EXPERIMENT: No. 1

Page No.
Date 21 6 23

Mongod : The core database server

'mongod' is the primary daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations

Steps to starting mongod :-

1. Open Command prompt
2. Change directory :-
   c:> cd c:\ProgramFiles\MongoDB Server\3.6\bin
3. Start server
   c:\Program Files\MongoDB\Server\3.6\bin> mongod --dbpath D:\TYITB3\data\

Mongo : The database Shell

The shell provides a full database interface for MongoDB enabling you play around with the data stored in MongoDB

Steps to starting mongo shell.

1 Open new command prompt
2. Change directory :
   C:> cd c:\Program Files\MongoDB\Server\3.6\bin
3. Start shell
   c:\Program Files\MongoDB\Server\3.6\bin> mongo

To connect to different hosts and ports, --host and --port can be used along with the command By default it connect to --host localhost --port 27017

Teacher's Sign. :

For Server in CMD

```
C:\Users\students.PC1>cd C:\Program Files\MongoDB\Server\3.6\bin
C:\Program Files\MongoDB\Server\3.6\bin>mongod --dbpath D:\TYITB3\DATA\DB
```

For Client in CMD (Mongo)

```
C:\Users\students.PC1>cd C:\Program Files\MongoDB\Server\3.6\bin
C:\Program Files\MongoDB\Server\3.6\bin>mongo
> db
test
> use tyitb3
switched to db tyitb3
> db.createCollection("user")
{ "ok" : 1 }
> db.createCollection("student")
{ "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
tyitb3  0.000GB
> db
tyitb3
> show collections
student
user
> db.user.insert({"Name":"Jayesh","Age":21,"Gender":"M"})
WriteResult({ "nInserted" : 1 })
> db.user.insert({"Name":"Sarah","Age":23,"Gender":"F"})
WriteResult({ "nInserted" : 1 })
> db.user.insert({"Name":"Rengoku","Age":23,"Gender":"M"})
WriteResult({ "nInserted" : 1 })
```

For Client in CMD using JSON file

```
C:\Program Files\MongoDB\Server\3.6\bin>mongoimport --db tyitb3 --collection student < D:\TYITB3\student.json
2023-06-23T15:53:31.914+0530    connected to: localhost
2023-06-23T15:53:31.915+0530    imported 10 documents
```

For Client in CMD (Mongo)

Find

```
> db.student.find()
{ "_id" : ObjectId("649572a3d109b2c6a1567f50"), "Name" : "Jayesh", "RollNo" : 3, "Class" : "TYIT", "CGPA" : 9.6 }
.....
> db.user.find()
{ "_id" : ObjectId("64956bd138d1e81b1aa61cfc"), "Name" : "Jayesh", "Age" : 21, "Gender" : "M" }
.....
> db.student.find({"Name":"Jayesh"})
{ "_id" : ObjectId("649572a3d109b2c6a1567f50"), "Name" : "Jayesh", "RollNo" : 3, "Class" : "TYIT", "CGPA" : 9.6 }
> db.student.find({"Name":"Jayesh"},{"RollNo":1,"CGPA":1})
{ "_id" : ObjectId("649572a3d109b2c6a1567f50"), "RollNo" : 3, "CGPA" : 9.6 }
> db.student.find({"Name":"Jayesh"},{"RollNo":1,"CGPA":1,"_id":0})
{ "RollNo" : 3, "CGPA" : 9.6 }
```

Update

```
> db.student.update({"Name":"Dipak"},{$set:{"Class":"SYIT"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.update({"Class":"SYIT"},{$set:{"Class":"Graduate"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.student.update({"Class":"SYIT"},{$set:{"Class":"Graduate"}},{multi:true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
> db.student.update({"Class":"Graduate"},{$unset:{"Class":""}},{multi:true})
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
> db.student.update({},{$unset:{"Class":""}},{multi:true})
WriteResult({ "nMatched" : 10, "nUpserted" : 0, "nModified" : 7 })
```

Delete

```
> db.student.remove({})
WriteResult({ "nRemoved" : 10 })
> db.student.drop()
true
> show collections
user
> db.dropDatabase()
{ "dropped" : "tyitb3", "ok" : 1 }
```

EXPERIMENT : No. 2 MongoDB Basic Queries

Page No. Date

1. Query documents can be passed as parameter to the find() method to filter documents within a collection
2. Using the find() command without any query document or an empty query document such as find({}) return all the document within the collection.
3. A query document can contain "selectors" and "projectors"
4. A selectors is like a where condition in SQL or a filter but is used to filter out the result.
5. A projector is like the select condition or the selection list that is used to display the data fields

Examples :

find() method parameters

1) selectors

```
>db.user.find({"Gender":"f"})
>db.user.find({"Gender":"f", $or[{"Country":"India"}]})
>db.user.find({"Gender":"f", $or[{"Country":"India"},
                                {"Country":"USA"}]})
```

2) Projectors :

```
>db.user.find({"Gender":"f"}, {"Name":1, "Age":1})
```

Sundaram

Teacher's Sign.

Practical No	Details
2	MongoDB simple queries

Note: start server and shell. Import given database file.

Server (Mongod.exe)

1. Before starting the server make sure following things:
    a. Path of bin directory (located in C:\Program Files\MongoDB\Server\4.2\bin)
    b. Path of db directory (you must explicitly create this directory in f:\data\db)
2. Open command prompt
3. Change directory :

C:>cd C:\Program Files\MongoDB\Server\4.2\bin

4. Start server:

C:\Program Files\……..\bin\>mongod.exe –dbpath f:\data\db

Shell (Mongo.exe)

1. Open new command prompt
2. Change directory : **C:>cd C:\Program Files\MongoDB\Server\4.2\bin**
3. Start shell: C:\Program Files\ ……..\bin**>mongo.exe**

Import given "restaurants.json" file (mongoimport.exe)

1. Open new command prompt
2. Change directory : **C:>cd C:\Program Files\MongoDB\Server\4.2\bin**
3. C:\……..\bin>mongoimport --db rest --collection rest < f:\ restaurants.json

2020-11-03T10:45:19.770+0530 connected to: mongodb://localhost/

2020-11-03T10:45:20.282+0530 3772 document(s) imported successfully.

