MongoDB:

1. Document-Oriented Storage
Documents: MongoDB stores data in JSON-like format (BSON), allowing for complex data structures. Each document can have different fields and structures.
Collections: Documents are grouped into collections, which are akin to tables in relational databases but without a fixed schema.
2. Schema Flexibility
Unlike traditional databases, MongoDB allows for dynamic schemas. This means you can add fields to documents without affecting other documents in the same collection.
3. Data Types
MongoDB supports various data types, including strings, numbers, arrays, objects, and dates, giving flexibility in how data is stored.
4. CRUD Operations
Basic operations include:
Create: Inserting new documents.
Read: Querying documents using various criteria.
Update: Modifying existing documents.
Delete: Removing documents.
5. Indexes
MongoDB supports indexing to improve query performance. Indexes can be created on any field, and compound indexes can also be formed for multi-field searches.
6. Aggregation Framework
MongoDB provides powerful tools for data aggregation, enabling operations like filtering, grouping, and transforming data to analyze it more effectively.
7. Replication and Sharding
Replication: MongoDB supports replication for high availability. A replica set is a group of MongoDB servers that maintain the same data set.
Sharding: For horizontal scalability, MongoDB can distribute data across multiple servers or clusters, balancing the load.
8. Drivers and APIs
MongoDB provides drivers for various programming languages, allowing applications to interact with the database seamlessly.

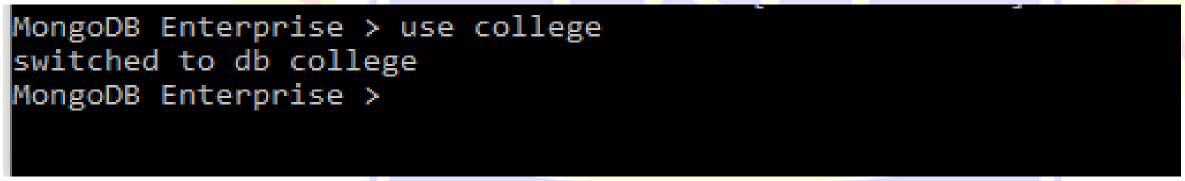
9. Use Cases
MongoDB is commonly used in scenarios requiring fast development, high availability, and scalable architecture, such as real-time analytics, content management systems, and mobile applications.
10. Security Features
MongoDB offers various security features, including authentication, authorization, and encryption to protect data.

Pratical 1a. Write a MongoDB query to create and drop database.
Syntax:-

```
use DATABASE_NAME
```

Query:-

```
>use college
```



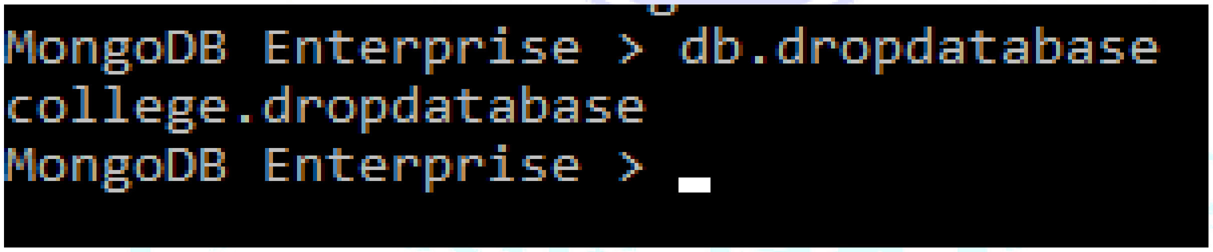
```
MongoDB Enterprise > use college
switched to db college
MongoDB Enterprise >
```

Syntax:-

```
db.dropDatabase()
```

Query:-

```
> db.dropDatabase()
```



```
MongoDB Enterprise > db.dropdatabase
college.dropdatabase
MongoDB Enterprise > _
```

> db.dropDatabase()

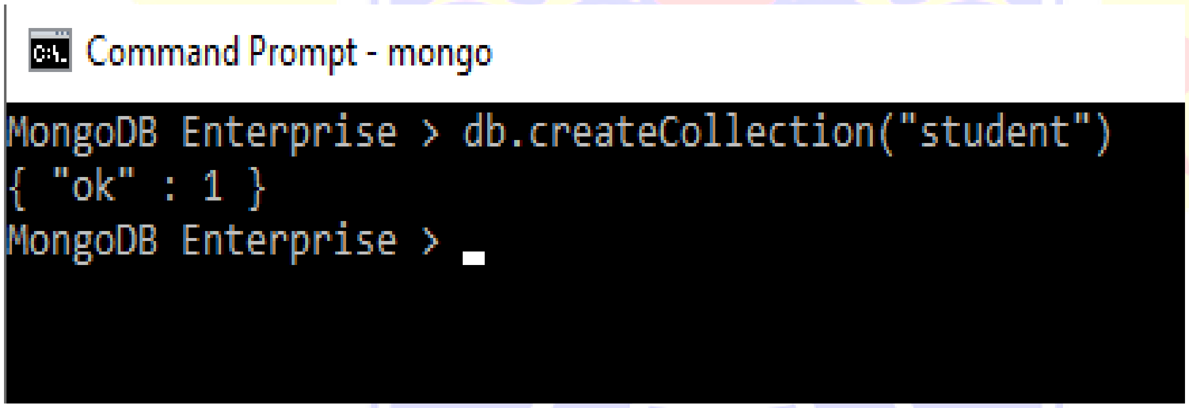
Pratical 1b. Write a MongoDB query to create, display and drop collection

Syntax:-

db.createCollection(name, options)

Query:-

>db.createCollection(“student”)



Command Prompt - mongo

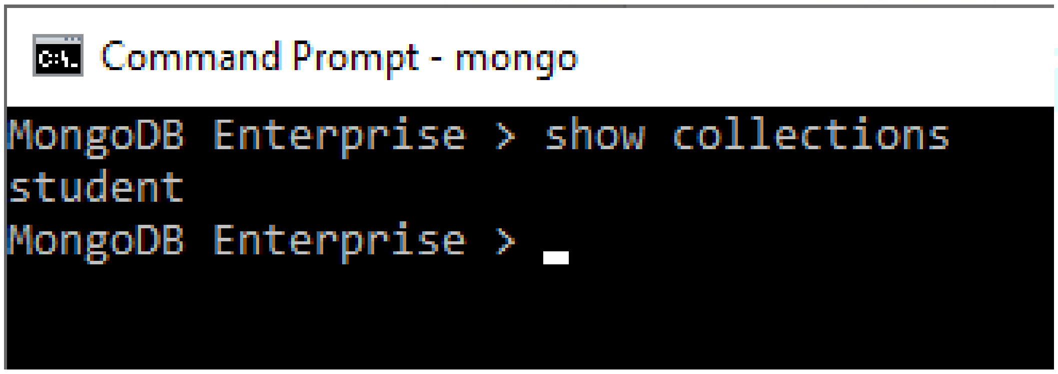
```
MongoDB Enterprise > db.createCollection("student")
{ "ok" : 1 }
MongoDB Enterprise > _
```

Syntax:-

show collections

Query:-

>show collections



Command Prompt - mongo

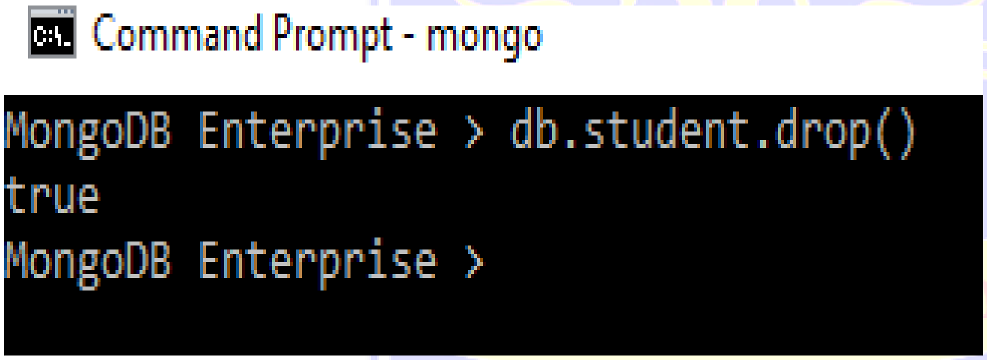
```
MongoDB Enterprise > show collections
student
MongoDB Enterprise > _
```

Syntax:-

```
db.COLLECTION_NAME.drop()
```

Query:-

```
> db.student.drop()
```



```
MongoDB Enterprise > db.student.drop()
true
MongoDB Enterprise >
```

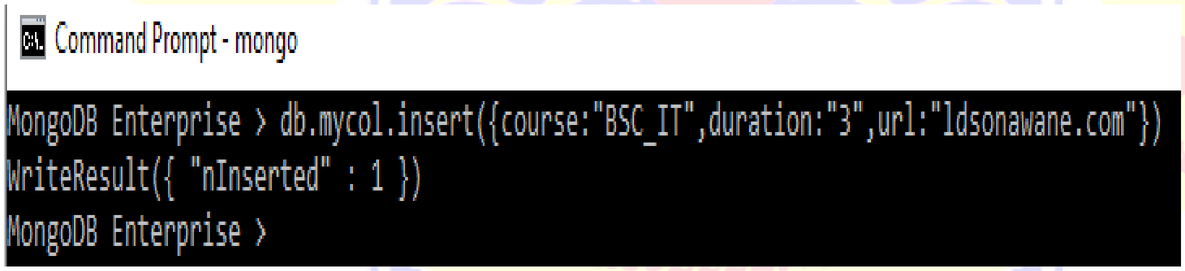
Pratical 1c. Write a MongoDB query to insert, query, update and delete a document.

INSERT DOCUMENT
Syntax:-

db.COLLECTION_NAME.insert(document)

Query:-

```
> db.mycol.insert({course:"BSC_IT",duration:"3", url:"ldsonawane.com"})
```



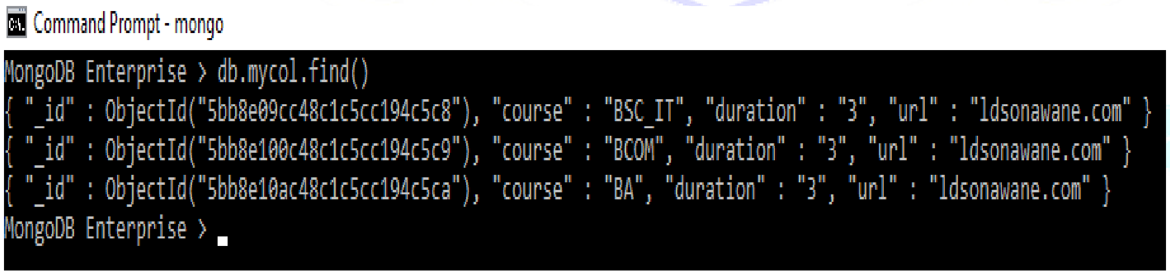
```
MongoDB Enterprise > db.mycol.insert({course:"BSC_IT",duration:"3",url:"ldsonawane.com"})
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise >
```

QUERY DOCUMENT
Syntax:-

>db.COLLECTION_NAME.find()

Query:-

```
> db.mycol.find()
```



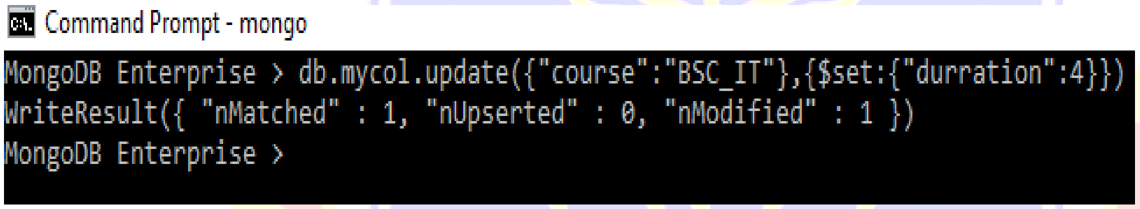
```
MongoDB Enterprise > db.mycol.find()
{ "_id" : ObjectId("5bb8e09cc48c1c5cc194c5c8"), "course" : "BSC_IT", "duration" : "3", "url" : "ldsonawane.com" }
{ "_id" : ObjectId("5bb8e100c48c1c5cc194c5c9"), "course" : "BCOM", "duration" : "3", "url" : "ldsonawane.com" }
{ "_id" : ObjectId("5bb8e10ac48c1c5cc194c5ca"), "course" : "BA", "duration" : "3", "url" : "ldsonawane.com" }
MongoDB Enterprise >
```

UPDATE DOCUMENT
Syntax:-

>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

Query:-

```
> db.mycol.update({"course":"BSC_IT"},{$set:{"durration":4}})
```


Command Prompt - mongo

```
MongoDB Enterprise > db.mycol.update({"course":"BSC_IT"},{$set:{"durration":4}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
MongoDB Enterprise >
```

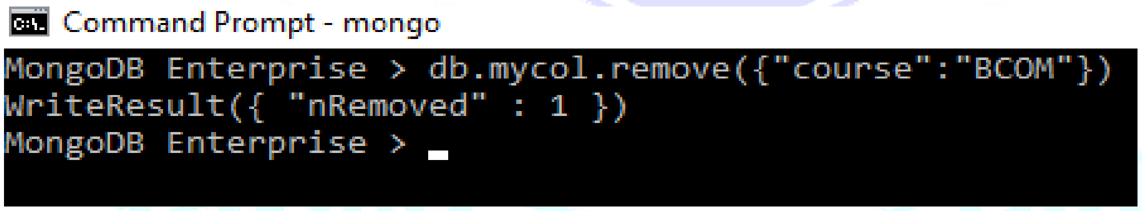
DELETE DOCUMENT

Syntax:-

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

Query:-

```
> db.mycol.remove({"course":"BCOM"})
```


Command Prompt - mongo

```
MongoDB Enterprise > db.mycol.remove({"course":"BCOM"})
WriteResult({ "nRemoved" : 1 })
MongoDB Enterprise > _
```

Learning Outcomes:
By the end of this course, participants will be able to:

Understand the fundamental concepts of NoSQL databases and the specific advantages of MongoDB.
Set up a MongoDB database environment, including installation and configuration.
Perform basic CRUD (Create, Read, Update, Delete) operations using MongoDB shell and drivers.
Utilize MongoDB's query language to retrieve and manipulate data effectively.
Design a simple schema for a MongoDB database, including embedding and referencing.
Implement indexing to optimize query performance.
Understand the basics of aggregation and data analysis in MongoDB.
Course Outcomes:
Upon successful completion of this course, participants will be able to:

Demonstrate proficiency in MongoDB by executing real-world data management tasks.
Apply MongoDB concepts to design and implement scalable applications.
Evaluate and optimize database performance using indexing and aggregation frameworks.
Work with MongoDB in a collaborative environment, utilizing best practices for data management.
Conclusion:
This course provides a foundational understanding of MongoDB, equipping participants with the skills needed to utilize this powerful NoSQL database in modern application development. By combining theory with practical exercises, learners will emerge with the ability to confidently manage data in MongoDB, preparing them for real-world challenges and projects.

Viva Questions:
What are the key differences between SQL and NoSQL databases?
Can you explain the document data model in MongoDB?
How do you perform a basic CRUD operation in MongoDB?
What are indexes, and why are they important in MongoDB?
Describe the aggregation framework in MongoDB.

PRACTICAL NO:-2

SIMPLE MONGODB QUERIES

1. Insert Documents
Adding One Document:

```
db.collectionName.insertOne({ key: "value" });
```

This command adds a single document to the collection. Think of it as putting a single piece of paper into a file.

Adding Multiple Documents:

```
db.collectionName.insertMany([
  { key: "value1" },
  { key: "value2" }
]);
```

This adds several documents at once, like adding multiple papers into the same file.

2. Find Documents
Getting All Documents:

```
db.collectionName.find({});
```

This retrieves everything in the collection, like opening a file to see all the papers inside.

Finding Specific Documents:

```
db.collectionName.find({ key: "value" });
```

This searches for documents that match a certain condition, similar to looking for specific papers with a particular topic.

Finding One Document:

```
db.collectionName.findOne({ key: "value" });
```

This retrieves just one document that matches the condition.

3. Update Documents
Updating One Document:

```
db.collectionName.updateOne(
  { key: "oldValue" },
  { $set: { key: "newValue" } }
);
```

This changes a specific document's value, like crossing out old information on a paper and writing something new.

Updating Multiple Documents:

```
db.collectionName.updateMany(
  { key: "oldValue" },
  { $set: { key: "newValue" } }
);
```

This updates several documents at once, just like changing multiple papers to reflect new information.

4. Delete Documents
Deleting One Document:

```
db.collectionName.deleteOne({ key: "value" });
```

This removes a specific document, like taking one paper out of a file.

Deleting Multiple Documents:

```
db.collectionName.deleteMany({ key: "value" });
```

This removes all documents that match a certain condition, similar to clearing out all related papers.

5. Query Operators
Finding Documents with Conditions:

Greater Than:

```
db.collectionName.find({ age: { $gt: 18 } });
```

This finds documents where age is greater than 18.

Less Than:

```
db.collectionName.find({ age: { $lt: 30 } });
```

This finds documents where age is less than 30.

Using AND/OR:

AND:

db.collectionName.find({ $and: [{ key1: "value1" }, { key2: "value2" }] });
This finds documents that meet both conditions.

OR:

db.collectionName.find({ $or: [{ key1: "value1" }, { key2: "value2" }] });
This finds documents that meet either condition.

