Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

# FACULTY OF ENGINEERING AND TECHNOLOGY

# BACHELOR OF TECHNOLOGY

## BIG DATA ANALYTICS LABORATORY

## (303105362)

## 6TH SEMESTER

## COMPUTER SCIENCE & ENGINEERING DEPARTMENT

# LABORATORY MANUAL

Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

# CERTIFICATE

This is to certify that _____with enrolment no. _____has successfully completed his/her laboratory experiments in the **BIG DATA ANALYTICS LABORATORY (303105362)** from the Department of COMPUTER SCIENCE AND ENGINEERING during the academic year 2025-26.

# INDEX

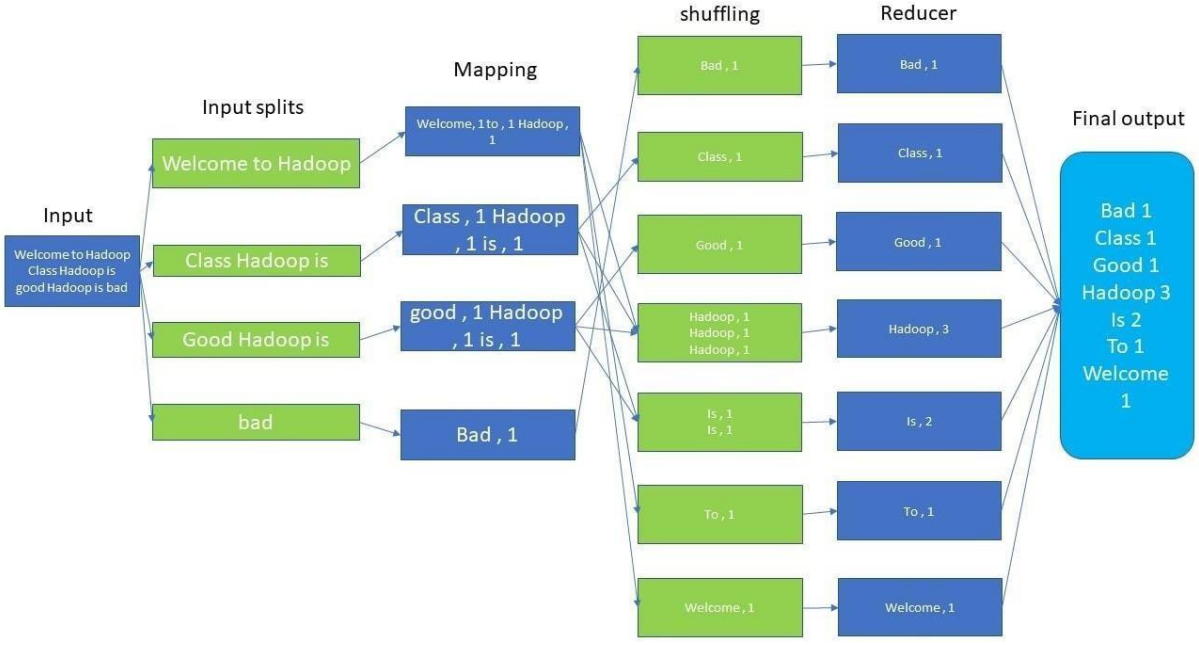
| SR NO | NAME | START DATE | END DATE | MARKS | SIGN |
|---|---|---|---|---|---|
| 1 | To understand the overall programming architecture using Map Reduce API. | | | | |
| 2 | Write a program of Word Count in Map Reduce over HDFS. | | | | |
| 3 | Basic CRUD operations in MongoDB. | | | | |
| 4 | Store the basic information about students such as roll no, name, date of birth, and address of student using various collection types such as List, Set and Map. | | | | |
| 5 | Basic commands available for the Hadoop Distributed File System. | | | | |
| 6 | Basic commands available for HIVE Query Language. | | | | |
| 7 | Basic commands of HBASE Shell. | | | | |
| 8 | Creating the HDFS tables and loading them in Hive and learn joining of tables in Hive. | | | | |

# Practical -1

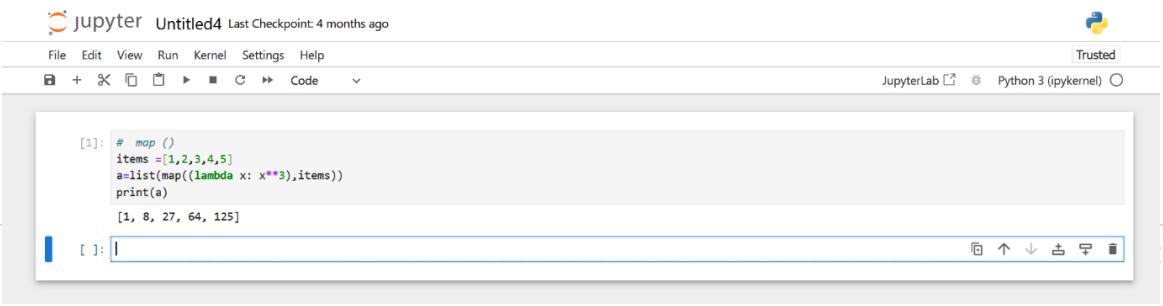
**Aim:** To Understand the overall programming architecture using MapReduce.