0 document(s) failed to import.

Execute following commands on shell.

>use rest

//Switched to database rest

1.	Write a MongoDB query to display all the documents in the collection restaurant.
	db.rest.find();
2.	Write a MongoDB query to display the fields, restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.
	db.rest.find({},{"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1});
3.	Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine, but exclude the field _id for all the documents in the collection restaurant
	db.rest.find({},{"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1,"_id":0});
4.	Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant
	db.rest.find({},{"restaurant_id" : 1,"name":1,"borough":1,"address.zipcode" :1, "_id":0});
5.	Write a MongoDB query to display all the restaurant which is in the borough Bronx
	db.rest.find({"borough": "Bronx"});
6.	Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.
	db.rest.find({"borough": "Bronx"}).limit(5);
7.	Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx.
	db.rest.find({"borough": "Bronx"}).skip(5).limit(5);
8.	Write a MongoDB query to find the restaurants who achieved a score more than 90
	db.rest.find({grades : { $elemMatch:{"score":{$gt : 90}}}});
9.	Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100
	db.rest.find({"grades.score":{$gt : 80 , $lt :100}}});
10.	Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168
	db.rest.find({"address.coord" : {$lt : -95.754168}});
11.	Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168
	db.rest.find({$and: [{"cuisine" : {$ne :"American "}}, {"grades.score" : {$gt : 70}}, {"address.coord" : {$lt : -65.754168}}] });

12.	Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168. Note : Do this query without using $and operator.
	db.rest.find({ "cuisine" : {$ne : "American "}, "grades.score" :{$gt: 70}, "address.coord" : {$lt : -65.754168} });
13.	Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American ' and achieved a grade point 'A' , belongs to the borough Brooklyn. The document must be displayed according to the cuisine in descending order
	db.rest.find({ "cuisine" : {$ne : "American "}, "grades.grade" :"A", "borough": "Brooklyn" }).sort({"cuisine":-1});
14	Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name.
	db.rest.find({name: /^Wil/}, { "restaurant_id" : 1, "name":1,"borough":1, "cuisine" :1 });
15.	Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'ces' as last three letters for its name.
	db.rest.find({name: /ces$/}, { "restaurant_id" : 1, "name":1,"borough":1, "cuisine" :1 });
16.	Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Reg' as three letters somewhere in its name
	db.rest.find({"name": /.*Reg.*/}, { "restaurant_id" : 1, "name":1,"borough":1, "cuisine" :1 });
17.	Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.
	db.rest.find({ "borough": "Bronx" , $or : [{ "cuisine" : "American " }, { "cuisine" : "Chinese" }]});
18.	Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronx or Brooklyn
	db.rest.find({"borough" :{$in :["Staten Island","Queens","Bronx","Brooklyn"]}}, { "restaurant_id" : 1, "name":1,"borough":1, "cuisine" :1 });
19.	Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronx or Brooklyn.
	db.rest.find({"borough" :{$nin :["Staten Island","Queens","Bronx","Brooklyn"]}}, { "restaurant_id" : 1, "name":1,"borough":1, "cuisine" :1 });

20.	Write a MongoDB query to find the restaurant Id, name, borough and grades score for those restaurants which achieved a score which is not more than 10.
	db.rest.find({"grades.score" : { $not: {$gt : 10} } }, { "restaurant_id" : 1, "name":1,"borough":1, "grades.score" :1 });
21.	Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'
	db.rest.find({$or: [{name: /^Wil/}, {"$and": [{"cuisine" : {$ne :"American "}}, {"cuisine" : {$ne :"Chinees"}}]}]} ,{"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1});
22.	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.
	db.rest.find({ "grades.date": ISODate("2014-08-11T00:00:00Z"), "grades.grade":"A" , "grades.score" : 11 }, {"restaurant_id" : 1,"name":1,"grades":1});
23.	Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z"
	db.rest.find({ "grades.1.date": ISODate("2014-08-11T00:00:00Z"), "grades.1.grade":"A" , "grades.1.score" : 9 }, {"restaurant_id" : 1,"name":1,"grades":1});
24	Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52.
	db.rest.find({ "address.coord.1": {$gt : 42, $lte : 52} }, {"restaurant_id" : 1,"name":1,"address":1,"coord":1});
25.	Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.
	db.rest.find().sort({"name":1});
26.	Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns
	db.rest.find().sort({"name":-1});
27	Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order

	db.rest.find().sort({"cuisine" :1,"borough" : -1,});
28	Write a MongoDB query to know whether all the addresses contains the street or not.
	db.rest.find({"address.street" : { $exists : true } });
29	Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double
	db.rest.find({"address.coord" : {$type : 1} });
30	Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.
	db.rest.find({"grades.score" : {$mod : [7,0]} }, {"restaurant_id" : 1,"name":1,"grades":1});
31.	Write a MongoDB query to find the restaurant name, borough, longitude and lattitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name
	db.rest.find({ name : { $regex : "mon.*", $options: "i" } }, { "name":1, "borough":1, "address.coord":1, "cuisine" :1 });
32.	Write a MongoDB query to find the restaurant name, borough, longitude and latitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name
	db.rest.find({ name : { $regex : /^Mad/i, } }, { "name":1, "borough":1, "address.coord":1, "cuisine" :1 });

EXPERIMENT No. 3 Aggregate Functions

Page No.
Date

aggregate() :-

The aggregate framework enables you to find out the aggregate value without using the Map Reduce function. Performance wise, the aggregation function framework is faster than the MapReduce Reduction function.

Examples :-

1) Find the count of male and female students:

>db.students.aggregate([{$group:{_id:"$Gender", totalStudent:{"$sum":1}}}])

2) Find out the class wise average score

>db.students.aggregate([{$group:{_id:"$class", AvgScore:{$avg:"$studentscore"}}}])

Functions of aggregate() :-

1. $group stage and $project stage :- Work with one of stage
2. $avg : Calculate & returns the average of numeric values.
3. $sum : Calculate & returns the sum of numeric values
4. $min : Returns the minimum value
5. $max : Returns the maximum value
6. $addToSet :- Returns an array of all unique values that results from applying an expression to each document in a group of document that share the same group by key.
7. $push : Returns an array of all values that results from Applying an expression to each document in a group of documents that share the same group by key
8. $first : Returns the value that results from applying an expression to the first document in a group of documents that share the same group by key.
9. $last : Returns the value that results from applying an expression to the last document in a group of document that shares the same group by a field.