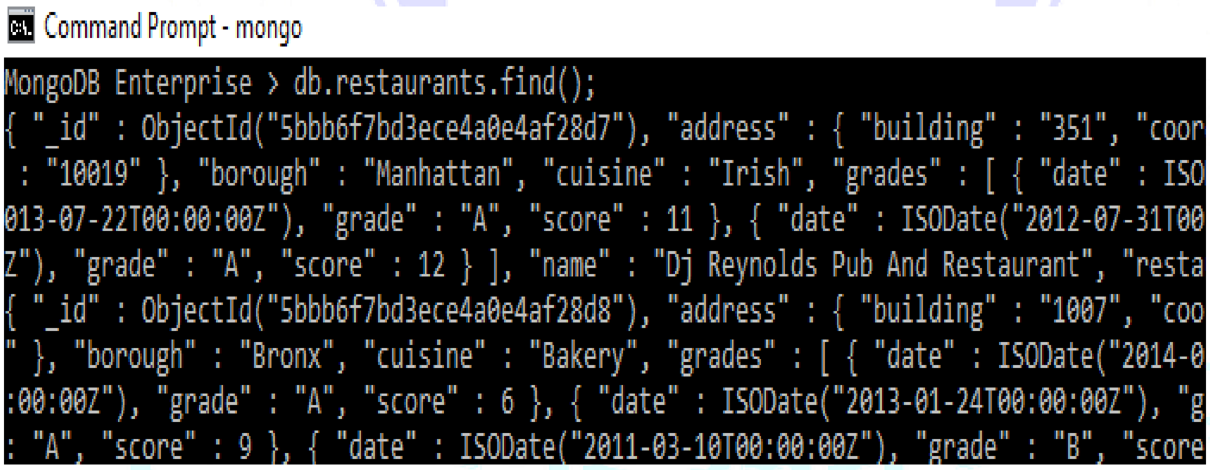
6. Projection
To control which fields you get back:

db.collectionName.find({}, { key1: 1, key2: 1 });
This gets only the specified fields (like key1 and key2) from the documents, ignoring everything else.

Pratical 2a. Write a mongo DB query to display all the documents in the collection restaurants

Query:-

```
db.restaurants.find()
```

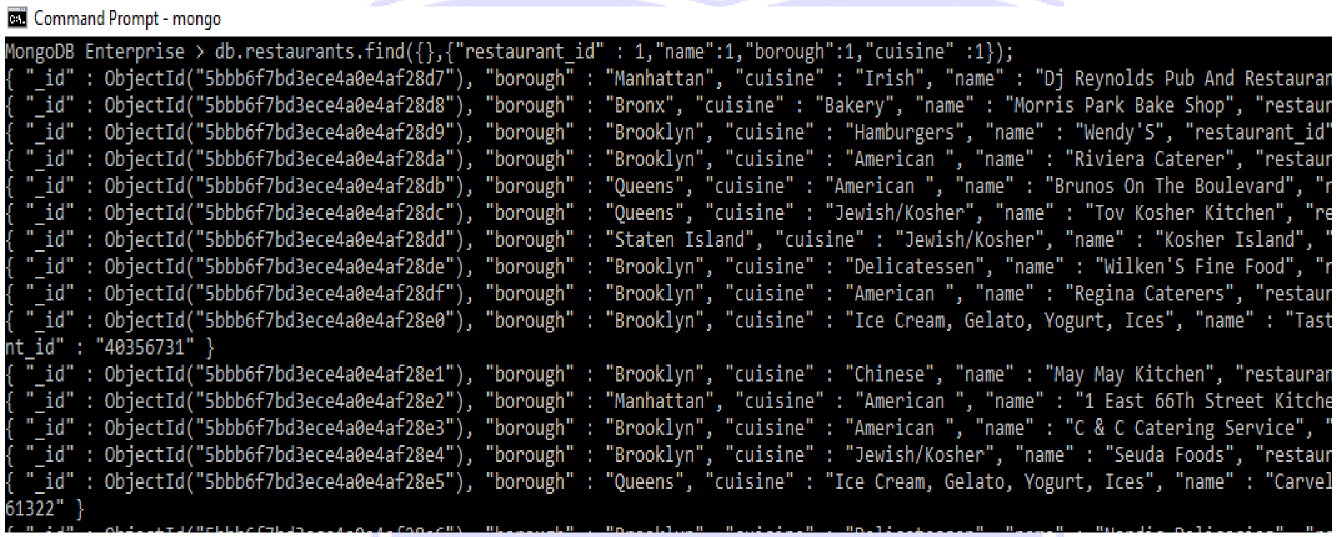

Command Prompt - mongo

```
MongoDB Enterprise > db.restaurants.find();
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28d7"), "address" : { "building" : "351", "coor
 : "10019" }, "borough" : "Manhattan", "cuisine" : "Irish", "grades" : [ { "date" : ISO
013-07-22T00:00:00Z"), "grade" : "A", "score" : 11 }, { "date" : ISODate("2012-07-31T00
Z"), "grade" : "A", "score" : 12 } ], "name" : "Dj Reynolds Pub And Restaurant", "resta
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28d8"), "address" : { "building" : "1007", "coo
" }, "borough" : "Bronx", "cuisine" : "Bakery", "grades" : [ { "date" : ISODate("2014-0
:00:00Z"), "grade" : "A", "score" : 6 }, { "date" : ISODate("2013-01-24T00:00:00Z"), "g
: "A", "score" : 9 }, { "date" : ISODate("2011-03-10T00:00:00Z"), "grade" : "B", "score
```

Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.

Query:-

```
db.restaurants.find({},{"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1});
```



Command Prompt - mongo

```
MongoDB Enterprise > db.restaurants.find({},{"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1});
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28d7"), "borough" : "Manhattan", "cuisine" : "Irish", "name" : "Dj Reynolds Pub And Restauran
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28d8"), "borough" : "Bronx", "cuisine" : "Bakery", "name" : "Morris Park Bake Shop", "restaur
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28d9"), "borough" : "Brooklyn", "cuisine" : "Hamburgers", "name" : "Wendy'S", "restaurant_id"
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28da"), "borough" : "Brooklyn", "cuisine" : "American ", "name" : "Riviera Caterer", "restaur
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28db"), "borough" : "Queens", "cuisine" : "American ", "name" : "Brunos On The Boulevard", "r
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28dc"), "borough" : "Queens", "cuisine" : "Jewish/Kosher", "name" : "Tov Kosher Kitchen", "re
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28dd"), "borough" : "Staten Island", "cuisine" : "Jewish/Kosher", "name" : "Kosher Island", "
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28de"), "borough" : "Brooklyn", "cuisine" : "Delicatessen", "name" : "Wilken'S Fine Food", "r
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28df"), "borough" : "Brooklyn", "cuisine" : "American ", "name" : "Regina Caterers", "restaur
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28e0"), "borough" : "Brooklyn", "cuisine" : "Ice Cream, Gelato, Yogurt, Ices", "name" : "Tast
nt_id" : "40356731" }
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28e1"), "borough" : "Brooklyn", "cuisine" : "Chinese", "name" : "May May Kitchen", "restauran
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28e2"), "borough" : "Manhattan", "cuisine" : "American ", "name" : "1 East 66Th Street Kitche
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28e3"), "borough" : "Brooklyn", "cuisine" : "American ", "name" : "C & C Catering Service", "
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28e4"), "borough" : "Brooklyn", "cuisine" : "Jewish/Kosher", "name" : "Seuda Foods", "restaur
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28e5"), "borough" : "Queens", "cuisine" : "Ice Cream, Gelato, Yogurt, Ices", "name" : "Carvel
61322" }
```

- Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx

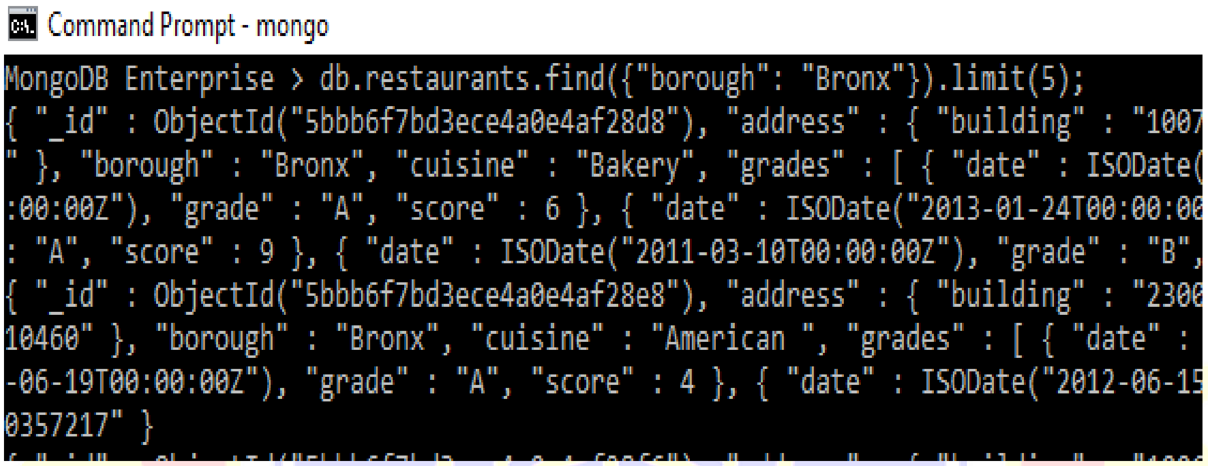
Query:-

```
db.restaurants.find({"borough": "Bronx"}).limit(5);
```

Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx

Query:-

```
db.restaurants.find({"borough":  "Bronx"}).limit(5);
```

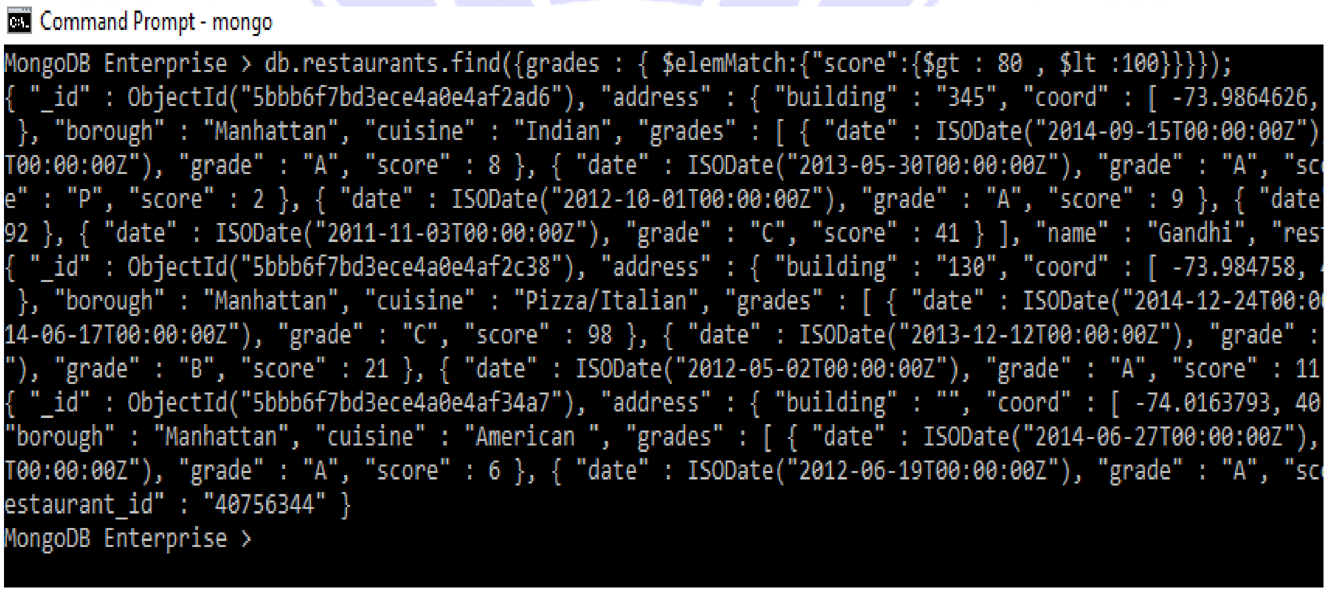


```
MongoDB Enterprise > db.restaurants.find({"borough": "Bronx"}).limit(5);
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28d8"), "address" : { "building" : "1007
" }, "borough" : "Bronx", "cuisine" : "Bakery", "grades" : [ { "date" : ISODate(
:00:00Z"), "grade" : "A", "score" : 6 }, { "date" : ISODate("2013-01-24T00:00:00
: "A", "score" : 9 }, { "date" : ISODate("2011-03-10T00:00:00Z"), "grade" : "B",
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af28e8"), "address" : { "building" : "2300
10460" }, "borough" : "Bronx", "cuisine" : "American ", "grades" : [ { "date" :
-06-19T00:00:00Z"), "grade" : "A", "score" : 4 }, { "date" : ISODate("2012-06-15
0357217" }
```

1)Write a MongoDB query to find the restaurants that achieved a score is more than 80 but less than 100

Query:-

db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 80 , $lt :100}}}});



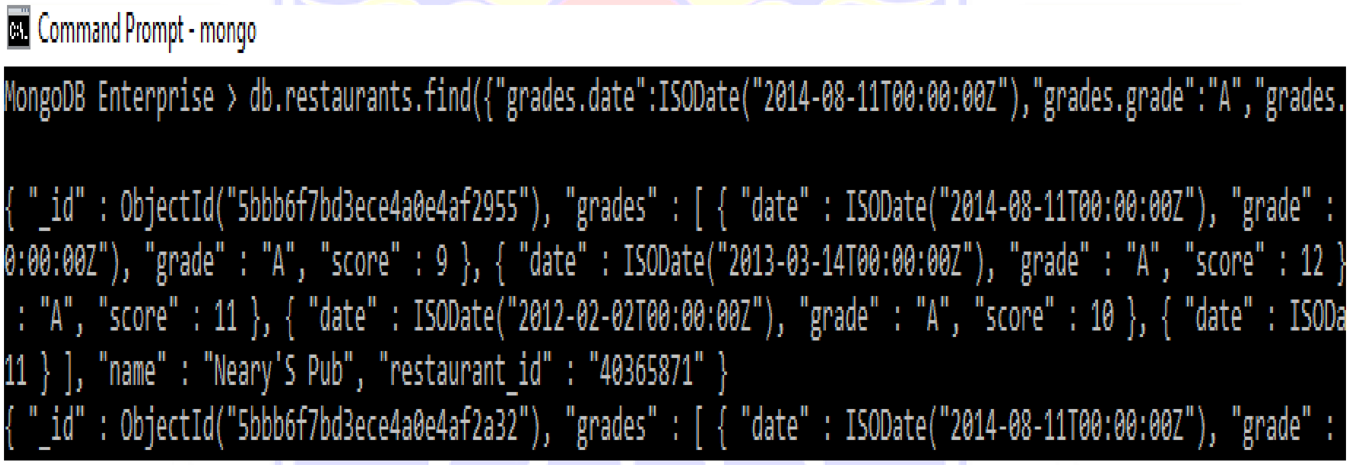
```
MongoDB Enterprise > db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 80 , $lt :100}}}});
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af2ad6"), "address" : { "building" : "345", "coord" : [ -73.9864626,
 }, "borough" : "Manhattan", "cuisine" : "Indian", "grades" : [ { "date" : ISODate("2014-09-15T00:00:00Z")
T00:00:00Z"), "grade" : "A", "score" : 8 }, { "date" : ISODate("2013-05-30T00:00:00Z"), "grade" : "A", "sc
e" : "P", "score" : 2 }, { "date" : ISODate("2012-10-01T00:00:00Z"), "grade" : "A", "score" : 9 }, { "date
92 }, { "date" : ISODate("2011-11-03T00:00:00Z"), "grade" : "C", "score" : 41 } ], "name" : "Gandhi", "res
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af2c38"), "address" : { "building" : "130", "coord" : [ -73.984758, 4
 }, "borough" : "Manhattan", "cuisine" : "Pizza/Italian", "grades" : [ { "date" : ISODate("2014-12-24T00:0
14-06-17T00:00:00Z"), "grade" : "C", "score" : 98 }, { "date" : ISODate("2013-12-12T00:00:00Z"), "grade" :
"), "grade" : "B", "score" : 21 }, { "date" : ISODate("2012-05-02T00:00:00Z"), "grade" : "A", "score" : 11
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af34a7"), "address" : { "building" : "", "coord" : [ -74.0163793, 40
"borough" : "Manhattan", "cuisine" : "American ", "grades" : [ { "date" : ISODate("2014-06-27T00:00:00Z"),
T00:00:00Z"), "grade" : "A", "score" : 6 }, { "date" : ISODate("2012-06-19T00:00:00Z"), "grade" : "A", "sc
estaurant_id" : "40756344" }
MongoDB Enterprise >
```

2)Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an

ISODate "2014-08-11T00:00:00Z" among many of survey dates.