- MapReduce and HDFS are the two major components of Hadoop which makes it so powerful and efficient to use.
- MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner.
- The data is first split and then combined to produce the final result.
- The libraries for MapReduce is written in so many programming languages with various different-different optimizations.
- The purpose of MapReduce in Hadoop is to Map each of the jobs and then it will reduce it to equivalent tasks for providing less overhead over the cluster networ and to reduce the processing power.
- The MapReduce task is mainly divided into two phases Map Phase and Reduce Phase.

1. map(), filter(), and reduce() in Python.
2. These functions are most commonly used with Lambda function.




- map() : "A map function executes certain instructions or functionality provided to it on every item of an iterable. "The iterable could be a list, tuple, set, etc.



- reduce() : “Reduce functions apply a function to every item of an iterable and gives back a single value as a resultant."

[2]:
```
a = [1,2,3,4,5,6]
b = [2,5,0,7,3]

c = list(filter(lambda x: x in  a,b))
print(c)
```

```
[2, 5, 3]
```

- filter() : "A filter function in Python tests a specific user-defined condition for a functi and returns an iterable for the elementsand values that satisfy the condition or in other words, return true."

[3]:
```
from functools import reduce
a =reduce ( (lambda x, y: x *y ),[1,2,3,4])
print(a)
```