Teacher's Sign

Name:Bharat Rai Roll no:87

```
C:\Program Files\MongoDB\Server\3.6\bin>mongoimport --db tyit87 --collection emp < D:\tyit87\employee.json
2023-07-10T15:25:35.496+0530    connected to: localhost
2023-07-10T15:25:35.621+0530    imported 10 documents

> use tyit87
switched to db tyit87
> db.emp.find()
{ "_id" : 1004, "ename" : "mickey", "salary" : 70000, "city" : "tokyo", "dob" : ISODate("2007-08-20T04:55:12Z") }
{ "_id" : 1002, "ename" : "sanoj", "salary" : 10000, "city" : "jharkhand", "dob" : ISODate("2005-10-20T02:55:58Z") }
{ "_id" : 1005, "ename" : "kisaki", "salary" : 16000, "city" : "paris", "dob" : ISODate("2007-07-20T05:55:54Z") }
{ "_id" : 1006, "ename" : "hanmesh", "salary" : 30000, "city" : "up", "dob" : ISODate("2008-06-20T06:55:34Z") }
{ "_id" : 1008, "ename" : "sam", "salary" : 15000, "city" : "srinagar", "dob" : ISODate("2010-04-20T08:55:33Z") }
{ "_id" : 1007, "ename" : "zoro", "salary" : 90000, "city" : "newyork", "dob" : ISODate("2009-05-20T07:55:45Z") }
{ "_id" : 1009, "ename" : "eninem", "salary" : 40000, "city" : "bhopal", "dob" : ISODate("2011-03-20T09:55:09Z") }
{ "_id" : 1001, "ename" : "manoj", "salary" : 20000, "city" : "delhi", "dob" : ISODate("2003-11-20T01:55:00Z") }
{ "_id" : 1003, "ename" : "luffy", "salary" : 50000, "city" : "mumbai", "dob" : ISODate("2006-09-20T03:55:56Z") }
{ "_id" : 1010, "ename" : "raymond", "salary" : 100000, "city" : "mumbai", "dob" : ISODate("2013-01-
20T10:55:12Z") }

Calculate aggregate function
> db.emp.aggregate({$group:{"_id":"$city","salsum":{$sum:"$salary"}}});
{ "_id" : "bhopal", "salsum" : 40000 }
{ "_id" : "newyork", "salsum" : 90000 }
{ "_id" : "delhi", "salsum" : 20000 }
{ "_id" : "up", "salsum" : 30000 }
{ "_id" : "mumbai", "salsum" : 150000 }
{ "_id" : "paris", "salsum" : 16000 }
{ "_id" : "jharkhand", "salsum" : 10000 }
{ "_id" : "srinagar", "salsum" : 15000 }
{ "_id" : "tokyo", "salsum" : 70000 }
{ "_id" : "tokyo", "salsum" : 70000 }
> db.emp.aggregate({$group:{"_id":"$city","salmax":{$max:"$salary"}}});
{ "_id" : "bhopal", "salmax" : 40000 }
{ "_id" : "newyork", "salmax" : 90000 }
{ "_id" : "delhi", "salmax" : 20000 }
{ "_id" : "up", "salmax" : 30000 }
{ "_id" : "mumbai", "salmax" : 100000 }
{ "_id" : "paris", "salmax" : 16000 }
{ "_id" : "jharkhand", "salmax" : 10000 }
{ "_id" : "srinagar", "salmax" : 15000 }
{ "_id" : "tokyo", "salmax" : 70000 }
> db.emp.aggregate({$group:{"_id":"$city","salmin":{$min:"$salary"}}});
{ "_id" : "bhopal", "salmin" : 40000 }
{ "_id" : "newyork", "salmin" : 90000 }
{ "_id" : "delhi", "salmin" : 20000 }
{ "_id" : "up", "salmin" : 30000 }
{ "_id" : "mumbai", "salmin" : 50000 }
{ "_id" : "paris", "salmin" : 16000 }
{ "_id" : "jharkhand", "salmin" : 10000 }
{ "_id" : "srinagar", "salmin" : 15000 }
{ "_id" : "tokyo", "salmin" : 70000 }
```

```
> db.emp.aggregate({$group:{"_id":"$city","salavg":{$avg:"$salary"}}});
{ "_id" : "bhopal", "salavg" : 40000 }
{ "_id" : "newyork", "salavg" : 90000 }
{ "_id" : "delhi", "salavg" : 20000 }
{ "_id" : "up", "salavg" : 30000 }
{ "_id" : "mumbai", "salavg" : 75000 }
{ "_id" : "paris", "salavg" : 16000 }
{ "_id" : "jharkhand", "salavg" : 10000 }
{ "_id" : "srinagar", "salavg" : 15000 }
{ "_id" : "tokyo", "salavg" : 70000 }
> db.emp.aggregate({$group:{"_id":"$city","list":{$push:{"name":"$ename","salary":"$salary"}}}})
{ "_id" : "bhopal", "list" : [ { "name" : "eninem", "salary" : 40000 } ] }
{ "_id" : "newyork", "list" : [ { "name" : "zoro", "salary" : 90000 } ] }
{ "_id" : "delhi", "list" : [ { "name" : "manoj", "salary" : 20000 } ] }
{ "_id" : "up", "list" : [ { "name" : "hanmesh", "salary" : 30000 } ] }
{ "_id" : "mumbai", "list" : [ { "name" : "luffy", "salary" : 50000 }, { "name" : "raymond", "salary" : 100000 } ] }
{ "_id" : "paris", "list" : [ { "name" : "kisaki", "salary" : 16000 } ] }
{ "_id" : "jharkhand", "list" : [ { "name" : "sanoj", "salary" : 10000 } ] }
{ "_id" : "srinagar", "list" : [ { "name" : "sam", "salary" : 15000 } ] }
{ "_id" : "tokyo", "list" : [ { "name" : "mickey", "salary" : 70000 } ] }
> db.emp.aggregate({$group:{"_id":"$city","list":{"$addToSet":"$ename"}}})
{ "_id" : "bhopal", "list" : [ "eninem" ] }
{ "_id" : "newyork", "list" : [ "zoro" ] }
{ "_id" : "delhi", "list" : [ "manoj" ] }
{ "_id" : "up", "list" : [ "hanmesh" ] }
{ "_id" : "mumbai", "list" : [ "raymond", "luffy" ] }
{ "_id" : "paris", "list" : [ "kisaki" ] }
{ "_id" : "jharkhand", "list" : [ "sanoj" ] }
{ "_id" : "srinagar", "list" : [ "sam" ] }
{ "_id" : "tokyo", "list" : [ "mickey" ] }
> db.emp.aggregate({$sort:{"dob":-1}},{$group:{"_id":"$city","youngest":{"$first":
{ "_id" : "tokyo", "youngest" : ISODate("2007-08-20T04:55:12Z") }
{ "_id" : "up", "youngest" : ISODate("2008-06-20T06:55:34Z") }
{ "_id" : "delhi", "youngest" : ISODate("2003-11-20T01:55:00Z") }
{ "_id" : "newyork", "youngest" : ISODate("2009-05-20T07:55:45Z") }
{ "_id" : "srinagar", "youngest" : ISODate("2010-04-20T08:55:33Z") }
{ "_id" : "jharkhand", "youngest" : ISODate("2005-10-20T02:55:58Z") }
{ "_id" : "bhopal", "youngest" : ISODate("2011-03-20T09:55:09Z") }
{ "_id" : "mumbai", "youngest" : ISODate("2013-01-20T10:55:12Z") }
{ "_id" : "paris", "youngest" : ISODate("2007-07-20T05:55:54Z") }
> db.emp.aggregate({$sort:{"dob":-1}},{$group:{"_id":"$city","oldest":{"$last":
{ "_id" : "tokyo", "oldest" : ISODate("2007-08-20T04:55:12Z") }
{ "_id" : "up", "oldest" : ISODate("2008-06-20T06:55:34Z") }
{ "_id" : "delhi", "oldest" : ISODate("2003-11-20T01:55:00Z") }
{ "_id" : "newyork", "oldest" : ISODate("2009-05-20T07:55:45Z") }
{ "_id" : "srinagar", "oldest" : ISODate("2010-04-20T08:55:33Z") }
{ "_id" : "jharkhand", "oldest" : ISODate("2005-10-20T02:55:58Z") }
{ "_id" : "bhopal", "oldest" : ISODate("2011-03-20T09:55:09Z") }
{ "_id" : "mumbai", "oldest" : ISODate("2006-09-20T03:55:56Z") }
{ "_id" : "paris", "oldest" : ISODate("2007-07-20T05:55:54Z") }
```