Query:-

```
        db.restaurants.find({"grades.date":ISODate("2014- 08-
11T00:00:00Z"),"grades.grade":"A","grades.score"
:11},{"restaurant_id":1,"name":1,"grades":1});
```


Command Prompt - mongo

```
MongoDB Enterprise > db.restaurants.find({"grades.date":ISODate("2014-08-11T00:00:00Z"),"grades.grade":"A","grades.

{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af2955"), "grades" : [ { "date" : ISODate("2014-08-11T00:00:00Z"), "grade" :
0:00:00Z"), "grade" : "A", "score" : 9 }, { "date" : ISODate("2013-03-14T00:00:00Z"), "grade" : "A", "score" : 12 }
 : "A", "score" : 11 }, { "date" : ISODate("2012-02-02T00:00:00Z"), "grade" : "A", "score" : 10 }, { "date" : ISODa
11 } ], "name" : "Neary'S Pub", "restaurant_id" : "40365871" }
{ "_id" : ObjectId("5bbb6f7bd3ece4a0e4af2a32"), "grades" : [ { "date" : ISODate("2014-08-11T00:00:00Z"), "grade" :
```

Learning Outcomes:
By the end of this course, participants will be able to:

Understand the basic operations (CRUD) in MongoDB.
Perform data insertion, retrieval, updating, and deletion using simple queries.
Use query operators to filter results based on specific conditions.
Apply projection to limit the fields returned in query results.
Recognize the significance of MongoDB's document-oriented structure in data management.
Course Outcomes:
Upon successful completion of this course, participants will be able to:

Execute basic MongoDB queries to manage data effectively.
Design simple queries to retrieve specific information from collections.
Modify and remove documents using various query methods.
Demonstrate an understanding of how to use query operators to enhance data retrieval.
Explain the role of projection in optimizing query results.
Conclusion:
This course has equipped participants with essential skills to perform simple MongoDB queries effectively. By mastering CRUD operations and understanding query operators, learners can efficiently manage and manipulate data in a MongoDB environment. These foundational skills will serve as a stepping stone for more advanced topics in NoSQL databases.

Viva Questions on Simple MongoDB Queries:
What does CRUD stand for in the context of MongoDB?
How do you insert a single document into a MongoDB collection?
Describe the difference between find() and findOne() methods.
What is the purpose of the updateOne() method in MongoDB?
How would you delete multiple documents that match a specific condition?

PRACTICAL NO:-3

Implementing Aggregation

Objective:
To understand and implement the aggregation framework in MongoDB for data analysis and transformation.

Introduction to Aggregation:
Aggregation in MongoDB is a powerful tool used to process data and return computed results. It allows for operations like filtering, grouping, and sorting data, similar to SQL's GROUP BY clause but with more flexibility.

Key Aggregation Stages:
$match: Filters documents based on specified criteria (similar to a WHERE clause).
$group: Groups documents by a specified identifier and performs aggregations (like SUM, AVG).
$sort: Sorts the documents based on specified fields.
$project: Reshapes documents by specifying which fields to include or exclude.
$limit: Limits the number of documents returned.

Pratical 3a. Write a MongoDB query to use sum, avg, min and max expression.

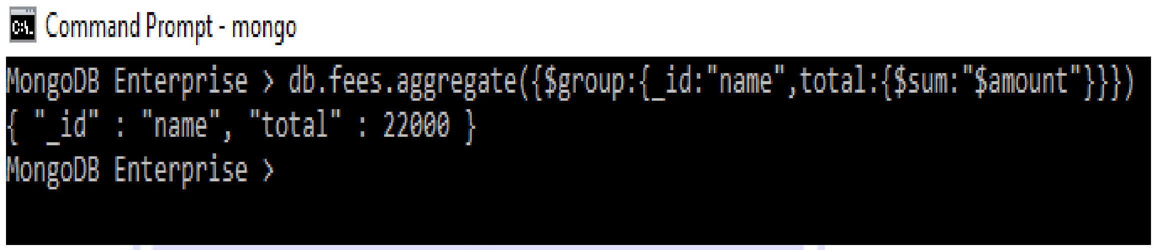
SUM

Syntax:-

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Query:-

```
db.fees.aggregate({$group:{_id:"name",total:{$sum:"$amou
nt"}}})
```



Command Prompt - mongo

```
MongoDB Enterprise > db.fees.aggregate({$group:{_id:"name",total:{$sum:"$amount"}}})
{ "_id" : "name", "total" : 22000 }
MongoDB Enterprise >
```

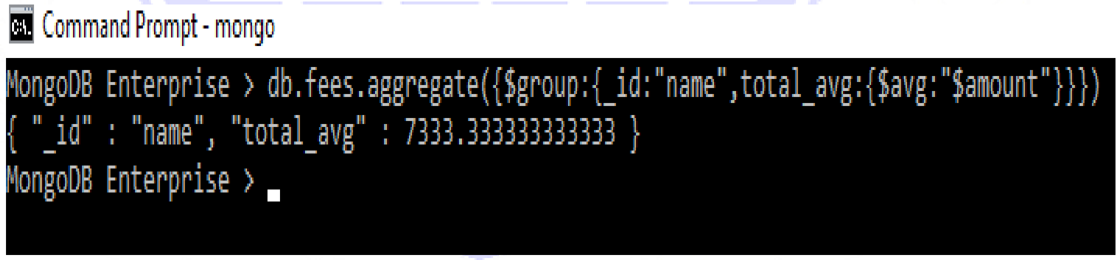
AVG

Syntax:-

```
>db.COLLECTION_NAME.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
```

Query:-

db.fees.aggregate({$group:{_id:"name",total_avg:{$avg:"$amount"}}})



Command Prompt - mongo

```
MongoDB Enterprise > db.fees.aggregate({$group:{_id:"name",total_avg:{$avg:"$amount"}}})
{ "_id" : "name", "total_avg" : 7333.333333333333 }
MongoDB Enterprise >
```

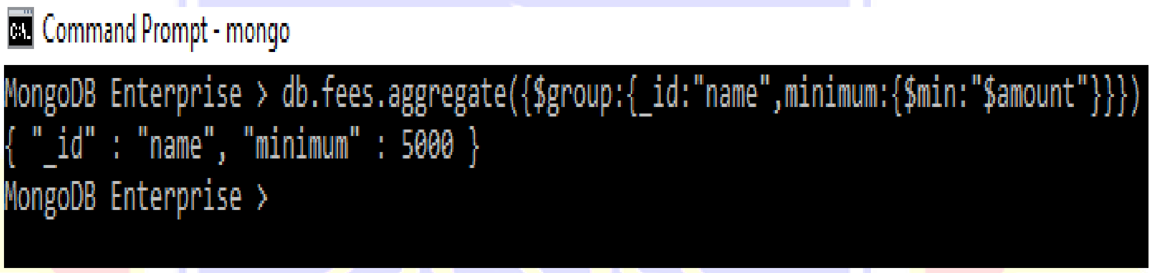
MIN

Syntax:-

```
>db.COLLECTION_NAME.aggregate[{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
```

Query:-

db.fees.aggregate({$group:{_id:"name",minimum:{$min:"$amount"}}})


Command Prompt - mongo

```
MongoDB Enterprise > db.fees.aggregate({$group:{_id:"name",minimum:{$min:"$amount"}}})
{ "_id" : "name", "minimum" : 5000 }
MongoDB Enterprise >
```

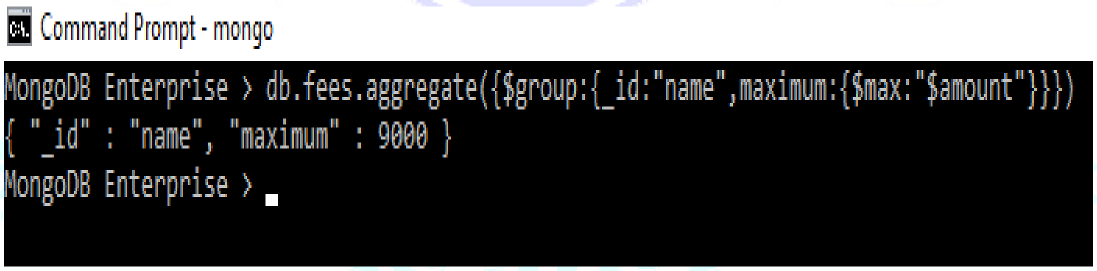
MAX

Syntax:-

```
>db.COLLECTION_NAME.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
```

Query:-

db.fees.aggregate({$group:{_id:"name",maximum:{$max:"$amount"}}})


Command Prompt - mongo

```
MongoDB Enterprise > db.fees.aggregate({$group:{_id:"name",maximum:{$max:"$amount"}}})
{ "_id" : "name", "maximum" : 9000 }
MongoDB Enterprise >
```

Pratical 3b. Write a MongoDB query to use push and addToSet expression.

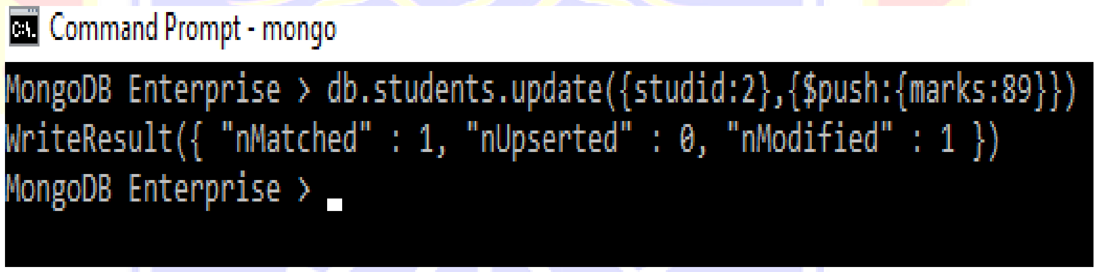
PUSH

Syntax:-

```
>db.COLLECTION_NAME.update({ $push: { <field1>: <value1>, ... } })
```

Query:-

```
db.students.update({ name: “sid” },{ $push: { marks: 89 } })
```



Command Prompt - mongo

```
MongoDB Enterprise > db.students.update({studid:2},{$push:{marks:89}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
MongoDB Enterprise >
```

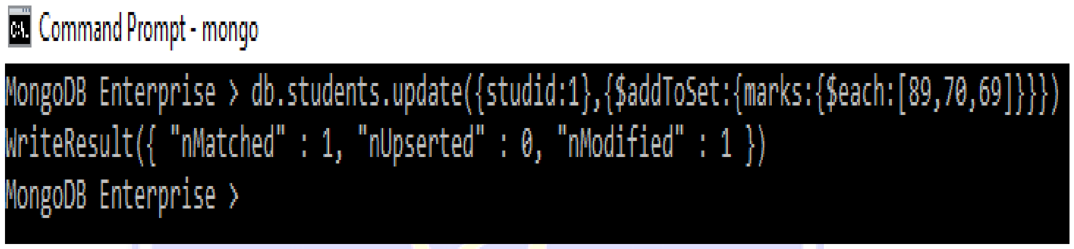
ADDTOSET

Syntax:-

```
>db.COLLECTION_NAME.update({ <field> : <value> } , { $addtoset:
{ <field1>: <addition> } })
```

Query:-

```
db.students.update({ name: “sid” },{ $addToSet: {
marks:{$each:[89,70,69]} } })
```


Command Prompt - mongo

```
MongoDB Enterprise > db.students.update({studid:1},{$addToSet:{marks:{$each:[89,70,69]}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
MongoDB Enterprise >
```

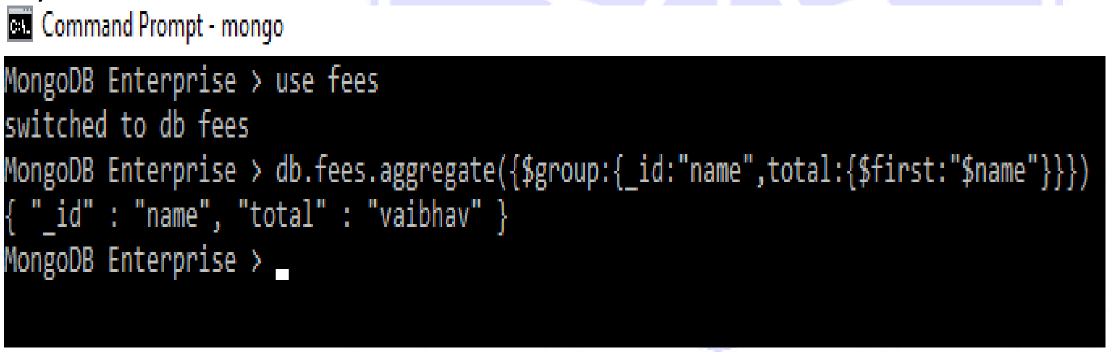
Pratical 3c.   Write a MongoDB query to use first and last expression

Syntax:-

>db.COLLECTION_NAME.aggregate([$group : {_id: “$by_user”, first url : {$first : “$url”} } })

Query:-

db.fees.aggregate({$group:{_id:"name",total:{$first:"$name"}}}
)


Command Prompt - mongo

```
MongoDB Enterprise > use fees
switched to db fees
MongoDB Enterprise > db.fees.aggregate({$group:{_id:"name",total:{$first:"$name"}}})
{ "_id" : "name", "total" : "vaibhav" }
MongoDB Enterprise >
```

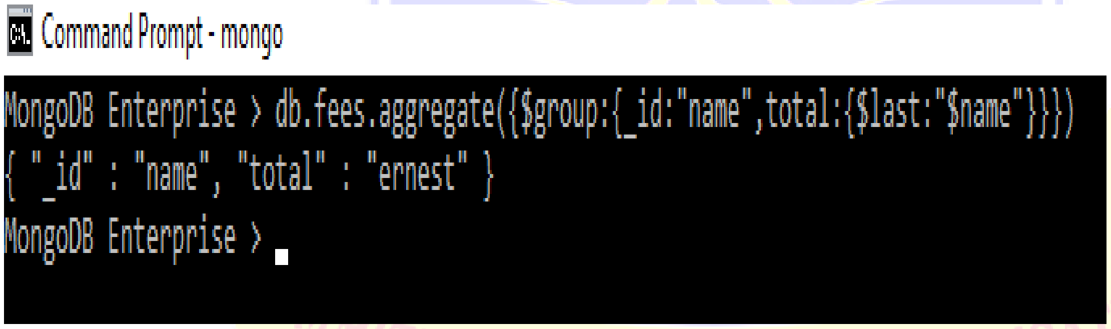
LAST

Syntax:-

>db.COLLECTION_NAME.aggregate([$group : {_id: “$by_user”, last ur
l : {$last : “$url”} } })

Query:-

db.fees.aggregate({$group:{_id:"name",total:{$last:"$name"}}})



```
Command Prompt - mongo
MongoDB Enterprise > db.fees.aggregate({$group:{_id:"name",total:{$last:"$name"}}})
{ "_id" : "name", "total" : "ernest" }
MongoDB Enterprise >
```

Learning Outcomes:
By the end of this practical session, participants will be able to:
Understand the concept and importance of data aggregation in MongoDB.
Utilize the aggregation framework to perform complex data queries.
Implement various aggregation stages, including $match, $group, $sort, $project, and $limit.
Analyze data to derive meaningful insights using aggregation techniques.
Course Outcomes:
Upon successful completion of this practical session, participants will be able to:

Construct aggregation pipelines to summarize and transform data effectively.
Execute queries that involve grouping, filtering, and sorting data.
Demonstrate the ability to work with real-world datasets to extract actionable insights.
Communicate the results of aggregation queries in a clear and concise manner.
Conclusion:
This practical session has equipped participants with foundational skills in using MongoDB's aggregation framework. By exploring various aggregation stages and their applications, learners are now capable of analyzing and manipulating data to derive valuable insights. Mastering these techniques is essential for effective data management and analysis in MongoDB.

Viva Questions on Implementing Aggregation:
What is the purpose of the aggregation framework in MongoDB?
Describe the role of the $group stage in an aggregation pipeline.
How does the $match stage differ from the $filter operator?
Can you explain what the $sort stage does in an aggregation pipeline?

PRACTICAL NO :- 4

Replication, Backup and Restore

Introduction
MongoDB, as a NoSQL database, offers various features for data redundancy, high availability, and disaster recovery. Understanding replication, backup, and restore processes is crucial for maintaining data integrity and availability in production environments.

Replication
Definition: Replication in MongoDB is the process of synchronizing data across multiple servers to ensure high availability and data redundancy. This is achieved through replica sets.

Key Concepts:

Replica Set:

A replica set is a group of MongoDB servers that maintain the same dataset.
It consists of a primary node (which receives all write operations) and one or more secondary nodes (which replicate data from the primary).
Automatic Failover:

If the primary node fails, one of the secondary nodes can be automatically elected as the new primary, ensuring continuous availability.
Data Consistency:

MongoDB uses a write concern mechanism to define the level of acknowledgment required from the server for write operations. This ensures data consistency across replicas.
Benefits of Replication:

High availability: The system can continue to operate even if one node goes down.
Data redundancy: Multiple copies of data provide safety against data loss.
Read scaling: Read operations can be distributed among secondary nodes.
Backup
Definition: Backup in MongoDB refers to the process of creating copies of the database to safeguard against data loss due to failures, accidental deletions, or corruption.

Pratical 4a. Write a MongoDB query to create Replica of existing database.

1) Create Replica Set

• MongoDB must successfully be running on the primary server instance. The standalone instance of MongoDB can become a replica set member by assigning by replica set name in the MongoDB configuration file.

• To create a replica set called “rs0”, add the following line to/etc/mongod conf.

replSet=rs0

• MongoDB will need to be restarted for the change to take affect. Sudo/etc/init.d/mongod restart

• Use the mongo shell to connect once the service is running again. mongo

• Initiate the replica set from within the mongo shell.

rs.initiate()

• The initial replica set configuration can be verified using rs.conf().

```
rs.conf()
{
“_id”:”rs0”, “version”:1,
“members”:,
{
“_id”:0,
“host”:”mongodb01.yourdomain.com:27017”
}
]
}
```

- At this stage, the mongo shell prompt should also indicate the replica name andreplica member role. For example:
  Rs0:PRIMARY>
- The replica set is now created but more member must be added to achieveredundancy and increased data availability.