EXPERIMENT : No. 4

Page No.
Date

Replica Set

1. Setting up a Replica Set
2. Removing a Server
3. Adding a server
4. Adding an arbitor
5. Inspecting the Status
6. Forcing a new election of a primary
7. Using the web interface to inspect the status of the replica set

The Following examples assume a Replica set name "set2" that was the configuration show in Table:

Member_Data	Daemon	Host Port	File Path
Active_Member_1	Mongod	[host name]: 27071	D:\db\active1\data
Active_Member_2	Mongod	[host name]: 27072	D:\db\active2\data
Passive_Member_3	Mongod	[host name] : 27073	D:\db\passive1\data

Backup and Restore

mongodump - This utility is used as part of an effective backup strategy. It creates a binary export of the database contents

mongorestore - The binary database dump created by the mongodump utility is imported to new or an existing database using the mongorestore utility.

Teacher's Sign :

5.a] Write a MongoDB query to create Replica of existing database.

New command prompt to start a server 27041

```
C:\Program Files\MongoDB\Server\4.2\bin>mongod --dbpath d:\db1\active1\data --port 27041 --replSet setb1
```

New command prompt to start a server 27042

```
C:\Program Files\MongoDB\Server\4.2\bin>mongod --dbpath d:\db1\active2\data --port 27042 --replSet setb1
```

New command prompt to start a server 27043

```
C:\Program Files\MongoDB\Server\4.2\bin>mongod --dbpath d:\db1\passive1\data --port 27043 --replSet setb1
```

New command prompt to start a shell 27041

```
C:\Program Files\MongoDB\Server\4.2\bin>mongo --port 27041
> db
test
//create variable con and add member 27041 server details
> con={"_id":"setb1", "members":[{"_id":0, "host":"127.0.0.1:27041"}]}
//Initiate replica set
> rs.initiate(con)
o/p:
{
"ok" : 1,.....
//Edit variable con and add other members (27042, 27043) server details
set2:SECONDARY> con={"_id":"setb1", "members":[{"_id":0, "host":"127.0.0.1:27041"},
{"_id":1, "host":"127.0.0.1:27042"},
{"_id":2, "host":"127.0.0.1:27043"}]}
//check 27041 is master?
set2:PRIMARY> rs.isMaster()
//New command prompt to start a server 27044
C:\Program Files\MongoDB\Server\4.2\bin>mongod --dbpath d:\db1\active3\data --port 27044 --replSet setb1
//Switch/open 27041 shell command prompt
set2:PRIMARY> use admin
//add new member 27042/27043/27044 to replica set using rs.add command
set2:PRIMARY> rs.add("127.0.0.1:27042")
{
"ok" : 1,.....
set2:PRIMARY> rs.add("127.0.0.1:27043")
{
"ok" : 1,.....

set2:PRIMARY> rs.add("127.0.0.1:27044")
{
"ok" : 1,.....
Check status
set2:PRIMARY> rs.status()
//new command prompt to start a shell 27042
C:\Program Files\MongoDB\Server\4.2\bin>mongo --port 27042
//check 27042 is master?
```

```
set2:SECONDARY> rs.isMaster()
//Switch/open shell 27041 command prompt
set2:PRIMARY> use admin
//remove member 27042 from replica set using rs.remove command
set2:PRIMARY> rs.remove("127.0.0.1:27042")
{
"ok" : 1,.....
//Check status
set2:PRIMARY> rs.status()
set2:PRIMARY> rs.stepDown()
{
"ok" : 1,.....
//You can see now 27041 became secondary
set2:SECONDARY>
new command prompt to start a shell 27043
C:\Program Files\MongoDB\Server\4.2\bin>mongo --port 27044
Insert and read some documents in new primary (27042/27043/27044)
set2:PRIMARY> use user
set2:PRIMARY> db.user.insert({"name":"abc","rollno":1})
WriteResult({ "nInserted" : 1 })
set2:PRIMARY> db.user.insert({"name":"pqr","rollno":2})
WriteResult({ "nInserted" : 1 })
set2:PRIMARY> db.user.find()
{ "_id" : ObjectId("5f7174468ea83df5d3e8bb04"), "name" : "abc", "rollno" : 1 }
{ "_id" : ObjectId("5f7174518ea83df5d3e8bb05"), "name" : "pqr", "rollno" : 2 }
//Switch/Open 27041 shell command prompt which is now secondary
set2:SECONDARY> db
admin
//switch to database test
set2:SECONDARY> use user
switched to db user
//Execute rs.slaveOk command to sync with primary
set2:SECONDARY> rs.slaveOk()
//read documents exist in primary
set2:SECONDARY> show collections
user
set2:SECONDARY> db.user.find()
{ "_id" : ObjectId("5f7174468ea83df5d3e8bb04"), "name" : "abc", "rollno" : 1 }
{ "_id" : ObjectId("5f7174518ea83df5d3e8bb05"), "name" : "pqr", "rollno" : 2 }
set2:SECONDARY>
```


Name :
Roll No:
Aim : Backup and Restore

MongoDump

```
C:\Program Files\MongoDB\Server\3.6\bin>mongodump -d emp -c emp -o "D:\TYIT_98\backup"
2023-07-24T16:19:49.672+0530  writing emp.emp to
2023-07-24T16:19:49.677+0530  done dumping emp.emp (5 documents)
```

MongoRestore

```
C:\Program Files\MongoDB\Server\3.6\bin>mongorestore -d emp -c emp
"D:\TYIT_98\backup\emp\emp.bson"
2023-07-24T16:31:27.813+0530  checking for collection data in D:\TYIT_98\backup\emp\emp.bson
2023-07-24T16:31:27.818+0530  reading metadata for emp.emp from
D:\TYIT_98\backup\emp\emp.metadata.json
2023-07-24T16:31:27.932+0530  restoring emp.emp from D:\TYIT_98\backup\emp\emp.bson
2023-07-24T16:31:28.945+0530  no indexes to restore
2023-07-24T16:31:28.946+0530  finished restoring emp.emp (5 documents)
2023-07-24T16:31:28.947+0530  done
```

Backup Database

```
C:\Program Files\MongoDB\Server\3.6\bin>mongodump -d emp -o "D:\TYIT_98\backup"
2023-07-24T16:32:13.416+0530  writing emp.emp to
2023-07-24T16:32:13.421+0530  done dumping emp.emp (5 documents)
```

Restore Database

```
C:\Program Files\MongoDB\Server\3.6\bin>mongorestore -d emp -c emp
"D:\TYIT_98\backup\emp\emp.bson" --drop
2023-07-24T16:36:01.438+0530  checking for collection data in D:\TYIT_98\backup\emp\emp.bson
2023-07-24T16:36:01.531+0530  reading metadata for emp.emp from
D:\TYIT_98\backup\emp\emp.metadata.json
2023-07-24T16:36:01.711+0530  restoring emp.emp from D:\TYIT_98\backup\emp\emp.bson
2023-07-24T16:36:01.772+0530  no indexes to restore
2023-07-24T16:36:01.772+0530  finished restoring emp.emp (5 documents)
2023-07-24T16:36:01.772+0530  done
```

EXPERIMENT : No. 5 JQuery Basics

Page No.
Date 7 8 23

JQuery

JQuery is a fast, small and feature-rich Javascript library. It makes things like HTML document transveral and manipulation event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browser. With a combination of verasatility and extensibility. JQuery has changed the way that millions of people write Javascript.

What is DOM (Document Object Model)

- When you look at a website, you see a lot of elements grouped together and styled to form what's in front of you.
- To be able to access those element through code to remove, add and manipulate them, you need some form of interface- a representation of the elements on a page that is structured and follows a set of rules on how to model them.
- This is what the DOM is. The DOM also lets you capture browser events - such as a user clicking a link, submitting a form, or scrolling down the page.

Teacher's Sign. :

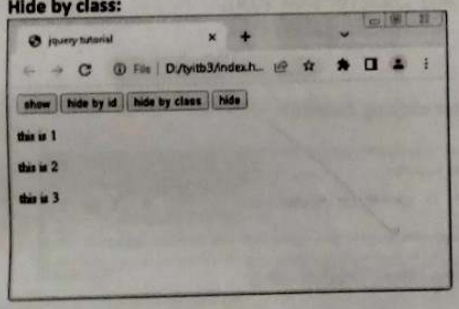
Html file: index.html

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.js"></script>
<script src="app.js"></script>
<title> jquery tutorial
</title>
</head>
<body>
<button>show</button>
<button id="b1">hide by id</button>
<button class ="bclass">hide by class</button>
<button id="b2">hide</button>
<p id="p1">this is 1</p>
<p id="p2">this is 2</p>
<p id="p3">this is 3</p>
<p class ="pclass">this is 4</p>
</body>
</html>
```

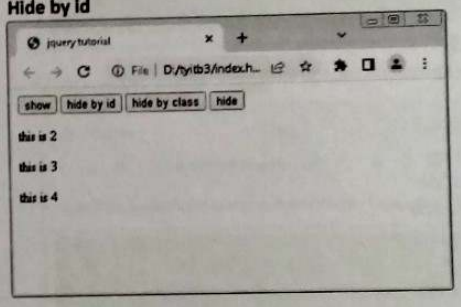
App.js file:

```
$(document).ready(function(){
$("button").click(function(){
$("p").show();
});
$("#b1").click(function(){
$("#p1").hide();
});
$(".bclass").click(function(){
$(".pclass").hide();});
$("#b2").click(function(){
$("p").hide();
});
});
```

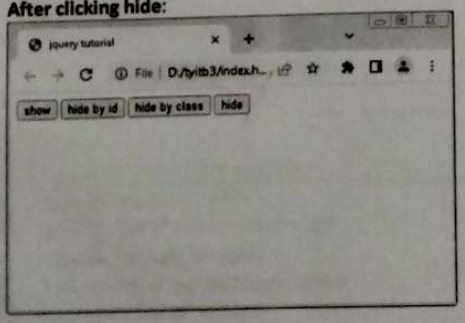
Hide by class:



Hide by id



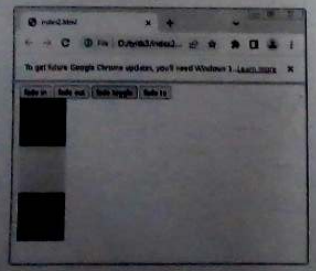
After clicking hide:



Html file:index2.html:

```
<html>
<head>
<script src="jquery.js"></script>
<script src="app2.js"></script>
<style>
div{
width:80px;
height:80px;
}
#d1{
background-color:red;
}
#d2{
background-color:yellow;
}
#d3{
background-color:green;
}
</style>
<title>
</title>
</head>
<body>
<button id="b1">fade in</button>
<button id="b2">fade out</button>
<button id="b3">fade toggle</button>
<button id="b4">fade to</button>
<div id="d1"></div>
<dlv id="d2"></div>
<dlv id="d3"></div>
</body></html>
```

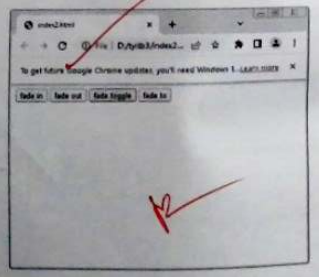
After clicking toggle:



App2.js file

```
$(document).ready(function(){
$("#b1").click(function(){
$("#d1").fadeIn("slow");
$("#d2").fadeIn("fast");
$("#d3").fadeIn("fast");
});
$("#b2").click(function(){
$("#d1").fadeOut();
$("#d2").fadeOut("slow");
$("#d3").fadeOut("fast");
});
$("#b3").click(function(){
$("#d1").fadeToggle();
$("#d2").fadeToggle("slow");
$("#d3").fadeToggle("fast");
});
$("#b4").click(function(){
$("#d1").fadeTo("slow",0.2);
$("#d2").fadeTo("slow",0.6);
$("#d3").fadeTo("slow",0.8);
});
});
```

After clicking fadeout:

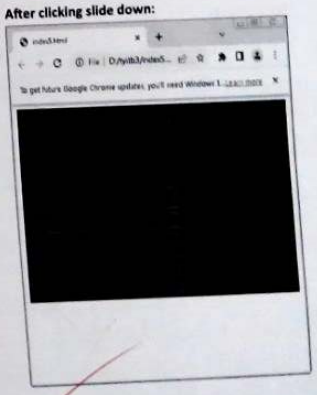


```
<!DOCTYPE html>
<html>
<head>
<title></title>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("#flip1").click(function(){
$("#panel1").slideDown();
});
$("#flip2").click(function(){
$("#panel2").slideUp();
});
$("#flip3").click(function(){
$("#panel3").slideToggle();
});
});
</script>

<style>
div{
        padding: 5px;
        background-color:Red;
        border: 1px solid black;
}
#panel1{
        padding:50px;
        display:none;
}
#panel2{
        padding:50px;
}
#panel3{
        padding:50px;
}
</style>
</head>
<body>
<div id="flip1">slide up</div>
<div id="panel1"></div>
<div id="flip2">slide down</div>
<div id="panel2"></div>
<div id="flip3">slide toggle</div>
<div id="panel3"></div>
</body>
</html>
```

After clicking slide up:

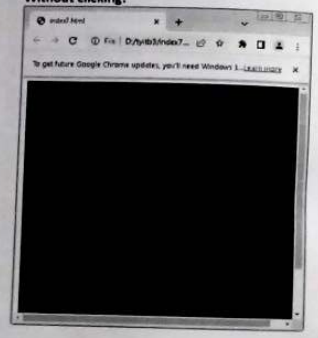
After clicking slide down:



Colour changing with jquery

```
<!DOCTYPE html>
<html>
<head>
<title></title>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("h5").click(function(){
$("div").trigger("bgchange");
});
$("div").on("bgchange",function(){
$(this).css("background-color","red");
});
});
</script>
<style>
div{
height:500px;
width:500px;
background-color:green;
position : absolute;
}
</style>
</head>
<body>
<div>
<h5>click me</h5>
<h5>or me</h5>
</div>
</body>
<html>
```

Without clicking:



Ater clicking:

EXPERIMENT : No. 6 JQuery Advance

Page No.
Date 19 | 8 | 23

JQuery Events

JQuery events are the actions that can be detected by your web application. They are used to create dynamic web pages. An event shows the exact moment when something happens.

These are some examples of events:-

1) click - Clicking an elements such as a button
2) hover - Interacting with an element via the mouse, in pure javascript, known as mouse enter and mouseleave.
3) submit - Submitting the form.
4) trigger - Making an event happen
5) off - Removing an event

Binding Event

To binding code to an event, use event methods such as click() and pass the function into it.

Ex :-
```
$("div").click(function(){
    alert("clicked");
});
```

Unbinding Event

To unbinding event, use off() method

Ex : `$("div").off();`

Sundaram

Teacher's Sign

The event object

Whenever you bind an event to a function and that functio
is then triggered. JQuery passes what's known as the 'event'
object. This object contains a lots of information about the
event. To get access to this, just make your event handle
take one parameter as an argument,

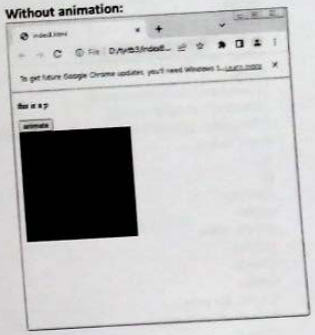
eg:
```
$(function(){
    $("p").on("click", function(event){
        console.log(event);
    });
});
```

Animation program:

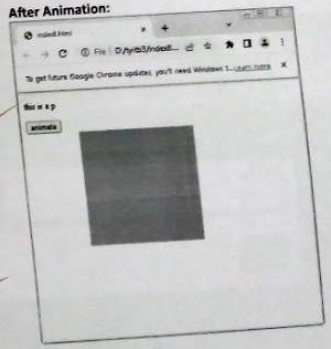
```
<!DOCTYPE html>
<html>
<head>
<title></title>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("div").animate({"height":"300px"
,"opacity":"0.5"}, "slow");