Add Secondary Members

- The secondary MongoDB server instances must be successfully running MongoDBand accessible by the other members.
- The configuration file on the secondary members must reflect the same replica set name as the primary member. Add the following line to the/etc/mongod.conf on allthe secondary members.
  replSet=rs0
- Restart the MongoDB service.
  Sudo/etc/init.d/mongod restart

- The secondary members can now be added from the mongo shell on the primaryserver using rs.add().
  ```
  rs():PRIMARY>rs.add("mongodb02.yourdomain.com")
  rs():PRIMARY>rs.add("mongodb03.yourdomain.com")
  ```
- The rs.status() command can be used to verify the status of the replica set:
  ```
  rs.status()
  ```

Add an Arbiter

- An arbiter is an optional member of a replica set which stores no data. Instead it isavailable to break election ties in a situation where there is an even number of MongoDB members.
- An arbiter member does not require the same performance capabilities of a full MongoDB server, but it must reside on a separate server instance. Never configurean arbiter on an existing replica set member.
- To add an arbiter, first install MongoDB on a server instance accessible by the otherreplica set members and add the replica set name to/etc/mongod.conf.
  ```
  replSet=rs0
  ```
- Unneccessary data can be reduced on the arbiter server by adding the following filesto/etc/mongod.conf.
  ```
  journal.enabled=false
  smallFiles=true
  preallocDataFiles=false
  ```
- Now long into the primary replica set member and launch the mongo shell.mongo
- The arbiter can be added with the following command:
  ```
  rs.addArb("arbiter01.yourdomain.com:27017")
  ```

Practical4b:- Write a mongoDB query to create a backup of existing database

a) 1)Backup a wholedatabase

```
Mongodump-d devdb-o mongoBackups
```

devdb is the name of the database and mongoBackups is the directory in which 1 want todump the database.

2) Backup a Single Collection

```
Mongodump –d devdb-o mongoBackups—collection vehicles
```

devdb is the name of the database and mongoBackups is the directory in which I want todump the database and vehicles is the name of the collection which I want to backup.

b) Write a mongoDb query to Restore database from the backup

1) Restore Whole Database

```
Mongorestore –d devdb mongoBackups/devdb
```

devdb is the name of the database into which we will restore data and mongoBackups is the directory in which dumped the database devdb.

Learning Outcomes:

1. Understand the concepts and importance of data replication, backup, and restore processes.
2. Implement various data replication techniques to ensure data availability and redundancy.
3. Execute backup strategies for different types of data and systems, including full, incremental, and differential backups.
4. Develop and test restore procedures to ensure data integrity and quick recovery in case of data loss.
5. Evaluate and choose appropriate tools and technologies for replication, backup, and restore based on specific use cases.

Course Outcomes

1. Demonstrate proficiency in setting up and managing data replication in various environments (e.g., databases, file systems).
2. Create a comprehensive backup plan tailored to organizational needs and compliance requirements.
3. Conduct disaster recovery drills to validate backup and restore procedures.
4. Analyze case studies on data loss and recovery to identify best practices and lessons learned.
5. Provide recommendations for improving data resilience and availability.

Conclusion

In conclusion, mastering replication, backup, and restore processes is essential for maintaining data integrity, availability, and disaster recovery capabilities in modern IT environments. By understanding and implementing these concepts, organizations can protect their critical assets and ensure business continuity in the face of unexpected data loss.

Viva Questions on Replication, Backup, and Restore

1. What is data replication, and why is it important?
2. Can you explain the difference between full, incremental, and differential backups?
3. What are some common replication methods?

PRACTICAL NO :- 5

JAVA AND MONGODB

Objective:
To understand how to integrate Java applications with MongoDB for data storage and retrieval.

Introduction:
MongoDB is a popular NoSQL database that is often used in conjunction with Java for building scalable and flexible applications. By leveraging the MongoDB Java Driver, developers can perform various operations such as creating, reading, updating, and deleting documents in MongoDB directly from Java code.

Key Concepts:
MongoDB Java Driver:

The MongoDB Java Driver is a library that allows Java applications to interact with MongoDB.
It provides a simple and flexible API for executing commands and managing data.
Maven Dependency:

To use the MongoDB Java Driver in a Maven project, include the following dependency in your pom.xml file:

```
<dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongodb-driver-sync</artifactId>
  <version>4.9.0</version> <!-- Check for the latest version -->
</dependency>
```

Connecting to MongoDB:

You can connect to a MongoDB instance using the MongoClient class.
Example:

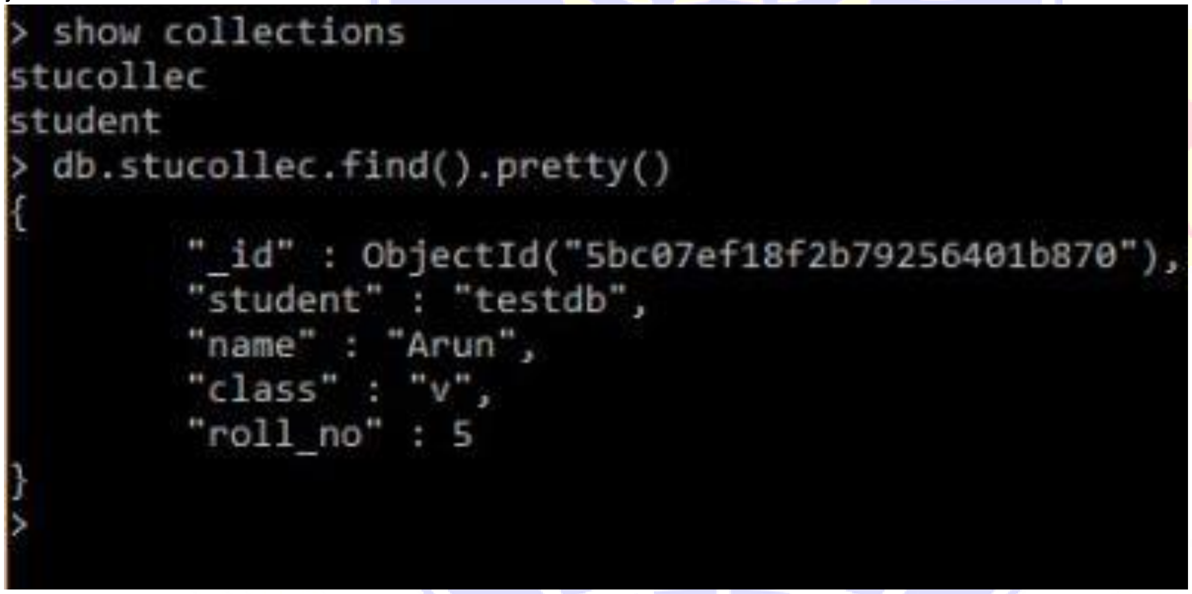
```
MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");
MongoDatabase database = mongoClient.getDatabase("mydatabase");
```

Pratical 5a. Connecting Java with MongoDB and inserting, retrieving, updating and deleting

INSERT

Query:-

```
public class Mongodb_connection_insert
{
public static void main(String args[])
{
try{
MongoClient mongoclient =new MongoClient("localhost",27017);
DB db=mongoClient.getDB("testdb");
System.out.println("Connection successful");
DBCollection collec=db.getCollection("stucollec");
BasicDBObject doc= new BasicDBObject("student","testdb").append("name","Arun
").apped("class","v").append("roll_no",5); collec.insert(doc);
System.out.println("inserted");}
 catch(Exception e)
{
System.err.println(e.getClass().getName()+"."+e.get Message())
}
}
}
```



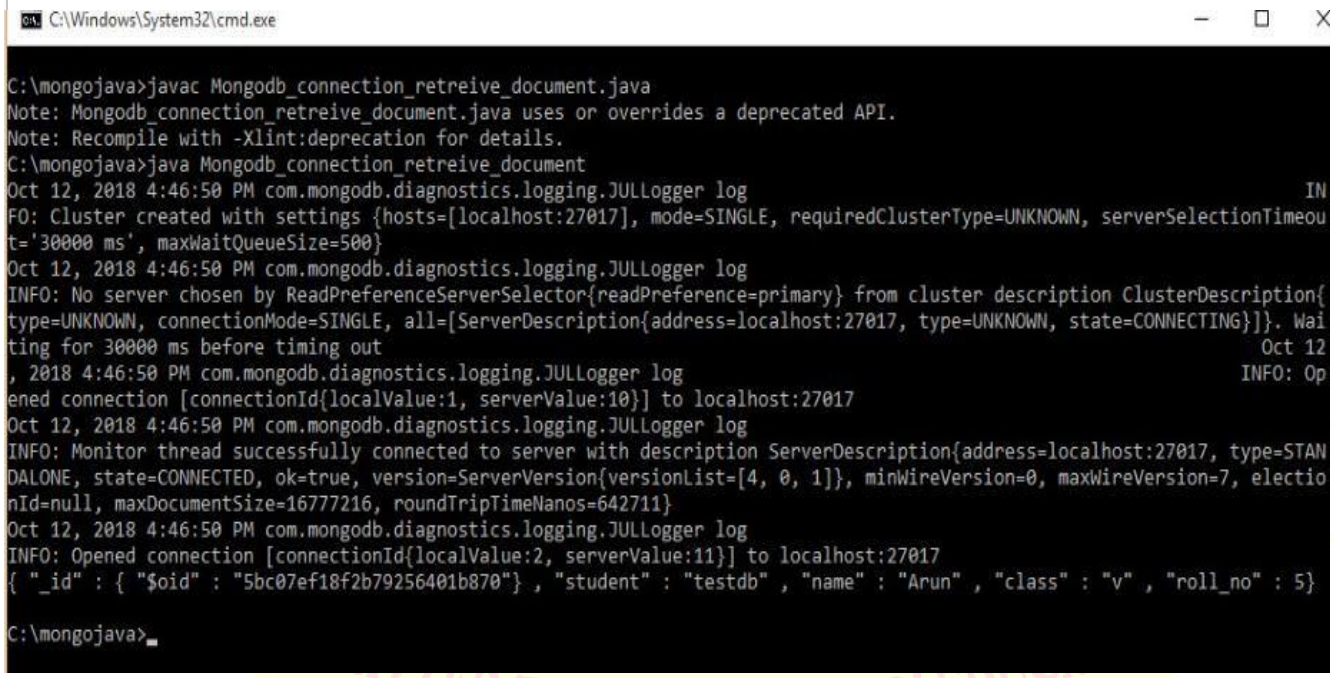
```
> show collections
stucollec
student
> db.stucollec.find().pretty()
{
        "_id" : ObjectId("5bc07ef18f2b79256401b870"),
        "student" : "testdb",
        "name" : "Arun",
        "class" : "v",
        "roll_no" : 5
}
>
```

RETRIVE

Query:-

```
        import com.mongodb.MongoClient; import
        com.mongodb.MongoException; import
        com.mongodb.WriteConcern; import com.mongodb.DB;
        import com.mongodb.DBCollection; import
        com.mongodb.BasicDBObject; import
        com.mongodb.DBObject; import
        com.mongodb.DBCursor; import
        com.mongodb.ServerAddress;
        import java.util.Arrays;
```

```
public class Mongodb_connection_find_all_documents
{
        public static void main(String args[])
                {
                try{
                MongoClient mongoclient=new MongoClient("localhost",27017);
                DB db=mongoClient.getDB("testdb"); DBCollection
                collec=db.getCollection("stucollec"); DBCursor cursor=collec.find();
                        try
                            {
                                    While(cursor.hasNext())
                            {
                                     System.out.println(cursor.next());
                            }
                                 } finally
                            {
                                  cursor.close();
                                     }
                                   } catch(Exception e)
                            {
              System.err.println(e.getClass().getName()+"."+e.getMessage());
                                }
                                    }
                            }
```

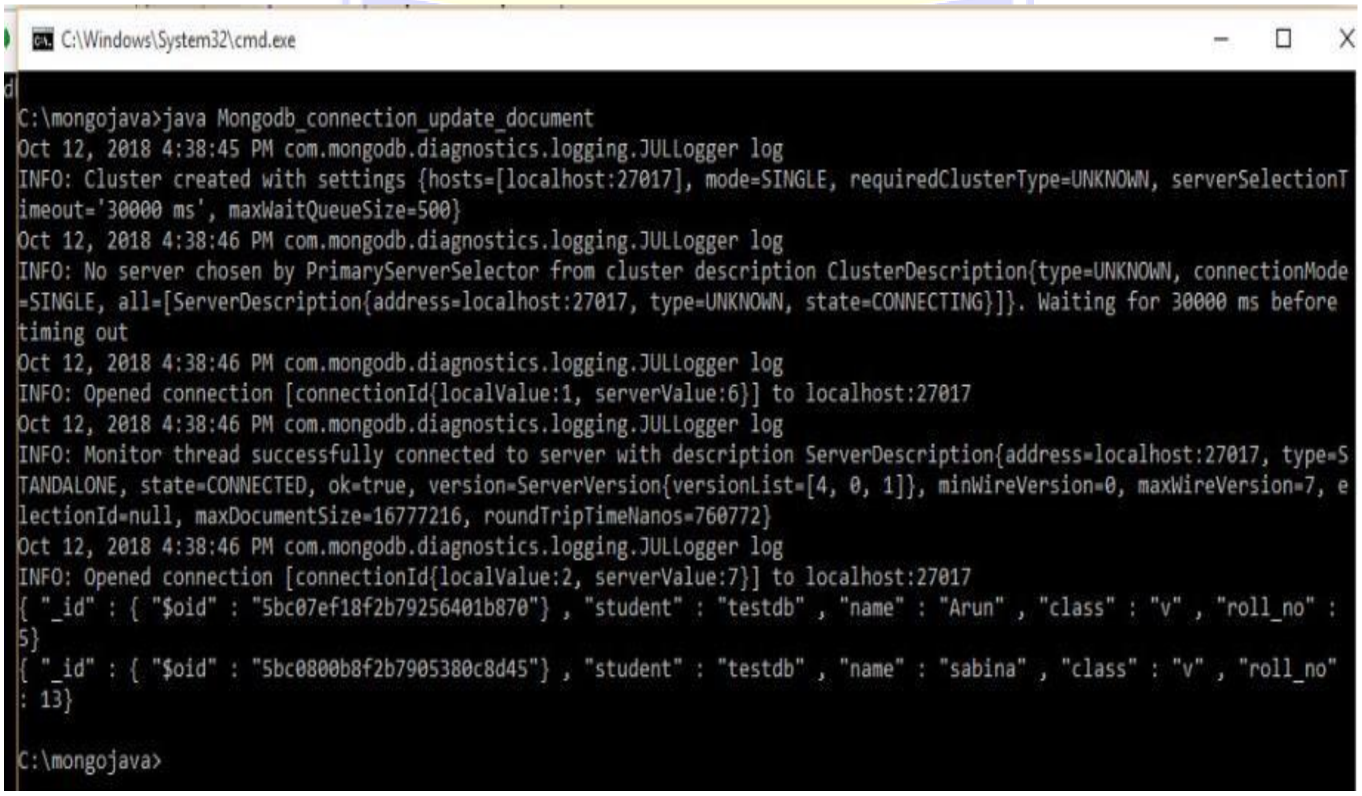


```
C:\Windows\System32\cmd.exe

C:\mongojava>javac Mongodb_connection_retreive_document.java
Note: Mongodb_connection_retreive_document.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
C:\mongojava>java Mongodb_connection_retreive_document
Oct 12, 2018 4:46:50 PM com.mongodb.diagnostics.logging.JULLogger log                                      IN
FO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeou
t='30000 ms', maxWaitQueueSize=500}
Oct 12, 2018 4:46:50 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: No server chosen by ReadPreferenceServerSelector{readPreference=primary} from cluster description ClusterDescription{
type=UNKNOWN, connectionMode=SINGLE, all=[ServerDescription{address=localhost:27017, type=UNKNOWN, state=CONNECTING}]}. Wai
ting for 30000 ms before timing out                                                                              Oct 12
, 2018 4:46:50 PM com.mongodb.diagnostics.logging.JULLogger log                                              INFO: Op
ened connection [connectionId{localValue:1, serverValue:10}] to localhost:27017
Oct 12, 2018 4:46:50 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=STAN
DALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[4, 0, 1]}, minWireVersion=0, maxWireVersion=7, electio
nId=null, maxDocumentSize=16777216, roundTripTimeNanos=642711}
Oct 12, 2018 4:46:50 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:2, serverValue:11}] to localhost:27017
{ "_id" : { "$oid" : "5bc07ef18f2b79256401b870"} , "student" : "testdb" , "name" : "Arun" , "class" : "v" , "roll_no" : 5}

C:\mongojava>_
```

UPDATE

Query:-

```
public class Mongodb_connection_update_document
{
public static void main(String args[])
{
try{
MongoClient mongoclient =new MongoClient("localhost",27017);
DB db=mongoClient.getDB("testdb");
  DBCollection collec=db.getCollection("stucollec");
 DBObject query=new BasicDBObject(“name”,”sabina”);
 DBObject update=new BasicDBObject();
 Update.put(“$set”,new BasicDBObject(“roll_no”,13));
  WriteResult result=collec.update(query,update);
 DBCursor cursor=collec.find();
try{
 while(cursor.hasNext()) System.out.println(cursor.next());

 {

  System.out.println(cursor.next());
}}

finally
        {       cursor close();
        }
}
        catch(Exception e)
        {
        System.err.println(e.getClass().getName()+"."+e.getMessage());
        }
    }
}
```