$("div").animate({"height":"200px",
"left":"100px"}, "slow" );
});
});
</script>
<style>
div
{
height:200px;
width:200px;
background-color:red;
position : absolute;
}
</style>
</head>
<body>
<p>this is a p</p>
<button>animate</button>
<div></div>
</body>
</html>
```

Without animation:

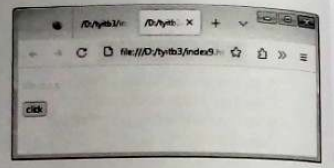


After Animation:

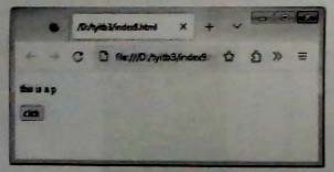


```
<!DOCTYPE html>
<html>
<head>
<title></title>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("p").slideUp("slow").slideDown("slow").css("c
olor","pink");
});
});
</script>
<style>
margin :10px;
</style>
</head>
<body>
<p> this is a p</p>
<button>click</button>
</body>
</html>
```

Without Animation:



After Animation:



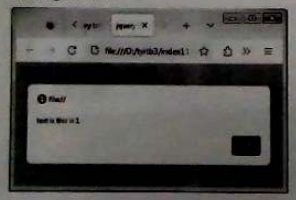
Practical6
Name:
Q1.write a jquery effect with callback function
Call back function:

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("p").slideUp("slow",function(){
alert("your para is slide up");
});
});
});
</script>
<title> jquery tutorial
</title>
</head>
<body>
<p> this is effect</p>
<button> call back
</button>
</body>
</html>
```

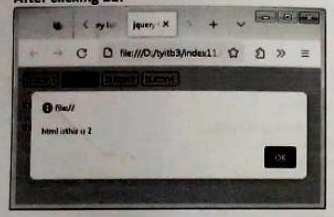
Q2.write a jquery to get and set text content

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("button").click(function(){
$("p").slideUp("slow",function(){
alert("your para is slide up");
});
});
});
</script>
<title> jquery tutorial
</title>
</head>
<body>
<p> this is effect</p>
<button> call back </button>
</body>
</html>
```

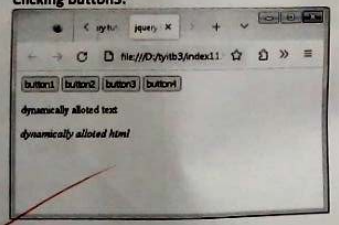
Clicking button1:



After clicking b2:



Clicking button3:

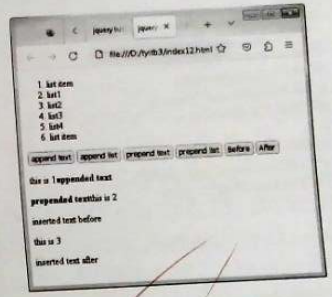


Name:

Q3.write a jquery to insert html element at the beginning and end of html.

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("#b1").click(function(){
$("#p1").append("appended text");
});
$("#b2").click(function(){
$("ol").append("<li>list item</li>");
});
$("#b3").click(function(){
$("#p2").prepend("prepended text");
});
$("#b4").click(function(){
$("ol").prepend("<li>list item</li>");
});
$("#b5").click(function(){
$("#p3").before("inserted text before");
});
$("#b6").click(function(){
$("#p3").after("inserted text after");
});
});
</script>
<title> jquery tutorial
</title>
</head>
<body>
<ol>
<li>list1</li>
<li>list2</li>
<li>list3</li>
<li>list4</li>
</ol>
<button id="b1">append text</button>
<button id="b2">append list</button>
<button id="b3">prepend text</button>
<button id="b4">prepend list</button>
<button id="b5">Before</button>
<button id="b6">After</button>
<p id="p1">this is 1</p>
<p id="p2">this is 2</p>
<p id="p3">this is 3</p>
</body>
</html>
```

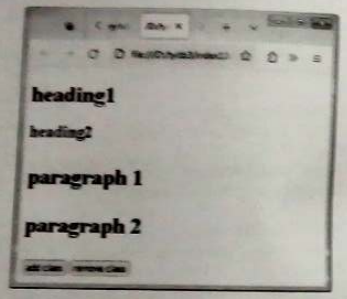
Output:



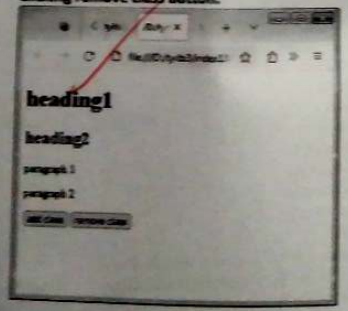
Q4.write a jquery to add and remove css classes from html