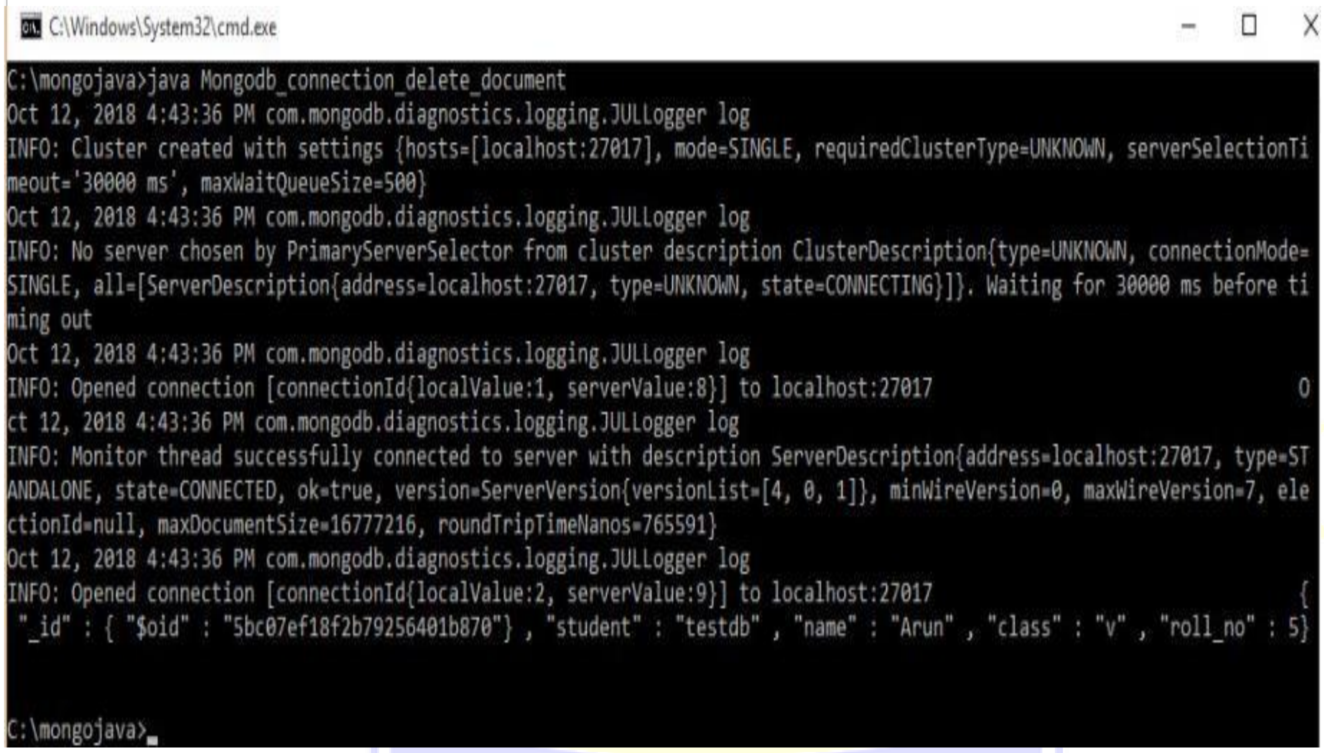


```
C:\mongojava>java Mongodb_connection_update_document
Oct 12, 2018 4:38:45 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionT
imeout='30000 ms', maxWaitQueueSize=500}
Oct 12, 2018 4:38:46 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: No server chosen by PrimaryServerSelector from cluster description ClusterDescription{type=UNKNOWN, connectionMode
=SINGLE, all=[ServerDescription{address=localhost:27017, type=UNKNOWN, state=CONNECTING}]}. Waiting for 30000 ms before
timing out
Oct 12, 2018 4:38:46 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:1, serverValue:6}] to localhost:27017
Oct 12, 2018 4:38:46 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=S
TANDALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[4, 0, 1]}, minWireVersion=0, maxWireVersion=7, e
lectionId=null, maxDocumentSize=16777216, roundTripTimeNanos=760772}
Oct 12, 2018 4:38:46 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:2, serverValue:7}] to localhost:27017
{ "_id" : { "$oid" : "5bc07ef18f2b79256401b870"} , "student" : "testdb" , "name" : "Arun" , "class" : "v" , "roll_no" :
5}
{ "_id" : { "$oid" : "5bc0800b8f2b7905380c8d45"} , "student" : "testdb" , "name" : "sabina" , "class" : "v" , "roll_no"
: 13}

C:\mongojava>
```

DELETE

Query:-

```
public class Mongodb_connection_delete_document
{
 public static void main(String args[])
   {
     try{
          MongoClient mongoclient =new MongoClient("localhost",27017);
          DB db=mongoClient.getDB("testdb"); DBCollection
           collec=db.getCollection("stucollec");
          DBObject query=new BasicDBObject("name","sabina");
          DBObject update=new BasicDBObject();
          Update.put("$set",new BasicDBObject("roll_no",13));
          WriteResult result=collec.remove(query); DBCursor
cursor=collec.find();
try{



while(cursor.hasNext())

{

System.out.println(cursor.next());
}}
finally
{         cursor close();
}
}
catch(Exception e)
{
System.err.println(e.getClass().getName()+"."+e.getMessage());
}
}
}
```



```
C:\mongojava>java Mongodb_connection_delete_document
Oct 12, 2018 4:43:36 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTi
meout='30000 ms', maxWaitQueueSize=500}
Oct 12, 2018 4:43:36 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: No server chosen by PrimaryServerSelector from cluster description ClusterDescription{type=UNKNOWN, connectionMode=
SINGLE, all=[ServerDescription{address=localhost:27017, type=UNKNOWN, state=CONNECTING}]}. Waiting for 30000 ms before ti
ming out
Oct 12, 2018 4:43:36 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:1, serverValue:8}] to localhost:27017                                  O
ct 12, 2018 4:43:36 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=ST
ANDALONE, state=CONNECTED, ok=true, version=ServerVersion{versionList=[4, 0, 1]}, minWireVersion=0, maxWireVersion=7, ele
ctionId=null, maxDocumentSize=16777216, roundTripTimeNanos=765591}
Oct 12, 2018 4:43:36 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:2, serverValue:9}] to localhost:27017                                  {
 "_id" : { "$oid" : "5bc07ef18f2b79256401b870"} , "student" : "testdb" , "name" : "Arun" , "class" : "v" , "roll_no" : 5}


C:\mongojava>_
```

Learning Outcomes:

By the end of this practical session, participants will be able to:

1. Understand how to connect a Java application to a MongoDB database.
2. Perform basic CRUD (Create, Read, Update, Delete) operations using Java.
3. Utilize the MongoDB Java Driver to interact with MongoDB collections.
4. Apply Java programming concepts to manage and manipulate data stored in MongoDB.

Course Outcomes:

Upon successful completion of this practical session, participants will be able to:

1. Implement a Java application that integrates with MongoDB for data storage.
2. Execute complex queries and data manipulations through Java code.
3. Demonstrate the ability to manage MongoDB collections effectively within a Java environment.
4. Communicate the principles of using NoSQL databases in Java applications.

Conclusion:

This practical session has equipped participants with essential skills to effectively use Java in conjunction with MongoDB. By mastering the integration of these technologies, learners can build scalable applications that leverage the power of NoSQL databases. Understanding these concepts will facilitate the development of data-driven applications in real-world scenarios.

Viva Questions on Java and MongoDB:

1. What is the purpose of the MongoDB Java Driver?
2. How do you connect a Java application to a MongoDB database?
3. Can you explain how to insert a document into a MongoDB collection using Java?
4. What is the difference between insertOne() and insertMany() methods?

PRACTICAL NO :- 6

PHP AND MONGODB

Pratical 6a. Connecting Java with MongoDB and inserting, retrieving, updating and deleting

CONNECTION

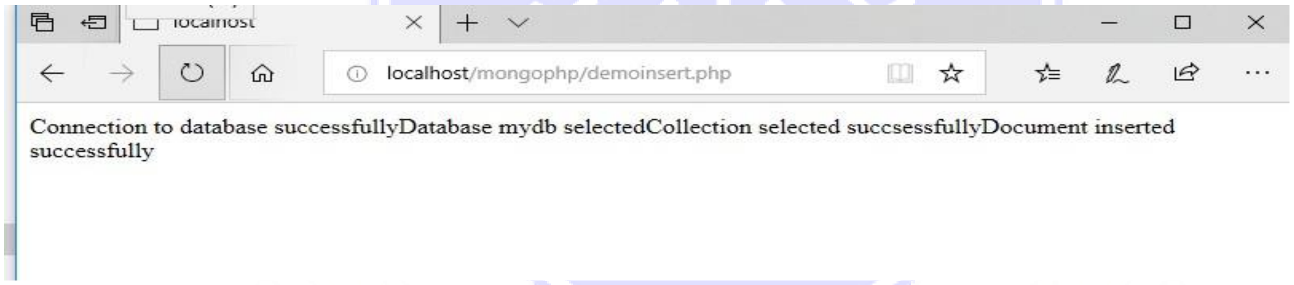
Query:-

```
<?php
  // connect to mongodb
  $m = new MongoClient();

  echo "Connection to database successfully";
  // select a database
  $db = $m->mydb;

  echo "Database mydb selected";
?>
```

INSERTING

Query:-

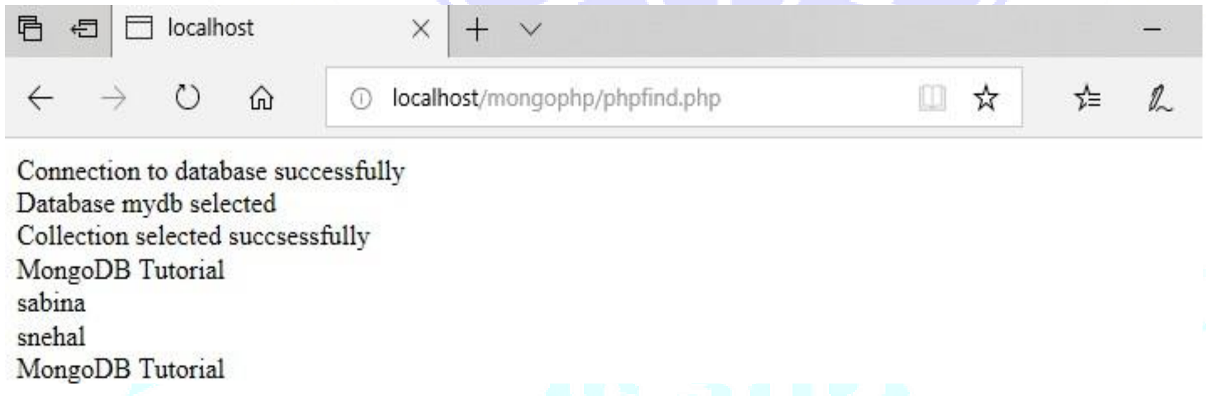
```
<?php
  // connect to mongodb
  $m = new MongoClient();
  echo "Connection to database successfully";

  // select a database
  $db = $m->mydb;
  echo "Database mydb selected";
  $collection = $db->mycol;
  echo "Collection selected succsessfully";

  $document = array(
    "title" => "MongoDB",
    "description" => "database",
    "likes" => 100,
    "url" => "http://www.tutorialspoint.com/mongodb/",
    "by" => "tutorials point"
  );

  $collection->insert($document);
  echo "Document inserted successfully";
?>
```



```
Connection to database successfullyDatabase mydb selectedCollection selected succsessfullyDocument inserted
successfully
```

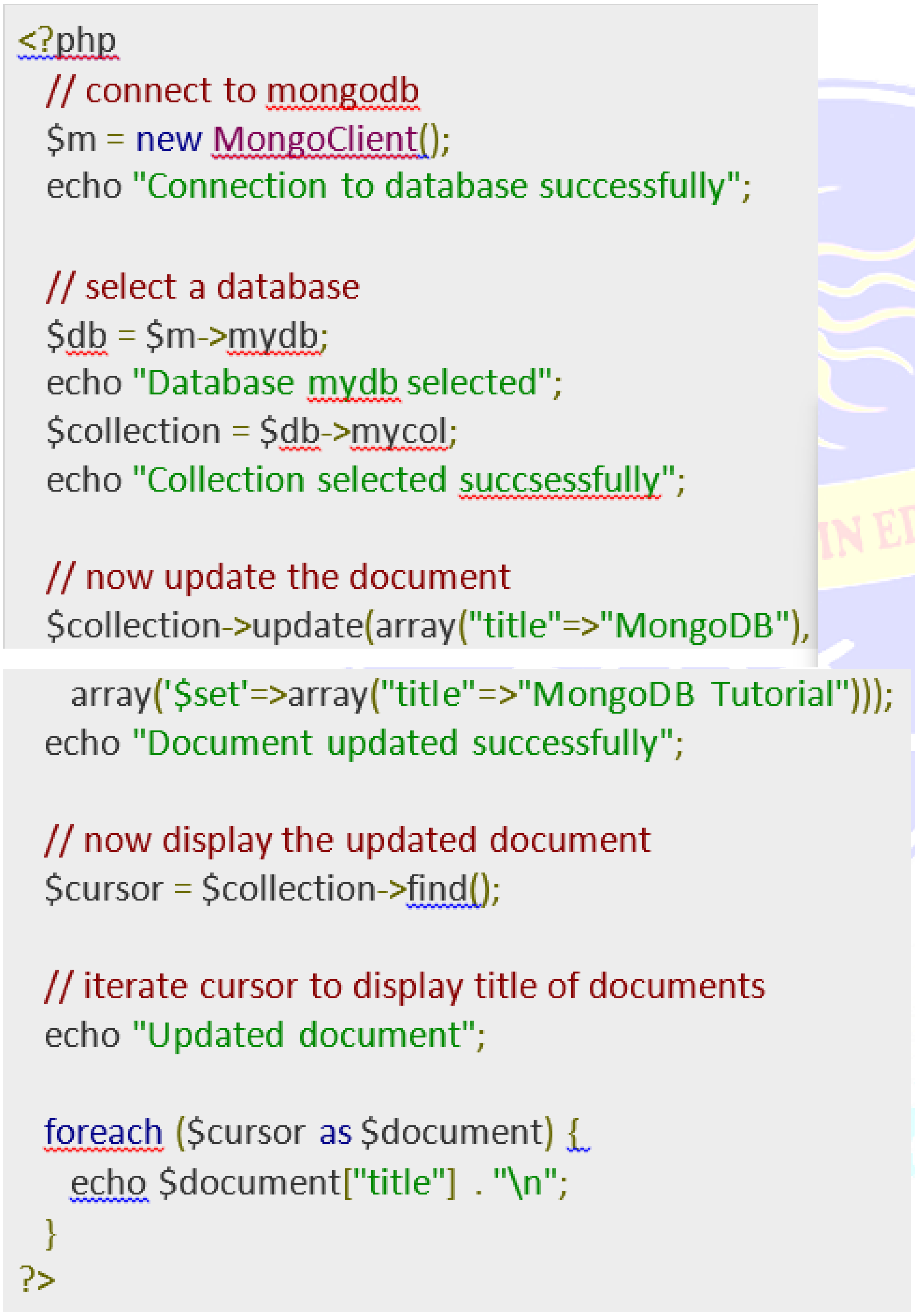
RETRIEVING
Query:-

```
<?php
  // connect to mongodb
  $m = new MongoClient();
  echo "Connection to database successfully";

  // select a database
  $db = $m->mydb;
  echo "Database mydb selected";
  $collection = $db->mycol;
  echo "Collection selected succsessfully";

  $cursor = $collection->find();
  // iterate cursor to display title of documents

  foreach ($cursor as $document) {
    echo $document["title"] . "\n";
  }
?>
```

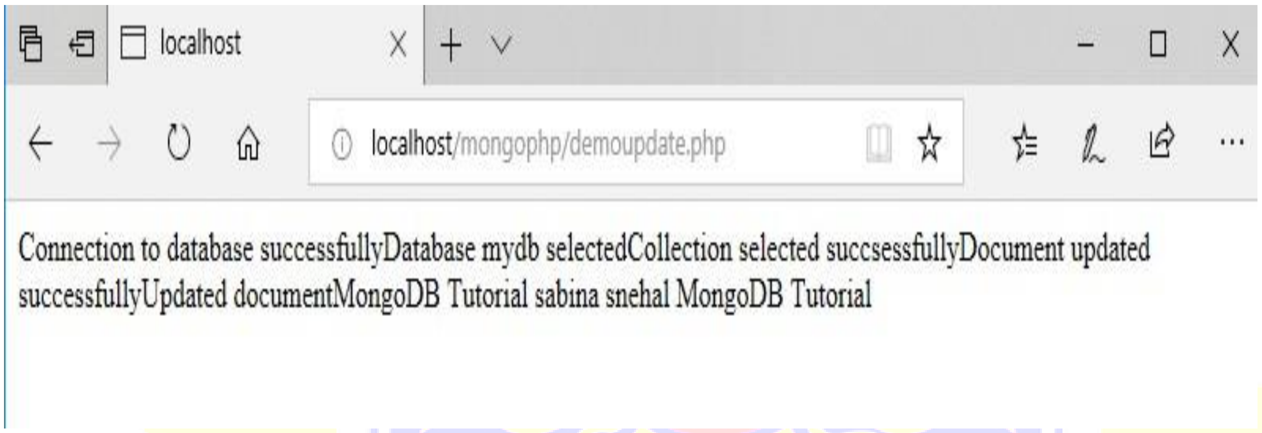


```
Connection to database successfully
Database mydb selected
Collection selected succsessfully
MongoDB Tutorial
sabina
snehal
MongoDB Tutorial
```

UPDATE
Query:-



```
<?php
  // connect to mongodb
  $m = new MongoClient();
  echo "Connection to database successfully";

  // select a database
  $db = $m->mydb;
  echo "Database mydb selected";
  $collection = $db->mycol;
  echo "Collection selected succsessfully";

  // now update the document
  $collection->update(array("title"=>"MongoDB"),
    array('$set'=>array("title"=>"MongoDB Tutorial")));
  echo "Document updated successfully";

  // now display the updated document
  $cursor = $collection->find();

  // iterate cursor to display title of documents
  echo "Updated document";

  foreach ($cursor as $document) {
    echo $document["title"] . "\n";
  }
?>
```



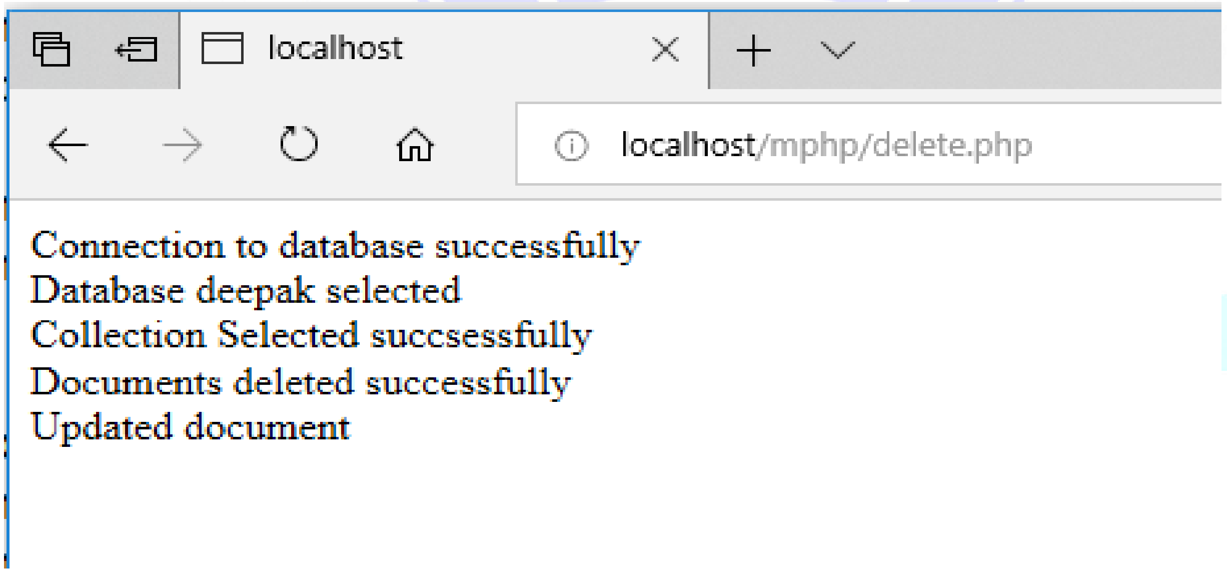
```
Connection to database successfullyDatabase mydb selectedCollection selected succsessfullyDocument updated successfullyUpdated documentMongoDB Tutorial sabina snehal MongoDB Tutorial
```

DELETE

Query:-

```php
<?php
  // connect to mongodb
  $m = new MongoClient();
  echo "Connection to database successfully";

  // select a database
  $db = $m->mydb;
```

```
   echo "Database mydb selected";
   $collection = $db->mycol;
   echo "Collection selected succsessfully";

   // now remove the document
   $collection->remove(array("title"=>"MongoDB Tutorial"),false);
   echo "Documents deleted successfully";

   // now display the available documents
   $cursor = $collection->find();

   // iterate cursor to display title of documents
   echo "Updated document";

   foreach ($cursor as $document) {
     echo $document["title"] . "\n";
   }
?>
```



```
Connection to database successfully
Database deepak selected
Collection Selected succsessfully
Documents deleted successfully
Updated document
```

Learning Outcomes:
By the end of this practical session, participants will be able to:

Understand how to connect a PHP application to a MongoDB database.
Perform basic CRUD (Create, Read, Update, Delete) operations using PHP.
Utilize the MongoDB PHP Library to interact with MongoDB collections.
Apply PHP programming concepts to manage and manipulate data stored in MongoDB.
Course Outcomes:
Upon successful completion of this practical session, participants will be able to:

Implement a PHP application that integrates with MongoDB for data storage and retrieval.
Execute complex queries and data manipulations through PHP code.
Demonstrate the ability to manage MongoDB collections effectively within a PHP environment.
Communicate the principles of using NoSQL databases in PHP applications.
Conclusion:
This practical session has equipped participants with essential skills to effectively use PHP in conjunction with MongoDB. By mastering the integration of these technologies, learners can build dynamic web applications that leverage the power of NoSQL databases. Understanding these concepts will facilitate the development of data-driven applications in real-world scenarios.

Viva Questions on PHP and MongoDB:
What is the purpose of the MongoDB PHP Library?
How do you connect a PHP application to a MongoDB database?
Can you explain how to insert a document into a MongoDB collection using PHP?
What functions are used to retrieve documents from a MongoDB collection in PHP?

PRACTICAL NO :- 7
PYTHON AND MONGODB

Objective:
To understand how to integrate Python applications with MongoDB for data storage and retrieval.

Introduction:
MongoDB is a popular NoSQL database that works well with Python, thanks to the pymongo library. This library allows developers to interact with MongoDB easily and perform various database operations, including creating, reading, updating, and deleting documents.

Key Concepts:
PyMongo:

PyMongo is the official Python driver for MongoDB, enabling Python applications to connect to MongoDB databases and execute commands.
Install PyMongo using pip:
bash
```
pip install pymongo
```
Connecting to MongoDB:

To connect to a MongoDB instance, you create a MongoClient object.
Copy code
```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
db = client['mydatabase']
```

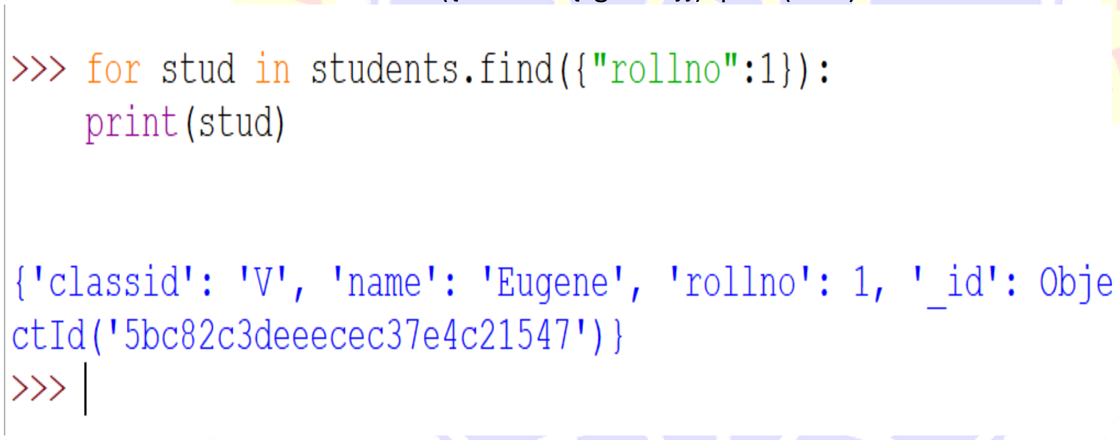
Pratical 7a. Connecting Python with MongoDB and inserting, retrieving, updating and deleting.

```
python -m pip install pymongo
```

INSERT

Query:-

import pymongo
from pymongo import MongoClient
client=MongoClient() db=client.testdb
students=db.students
for stud in students.find({“rollno”:{“gte”:2}}): print(stud)



```
>>> for stud in students.find({"rollno":1}):
    print(stud)


{'classid': 'V', 'name': 'Eugene', 'rollno': 1, '_id': Obje
ctId('5bc82c3deeecec37e4c21547')}
>>> 
```

UPDATE
Query:-

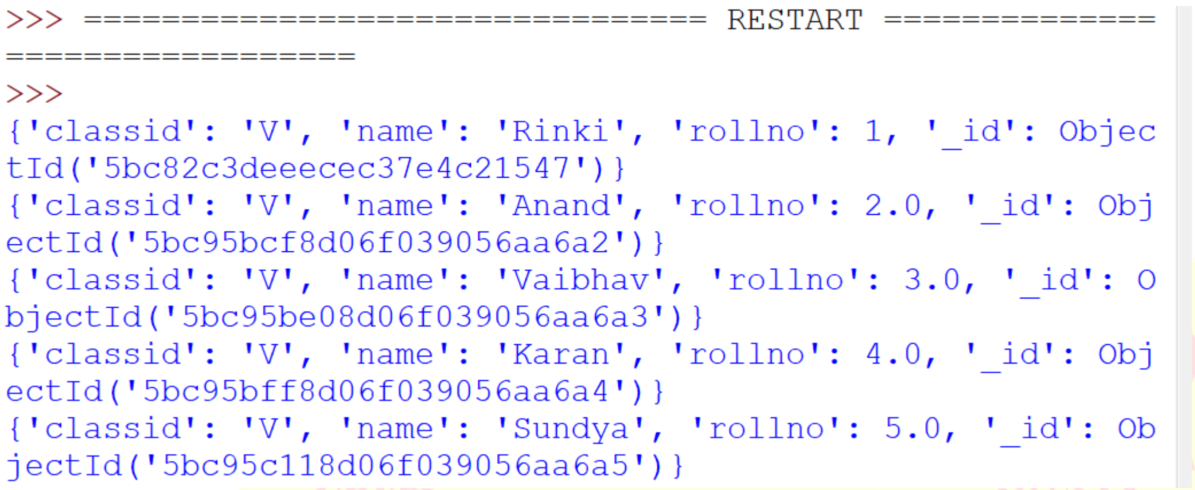
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"] mycol =
mydb["customers"]

myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } } mycol.update_one(myquery, newvalues)

#print "customers" after the update: for x in

```
mycol.find():
  print(x)
```

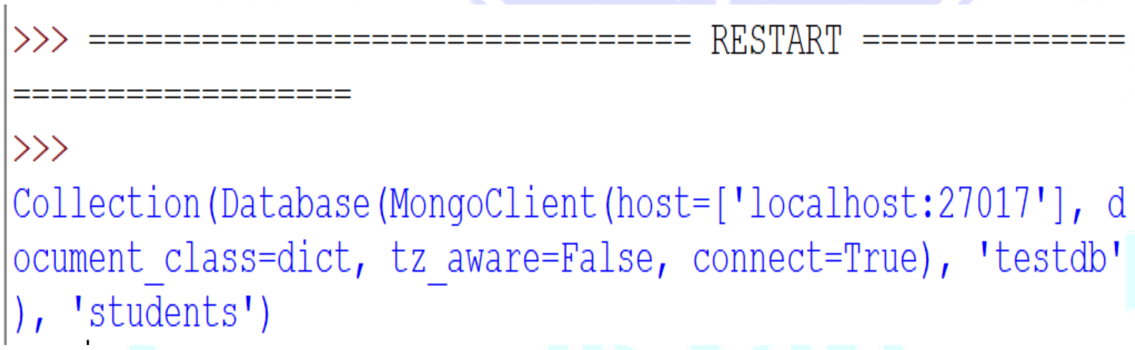


```
>>> ================================ RESTART ==============
==================
>>> 
{'classid': 'V', 'name': 'Rinki', 'rollno': 1, '_id': Objec
tId('5bc82c3deeecec37e4c21547')}
{'classid': 'V', 'name': 'Anand', 'rollno': 2.0, '_id': Obj
ectId('5bc95bcf8d06f039056aa6a2')}
{'classid': 'V', 'name': 'Vaibhav', 'rollno': 3.0, '_id': O
bjectId('5bc95be08d06f039056aa6a3')}
{'classid': 'V', 'name': 'Karan', 'rollno': 4.0, '_id': Obj
ectId('5bc95bff8d06f039056aa6a4')}
{'classid': 'V', 'name': 'Sundya', 'rollno': 5.0, '_id': Ob
jectId('5bc95c118d06f039056aa6a5')}
```

DELETE

Query:-

```
import pymongo
from pymongo import MongoClient
client=MongoClient() db=client.testdb
students=db.students
students.remove({'rollno':3}) print(student)
```



```
>>> ================================ RESTART ==============
==================
>>> 
Collection(Database(MongoClient(host=['localhost:27017'], d
ocument_class=dict, tz_aware=False, connect=True), 'testdb'
), 'students')
```

Learning Outcomes:
By the end of this practical session, participants will be able to:

Understand how to connect a Python application to a MongoDB database.
Perform basic CRUD (Create, Read, Update, Delete) operations using Python and the PyMongo library.
Utilize PyMongo to interact with MongoDB collections effectively.
Apply Python programming concepts to manage and manipulate data stored in MongoDB.
Course Outcomes:
Upon successful completion of this practical session, participants will be able to:

Implement a Python application that integrates with MongoDB for data storage and retrieval.
Execute complex queries and data manipulations through Python code.
Demonstrate the ability to manage MongoDB collections effectively within a Python environment.
Communicate the principles of using NoSQL databases in Python applications.
Conclusion:
This practical session has equipped participants with essential skills to effectively use Python in conjunction with MongoDB. By mastering the integration of these technologies, learners can build dynamic applications that leverage the power of NoSQL databases. Understanding these concepts will facilitate the development of data-driven applications in real-world scenarios.

Viva Questions on Python and MongoDB:
What is the purpose of the PyMongo library in Python?
How do you establish a connection to a MongoDB database using PyMongo?
Can you explain how to insert a document into a MongoDB collection using Python?
What method would you use to retrieve a single document from a MongoDB collection?

PRACTICAL NO :- 8
PROGRAMS ON BASIC JQUERY

Introduction
jQuery is a fast, lightweight, and feature-rich JavaScript library that simplifies HTML document traversal and manipulation, event handling, and animation. It provides an easy-to-use API that works across a multitude of browsers, making it a popular choice for web development.

Key Concepts
Selectors:

jQuery uses CSS-style selectors to select elements in the DOM. For example:

```
$("p") // Selects all <p> elements
$("#myId") // Selects an element with the ID "myId"
$(".myClass") // Selects all elements with the class "myClass"
```
Events:

jQuery provides methods to handle events, such as clicks, key presses, and form submissions. For example:

```
$("#myButton").click(function() {
  alert("Button clicked!");
});
```
DOM Manipulation:

jQuery allows you to modify the content and structure of the DOM easily. For example:

```
$("#myElement").text("New text content"); // Change text
$("#myElement").css("color", "blue"); // Change CSS style
$("#myList").append("<li>New item</li>"); // Add a new item to a list
```
Animations:

jQuery provides methods for creating animations, such as showing, hiding, and fading elements. For example:

```
$("#myElement").fadeIn(); // Fade in an element
$("#myElement").fadeOut(); // Fade out an element
```
AJAX:

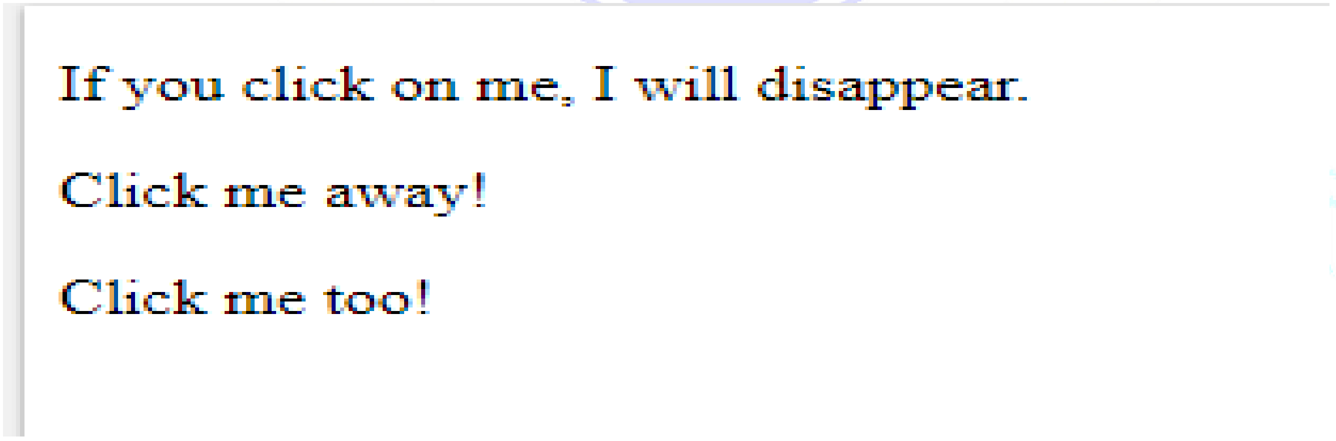
jQuery simplifies AJAX calls, allowing you to load data from the server without refreshing the page. For example:

```
$.ajax({
  url: "data.json",
  method: "GET",
  success: function(data) {
    console.log(data);
  },
  error: function(error) {
    console.error("Error loading data:", error);
  }
});
```

Pratical 8a. jQuery Basic, jQuery Events

click()