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("#b1").click(function(){
$("h1,h2").addClass("class2");
$("p").addClass("class1");
});
$("#b2").click(function(){
$("h1,h2").removeClass("class2");
$("p").removeClass("class1");
});
});
</script>
<style>
.class1{
font-weight:bold;
font-size:xx-large;
}
.class2{
color:blue;
}
</style>
</head>
<body>
<h1> heading1</h1>
<h2> heading2</h2>
<p id=p1> paragraph 1</p>
<p id=p2> paragraph 2</p>
<button id=b1>add class</button>
<button id=b2>remove class</button>
</body>
</html>
```

Clicking add class button:



Clicking remove class button:



EXPERIMENT: No. JSON

Page No:
Date 4 9 23

JSON

The Javascript Object Notation data format or JSON for short, is derived from the literals of the Javascript programming language. This makes a JSON a subset of the javascript language. As a subset, JSON does not possesses any additional features that the javascript language itself does not already posses. Although JSON is a subset of a programming language, it itself is not a programming language but, in fact, a data interchange format.

JSON is known as the data interchange standard which Subtextually implies that it can be used as the data format whenever the exchange of data occurs. A data exchange can occur between both browser and server and even server to server, for that matter, of course these are not the only possible means to exchange JSON and to leave it at those two would be rather limiting.

JSON is a nutshell, is a textual representation defines by a small set of governing is a textual in which data is structured. The JSON specification state that data can be structured in either of the two following compositions:-

1. A collection of name/value pairs
2. An ordered list of values

Eg:
```
{
  "name": "xyz",
  "cities": ["mumbai"]
}
```

Teacher's Sign:

Web Storage Interface

Web storage allows for the storing of data, the retrival of data and the removal of data. The means by which we will be working with data and the storage object is via the web storage API.

There are six members that make up the web storage API and each provides a specific need for working with data persistence.

1) setItem - The Storage object method setItem possesses the signature as given below and is the method that we will use to persist data

eg:- setItem (key, value);

2) getItem - The method allows us to retrive possisted Item

eg:- getItem (key);

3) removeItem - The Storage object method removeItem is the sole means of expiring the persistence of an Individual key/value pair

eg:- removeItem(key);

4) clear - delete all persisted items

eg:- clear()

5) key - The Storage object method key is used to obtained the identies of all stored keys that posses accompanying data retained by the given Storage object

eg:- key(index)

6) length - The Storage object provides us with access to the length of all values stored by the local Storage Object

eg:- var maxIndex = localStorage.length.

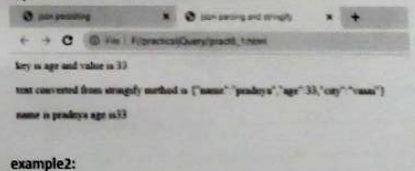
pract7_1.html

```
<html>
<head>
<title>json parsing and stringify</title>
<script src="jquery.js"> </script>
<script>
$(document).ready(function()
{
var text='{"name":"pradnya","age":33,"city":"vasai"}'
var obj=JSON.parse(text);
$("#p1").text("key is age and value is "+obj.age);

var textfromstrigify=JSON.stringify(obj);
$("#p2").text("text converted from stringify method is "+textfromstrigify);

$("#p3").text("name is "+obj.name+"\nage is"+obj.age);
});
</script>
</head>
<body>
<p id="p1"></p>
<p id="p2"></p>
<p id="p3"></p>
</body>
</html>
```

o/p:



example2:

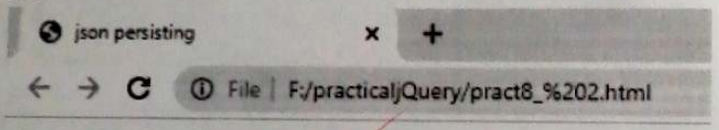
pract7_2.html:

```
<html>
<head>
<title>json persisting</title>
<script src="jquery.js"> </script>
<script>
$(document).ready(function()
{
text1 = '{name: "pradnya", age: 33, city: "vasai"}';
```

```
//storing data
localStorage.setItem("testJSON", text1);

// Retrieving data
textfrompersistingjson = localStorage.getItem("testJSON");
$("#p2").text(textfrompersistingjson);
});
</script>
</head>
<body>
<p id="p1"></p>
<p id="p2"></p>
</body>
</html>
```

o/p:



{name: "pradnya", age: 33, city: "vasai"}

EXPERIMEN

Sundaram

1. Start mongodb server
2. Install python mongodb driver

Execute following command in cmd:

C:\Users\students>pip install pymongo

o/p:

Collecting pymongo

Downloading pymongo-3.11.1-cp39-cp39-win_amd64.whl (383 kB)

[redacted] 383 kB 656 kB/s

Installing collected packages: pymongo

Successfully installed pymongo-3.11.1

Note:

If you got an error: pip is not recognised as internal or external command

Solution:

Paste path of scripts directory from python39 directory like this:

C:\Users\students\AppData\Local\Programs\Python\Python39\Scripts

In environmental variables user variables path edit new paste

ok ok Then open new cmd and execute pip install pymongo command.

3. Execute python codes in IDLE
a) Insert
b) Retrieve
c) Update
d) Delete

Insert.py:

```
from pymongo import MongoClient
client = MongoClient('localhost:27017')
db = client.employeedata
def insert():
  try:
    eid = input('Enter Employee id :')
    ename = input('Enter Name :')
    eage = input('Enter age :')
    ecountry =input('Enter Country :')

    db.employees.insert_one(
    {
```

```
            "eid": eid,
             "ename":ename,
            "eage":eage,
            "ecountry":ecountry
            ))
            print("\nInserted data successfully\n")

          except Exception:
             print(str(e))
        insert()
```

Retrieve.py:

```
from pymongo import MongoClient
client = MongoClient('localhost:27017')
db = client.employeedata
def read():
  try:
     empCol = db.employees.find()
     print("\n All data from EmployeeData Database \n")
     for emp in empCol:
       print(emp)
  except Exception:
     print(str(e))
read()
```

update.py:

```
from pymongo import MongoClient
client = MongoClient('localhost:27017')
db = client.employeedata
def update():
  try:
     ename = input("\nEnter name to update\n")
     eage = input("\nEnter age to update\n")
     ecountry = input("\nEnter country to update\n")
     db.employees.update_one(
       {"ename": ename},
       {
       "$set": {
       "eage":eage,
       "ecountry":ecountry
     }
     }
     )
     print( "\nRecords updated successfully\n")
  except Exception:
     print(str(e))
update()
```

delete.py

```
from pymongo import MongoClient
client = MongoClient('localhost:27017')
db = client.employeedata
def delete():
  try:
     criteria = input("\nEnter employee name to delete\n")
     db.employees.delete_many({"ename":criteria})
     print("\nDeletion successful\n")
  except (Exception):
     print(str(e))
delete()
```