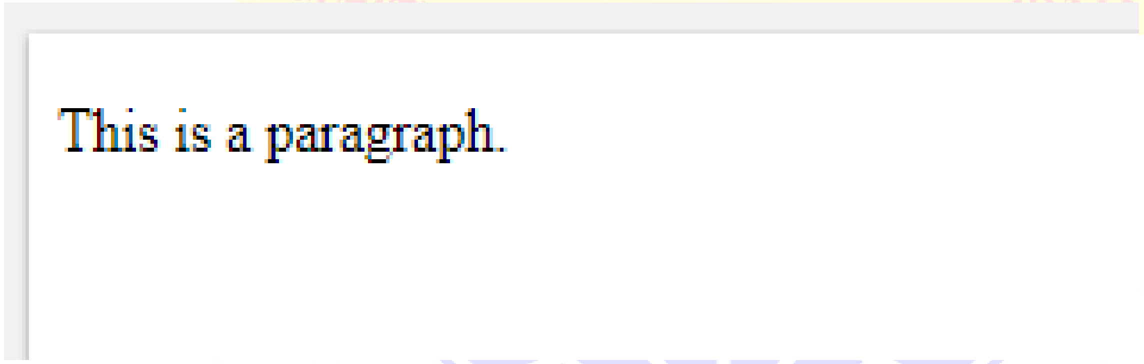
```
<!DOCTYPE html>
<html>
<head>
<script src="file:///C:/js/jquery-3.3.1.min.js"></script>
<script>
$(document).ready(function(){
$("p").click(function(){
$(this).hide();
});
});
</script>
</head>
<body>
<p>If you click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>
</body>
</html>
```



```
If you click on me, I will disappear.

Click me away!

Click me too!
```

mousedown()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("#p1").mousedown(function(){ alert("Mouse down over p1!");
});
});
</script>
</head>
<body>

<p id="p1">This is a paragraph.</p>
</body>
</html>
```

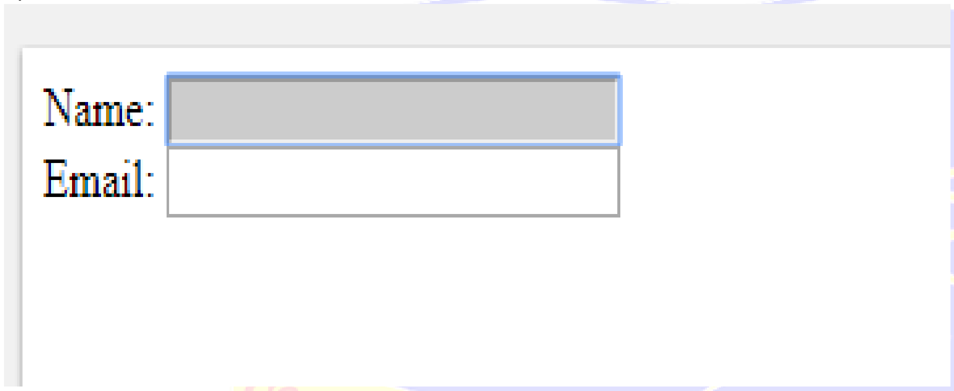


```
This is a paragraph.
```

focus()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("input").focus(function(){
$(this).css("background-color", "#cccccc");
});
$("input").blur(function(){
$(this).css("background-color", "#ffffff");
});
});
</script>
```

```
</head>
<body>

Name: <input type="text" name="fullname"><br> Email: <input type="text" name="email">
</body>
</html>
```



blur()

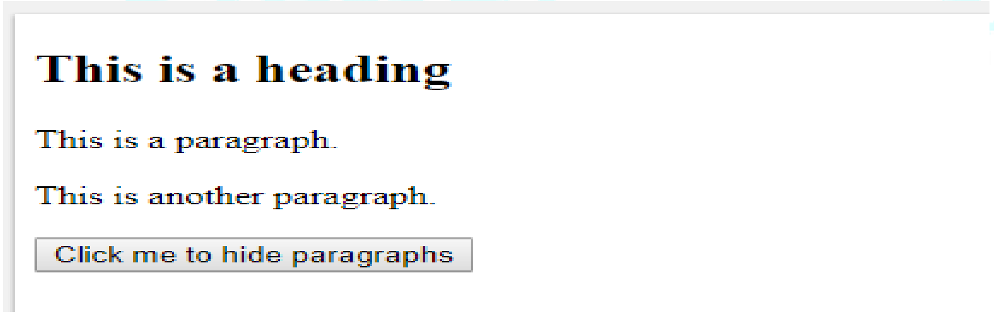
```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("input").focus(function(){
$(this).css("background-color", "#cccccc");
});
$("input").blur(function(){
$(this).css("background-color", "#ffffff");
});
});
</script>
</head>
<body>
Name: <input type="text" name="fullname"><br> Email: <input type="text" name="email">

</body>
</html>
```

Name:
Email:

Pratical 8b. jQuery Selectors, jQuery Hide and Show effects

selectors()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("p").hide();
});
});
</script>
</head>
<body>
<h2>This is a heading</h2>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me to hide paragraphs</button>

</body>
</html>
```

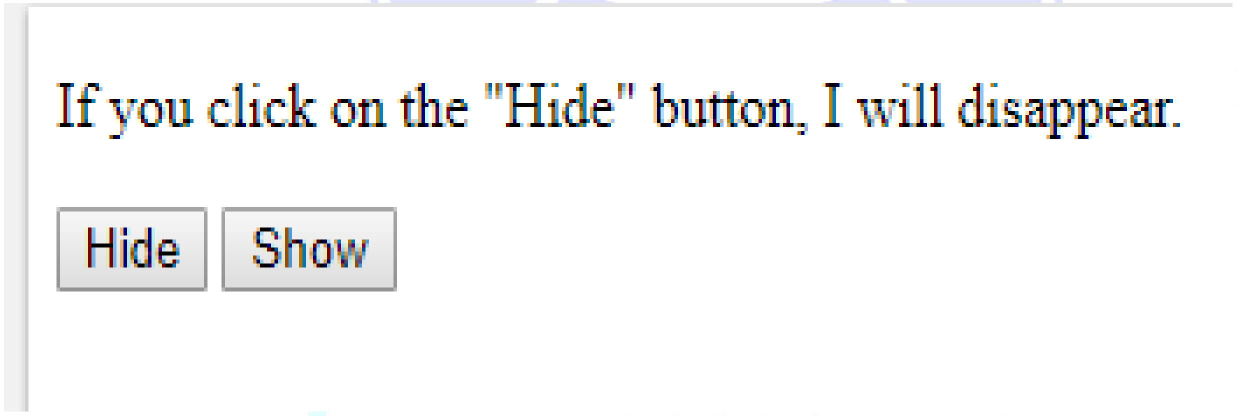

This is a heading

This is a paragraph.

This is another paragraph.

Hide() and show()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("#hide").click(function(){
$("p").hide();
});
$("#show").click(function(){
$("p").show();
});
});
</script>
</head>
<body>
<p>If you click on the "Hide" button, I will disappear.</p>

<button id="hide">Hide</button>
<button id="show">Show</button>
</body>
</html>
```

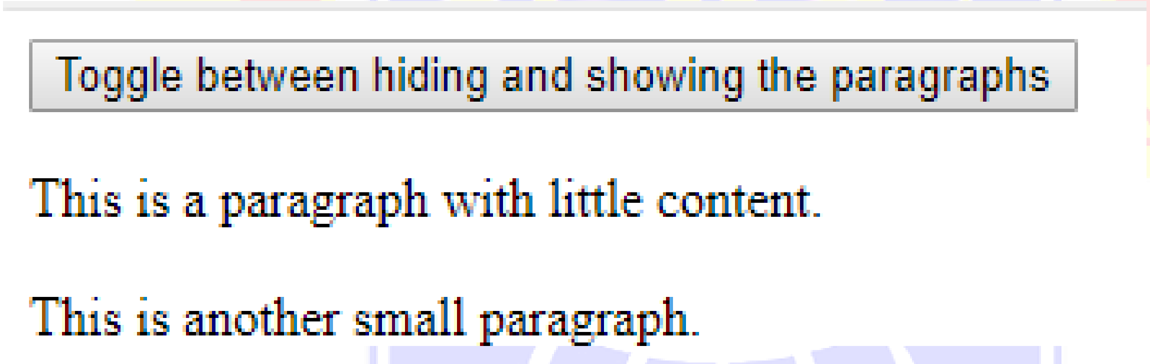


If you click on the "Hide" button, I will disappear.

toggle()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
```

```
$("p").toggle();
});
});
</script>
</head>
<body>
<button>Toggle between hiding and showing the paragraphs</button>

<p>This is a paragraph with little content.</p>
<p>This is another small paragraph.</p>

</body>
</html>
```



This is a paragraph with little content.

This is another small paragraph.

Pratical 8c. jQuery fading effects, jQuery Sliding effects

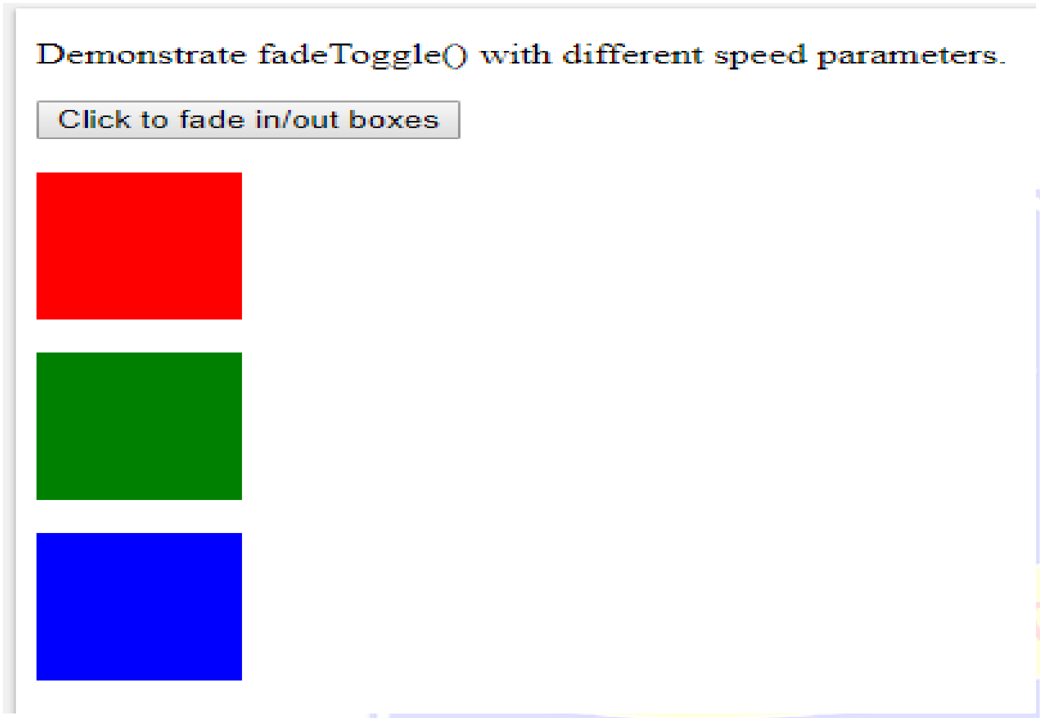
fadeIn()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("#div1").fadeIn();
$("#div2").fadeIn("slow");
$("#div3").fadeIn(3000);
});
});
</script>
</head>
```

```
<body>
<p>Demonstrate fadeIn() with different parameters.</p>

<button>Click to fade in boxes</button><br><br>
<div id="div1" style="width:80px;height:80px;display:none;background- color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;display:none;background- color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;display:none;background- color:blue;"></div>
</body>
</html>
```

fadeToggle()

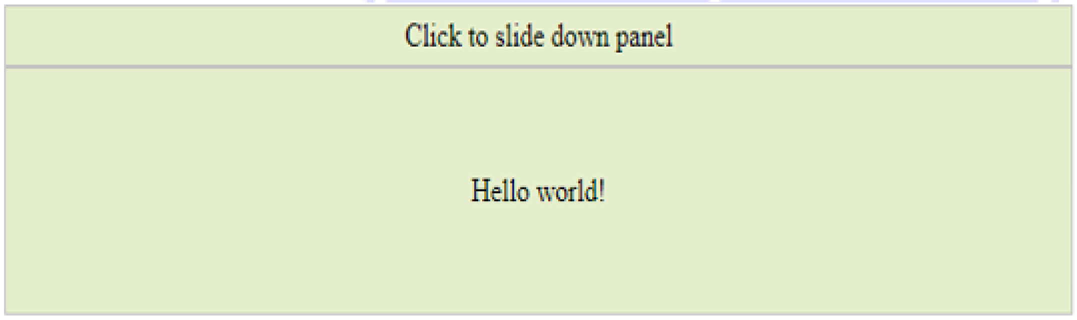
```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("#div1").fadeToggle();
$("#div2").fadeToggle("slow");
$("#div3").fadeToggle(3000);
});
});
</script>
</head>
<body>
<p>Demonstrate fadeToggle() with different speed parameters.</p>

<button>Click to fade in/out boxes</button><br><br>
<div id="div1" style="width:80px;height:80px;background- color:red;"></div>
<br>
<div id="div2" style="width:80px;height:80px;background- color:green;"></div>
<br>
<div id="div3" style="width:80px;height:80px;background- color:blue;"></div>
</body>
</html>
```


Demonstrate fadeToggle() with different speed parameters.

slideDown()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("#flip").click(function(){
$("#panel").slideDown("slow");
});
});
</script>

<style> #panel, #flip {
padding: 5px; text-align: center;
background-color: #e5eecc; border: solid 1px #c3c3c3;
}

#panel { padding: 50px; display: none;
}
</style>
```

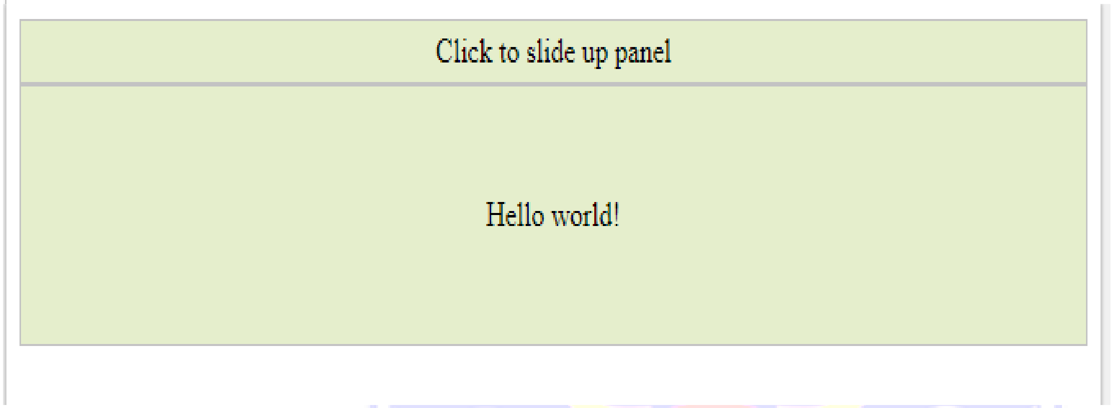
```
</head>
<body>

<div id="flip">Click to slide down panel</div>
<div id="panel">Hello world!</div>
</body>
</html>
```


Click to slide down panel

Hello world!

slideUp()

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("#flip").click(function(){
$("#panel").slideUp("slow");
});
});
</script>
<style> #panel, #flip {
padding: 5px; text-align: center;
background-color: #e5eecc; border: solid 1px #c3c3c3;
}
#panel { padding: 50px;
}
</style>
</head>
<body>

<div id="flip">Click to slide up panel</div>
<div id="panel">Hello world!</div>

</body>
</html>
```


Click to slide up panel

Hello world!

Learning Outcomes:
By the end of this session, participants will be able to:

Understand the fundamentals of jQuery and its syntax.
Implement basic jQuery selectors to manipulate DOM elements.
Handle events using jQuery for user interactions.
Create simple animations and effects using jQuery methods.
Utilize AJAX to load data dynamically from a server.
Course Outcomes:
Upon successful completion of this session, participants will be able to:

Develop interactive web pages using jQuery for DOM manipulation and event handling.
Demonstrate the ability to implement animations and effects to enhance user experience.
Use jQuery to make asynchronous requests to retrieve and display data without refreshing the page.
Identify and troubleshoot common issues in jQuery programming.
Conclusion:
This session has provided participants with a foundational understanding of jQuery and its practical applications in web development. By mastering basic jQuery techniques, learners can create dynamic, user-friendly web applications that enhance interactivity and responsiveness. Continued practice and exploration of jQuery's advanced features will further enrich their web development skills.

Viva Questions on Programs on Basic jQuery:
What is jQuery, and why is it used in web development?
How do you select an element with a specific ID using jQuery?
Can you explain the purpose of the $(document).ready() function?
How can you change the text of a DOM element using jQuery?

PRACTICAL NO :- 9

jQuery Advanced

Introduction
Advanced jQuery techniques enable developers to create more dynamic and responsive web applications. This section delves into advanced features of jQuery, including animations, event delegation, plugins, and AJAX enhancements.

Key Concepts
Chaining:

jQuery allows multiple methods to be called on the same jQuery object, enabling cleaner and more concise code.
Example:

```
$("#myElement").css("color", "red").slideUp(2000).slideDown(2000);
```

Event Delegation:

Instead of binding events directly to elements, event delegation allows you to attach a single event handler to a parent element. This is useful for dynamically added elements.
Example:

```
$("#parentElement").on("click", ".childElement", function() {
   $(this).toggleClass("active");
});
```

Custom Animations:

jQuery provides the ability to create custom animations using animate(). You can define CSS properties to animate over a specified duration.

```
$("#box").animate({ left: '250px', opacity: 0.5 }, 1500);
```

Deferred and Promises:

jQuery supports deferred objects to handle asynchronous operations, allowing you to execute code when a specific operation completes.
Example:

```
var deferred = $.Deferred();

// Simulate an asynchronous operation
setTimeout(function() {
   deferred.resolve("Operation completed!");
}, 1000);

deferred.done(function(message) {
   alert(message);
});
```

Creating Plugins:

jQuery allows developers to create custom plugins to extend its functionality. This involves defining a new method that can be called on jQuery objects.

Example:

```
$.fn.highlight = function(color) {
  this.css("background-color", color);
  return this; // Enable chaining
};

// Usage
$("p").highlight("yellow");
```

AJAX with Promises:

jQuery's AJAX methods return a promise, allowing for more manageable asynchronous code using .then(), .done(), and .fail().

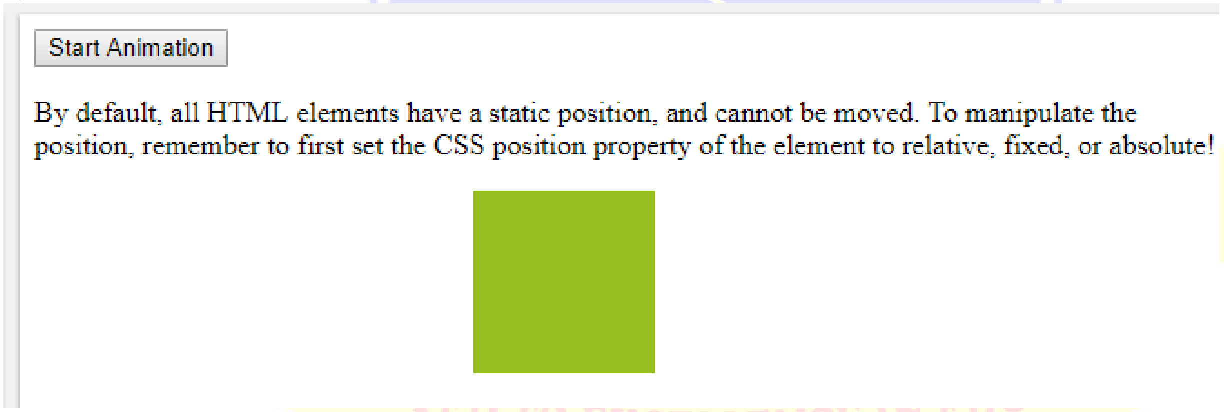
Example:

```
$.ajax({
  url: "data.json",
  method: "GET"
})
.done(function(data) {
  console.log("Data loaded:", data);
})
.fail(function() {
  console.error("Error loading data.");
});
```

Pratical 9a.   jQuery Animation effects, jQuery Chaining

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("div").animate({left: '250px'});
});
});
</script>
</head>
<body>

<button>Start Animation</button>
```

```
<p>By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:100px;position:absol ute;"></div>

</body>
</html>
```



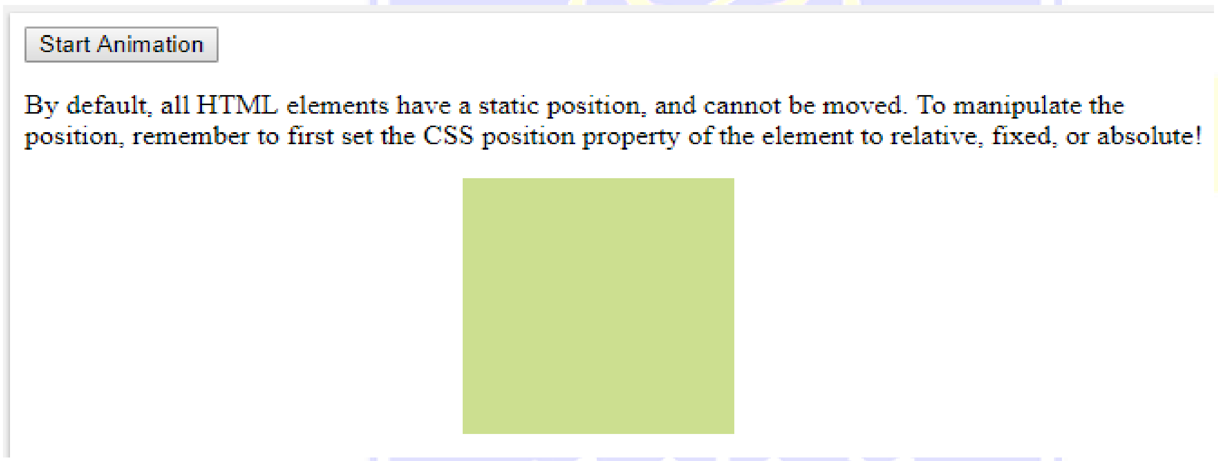
Start Animation

By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

jQuery animate() - Manipulate Multiple Properties

```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("div").animate({ left: '250px',
opacity: '0.5', height: '150px', width: '150px'



});


});

});

</script>
</head>
<body>
```

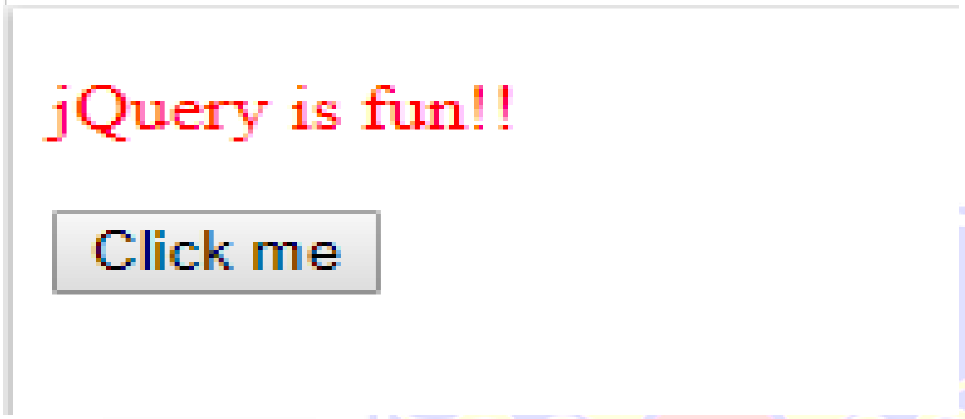
```
<button>Start Animation</button>
<p>By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!</p>

<div style="background:#98bf21;height:100px;width:100px;position:absol ute;"></div>
</body>
</html>
```



By default, all HTML elements have a static position, and cannot be moved. To manipulate the position, remember to first set the CSS position property of the element to relative, fixed, or absolute!

jQuery Method Chaining

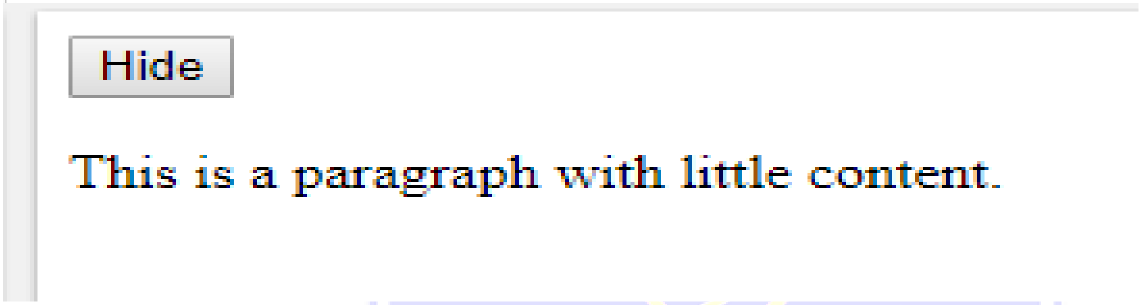
```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
});
});
</script>
</head>
<body>
<p id="p1">jQuery is fun!!</p>
<button>Click me</button>

</body>
</html>
```



Pratical 9b. jQuery Callback, jQuery Get and Set Contents

jQuery Callback Functions

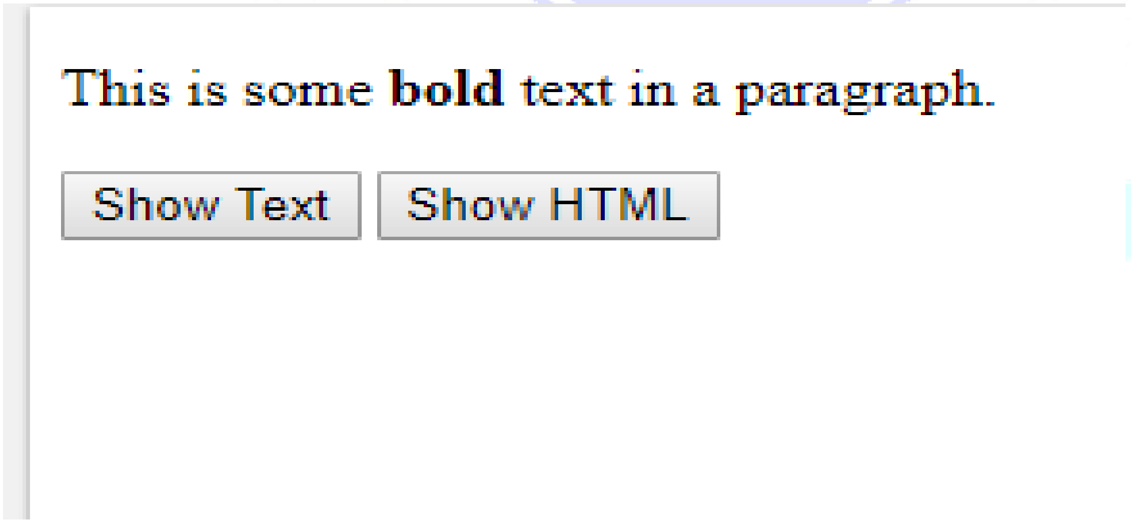
```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("p").hide("slow", function(){ alert("The paragraph is now hidden");


});


});

});

</script>
</head>
<body>
<button>Hide</button>

<p>This is a paragraph with little content.</p>
</body>
</html>
```



This is a paragraph with little content.

Get Content - text()

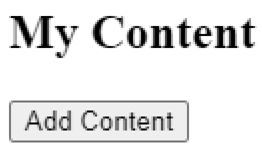
```
<!DOCTYPE html>
<html>
<head>
<script src=" file:///C:/js/jquery-3.3.1.min.js "></script>
<script>
$(document).ready(function(){
$("#btn1").click(function(){ alert("Text: " + $("#test").text());
});
$("#btn2").click(function(){ alert("HTML: " + $("#test").html());
});
});
</script>
</head>
<body>
<p id="test">This is some <b>bold</b> text in a paragraph.</p>
<button id="btn1">Show Text</button>
<button id="btn2">Show HTML</button>
</body>
</html>
```


This is some **bold** text in a paragraph.

Pratical9c:- jQuery Insert Content, jQuery Remove Elements and Attribute

jQuery Insert Content

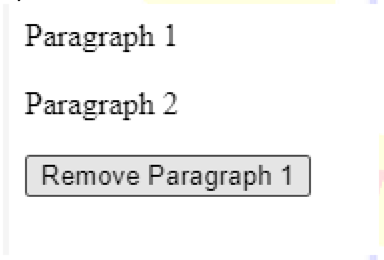
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>jQuery Insert Content Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <div id="content">
    <h2>My Content</h2>
  </div>
  <button id="addContent">Add Content</button>

  <script>
    $("#addContent").click(function() {
      // Insert content after the existing content
      $("#content").append("<p>This is new content added!</p>");
    });
  </script>
</body>
</html>
```


My Content

jQuery Remove Elements

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>jQuery Remove Elements Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <div id="elements">
```

```
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
    <button id="removeElement">Remove Paragraph 1</button>
  </div>

  <script>
    $("#removeElement").click(function() {
      // Remove the first paragraph
      $("#elements p:first").remove();
    });
  </script>
</body>
</html>
```

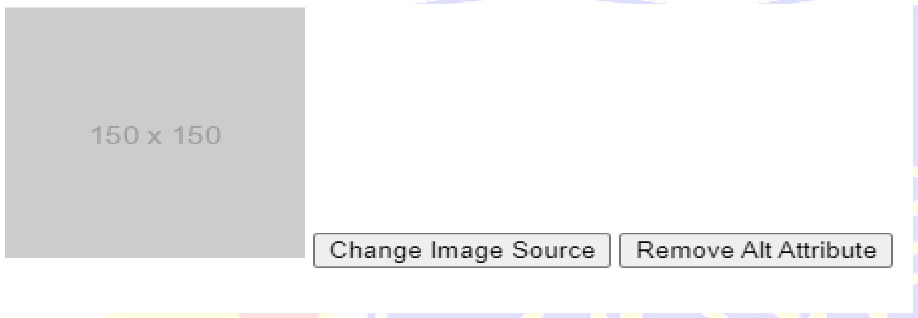

Paragraph 1

Paragraph 2

jQuery Manipulate Attributes

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>jQuery Attribute Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <img id="image" src="https://via.placeholder.com/150" alt="Placeholder Image">
  <button id="changeImage">Change Image Source</button>
  <button id="removeAttr">Remove Alt Attribute</button>

  <script>
    $("#changeImage").click(function() {
      // Change the image source
      $("#image").attr("src", "https://via.placeholder.com/300");
    });

    $("#removeAttr").click(function() {
      // Remove the alt attribute from the image
      $("#image").removeAttr("alt");
```

```
        });
    </script>
</body>
</html>
```


150 x 150
Change Image Source
Remove Alt Attribute

Learning Outcomes:
By the end of this session, participants will be able to:

Understand advanced jQuery features and techniques for DOM manipulation.
Implement event delegation and handle complex user interactions effectively.
Create custom animations and effects using jQuery.
Develop and use jQuery plugins to extend functionality.
Utilize jQuery's AJAX capabilities to perform asynchronous data retrieval and manipulation.
Course Outcomes:
Upon successful completion of this session, participants will be able to:

Write clean, efficient jQuery code using advanced features for web development.
Demonstrate proficiency in handling events and manipulating the DOM dynamically.
Create responsive user interfaces by implementing custom animations.
Build reusable components using jQuery plugins.
Execute AJAX requests to interact with server-side resources and handle responses effectively.
Conclusion:
This session has equipped participants with the knowledge and skills necessary to leverage advanced jQuery features in their web development projects. By mastering techniques such as event delegation, custom animations, and AJAX, developers can create more interactive and user-friendly applications. Continued practice and exploration of jQuery's capabilities will further enhance their web development expertise.

Viva Questions on jQuery Advanced:
What is event delegation in jQuery, and why is it beneficial?
How can you create a custom jQuery plugin, and what are its advantages?
Describe how chaining works in jQuery and provide an example.
Explain the difference between remove() and detach() methods in jQuery.

PRACTICAL NO:-10

JSON

Introduction
JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is primarily used to transmit data between a server and a web application, serving as an alternative to XML.

Key Concepts
Structure:

JSON is built on two structures:
Objects: Unordered sets of key-value pairs enclosed in curly braces {}.
Arrays: Ordered collections of values enclosed in square brackets [].
Data Types:

JSON supports the following data types:
String: A sequence of characters enclosed in double quotes (e.g., "Hello, World!").
Number: Numeric values (e.g., 42, 3.14).
Boolean: true or false.
Null: Represents an empty value (null).
Object: A collection of key-value pairs.
Array: A collection of values.
Syntax Rules:

Keys in JSON objects must be strings enclosed in double quotes.
Values can be strings, numbers, objects, arrays, booleans, or null.
JSON does not support comments.

Practical 10a:- Creating JSON

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand

Example:-
```
{
"employees":[
{"firstName":"John", "lastName":"Doe"},
{"firstName":"Anna", "lastName":"Smith"},
{"firstName":"Peter", "lastName":"Jones"}
]
```

}

Practical 10b: Parsing JSON

file:-

```
<!DOCTYPE html>
<html>
<body>
<h2>Use the XMLHttpRequest to get the content of a file.</h2>
<p>The content is written in JSON format, and can easily be converted into a JavaScript object.</p>

<p id="demo"></p>

<script>

var xmlhttp = new XMLHttpRequest(); xmlhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) { var myObj = JSON.parse(this.responseText);
document.getElementById("demo").innerHTML = myObj.name;
}
};
xmlhttp.open("GET", "json_demo.txt", true); xmlhttp.send();
</script>
<p>Take a look at <a href="json_demo.txt" target="_blank">json_demo.txt</a></p>

</body>
</html>
```

example:-

```
var obj = JSON.parse(text);
```

Learning Outcomes:
By the end of this session, participants will be able to:

Understand the structure and syntax of JSON.
Identify different data types supported by JSON.
Parse and manipulate JSON data in various programming languages.
Utilize JSON for data exchange in web applications and APIs.
Create and validate JSON data for different use cases.
Course Outcomes:
Upon successful completion of this session, participants will be able to:

Demonstrate proficiency in reading and writing JSON data.
Effectively use JSON in conjunction with AJAX for server communication.
Understand the advantages and limitations of using JSON compared to other data formats like XML.
Implement JSON in configuration files and data storage solutions.
Apply best practices for structuring JSON data for clarity and efficiency.
Conclusion:
This session has equipped participants with a comprehensive understanding of JSON as a data interchange format. By mastering its structure, syntax, and practical applications, developers can enhance their ability to work with data in web applications. JSON's lightweight nature and ease of use make it a valuable tool in modern software development.

Viva Questions on JSON:
What is JSON, and why is it commonly used in web development?
Describe the basic structure of a JSON object.
What are the main data types supported by JSON?
How do JSON arrays differ from JSON objects?