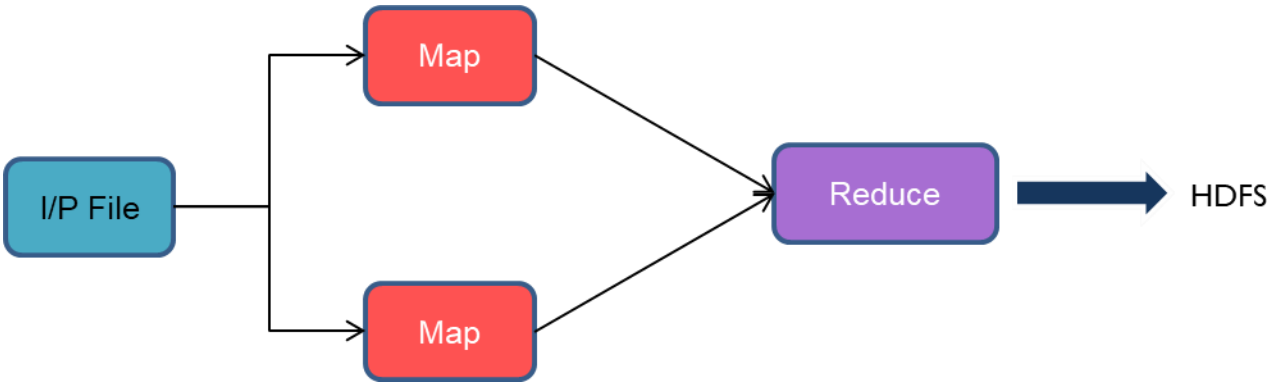
```
24
```

# Practical: 2

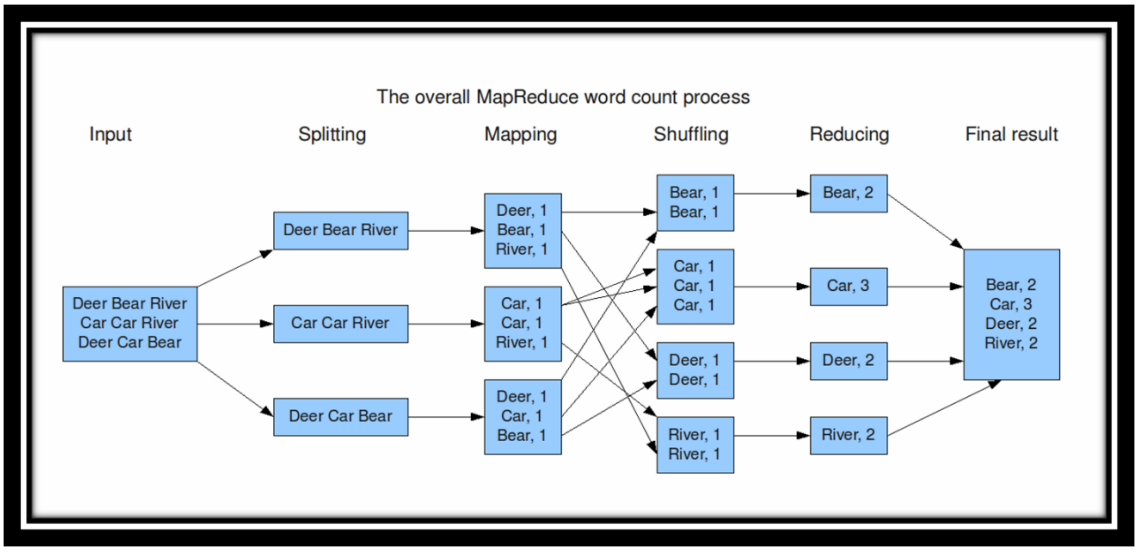
**Aim: Write a program of Word Count in Map Reduce over HDFS.**

**Description:**

MapReduce is a framework for processing large datasets using a large number of computers (nodes), collectively referred to as a cluster. Processing can occur on data stored in a file system (HDFS).A method for distributing computation across multiple nodes.Each node processes the data that is stored at that node.




Input data set is split into independent blocks – processed in parallel. Each input split is converted in Key Value pairs. Mapper logic processes each key value pair and produces and intermediate key value pairs based on the implementation logic. Resultant key value pairs can be of different type from that of input key value pairs. The output of Mapper is passed to the reducer. Output of Mapper function is the input for Reducer. Reducer sorts the intermediate key value pairs. Applies reducer logic upon the key value pairs and produces the output in desired format.Output is stored in HDFS.




MapReduce is a programming model designed for processing large amounts of data in parallel across a distributed cluster of computers. Here, I'll provide a basic example of a word count program using MapReduce in Hadoop.

## Setting Up Your Environment

1. **Install Hadoop**: Ensure you have Hadoop installed and configured on your system.
2. **Set Up HDFS**: Ensure HDFS is running and you have some text files to process.

### Word Count Program

#### 1. Mapper Class

The mapper class will read the input text line by line, split it into words, and emit each word with a count of 1.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

```
public class WordCountMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {


   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   @Override


   protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
      String line = value.toString();
      String[] words = line.split("\\s+");
      for (String str : words) {
         word.set(str);
         context.write(word, one);
      }
   }
}
```

**2. Reducer Class**

The reducer class will receive all word-count pairs with the same word and sum them up to get the final count.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

   @Override
```

```
    protected void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable value : values) {
            sum += value.get();
        }
```

```
        context.write(key, new IntWritable(sum));
    }}
```

**3. Driver Class**

The driver class is responsible for setting up the job configuration and running the job.

```
import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(WordCountMapper.class);
        job.setCombinerClass(WordCountReducer.class);
        job.setReducerClass(WordCountReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

**Running the Program**

1. **Package the Code**: Compile the code and create a JAR file.

   javac -classpath `hadoop classpath` -d wordcount_classes WordCountMapper.java WordCountReducer.java WordCount.java

jar -cvf wordcount.jar -C wordcount_classes/ .

2. **Upload Input Files to HDFS**: Place the input text files in HDFS.

   hdfs dfs -mkdir /input
   hdfs dfs -put local_input_file.txt /input

3. **Run the MapReduce Job**:

   hadoop jar wordcount.jar WordCount /input /output

4. **Check the Output**: Retrieve the output from HDFS.

   hdfs dfs -cat /output/part-r-00000

**Input:**

```
Hello world
Hello Hadoop
Hadoop MapReduce
```

**Output:**

```
Hadoop      2
Hello       2
MapReduce   1
world       1
```

**CODE for program without Mapreduce:**

```
import urllib.request
import random
from operator import itemgetter
current_word = {}
current_count = 0
story = 'http://sixty-north.com/c/t.txt'
request = urllib.request.Request(story)
response = urllib.request.urlopen(request)
each_word = []
words = None
count = 1
same_words ={}
word = []
""" looping the entire file """
#Collect All the words into a list
for line in response:
   #print "Line = " , line
   line_words = line.split()
   for word in line_words: # looping each line and extracting words
      each_word.append(word)
#for every word collected, in dict same_words
#if a key exists, such that key == word then increment Mapping Value by 1
# Else add word as new key with mapped value as 1
for words in each_word:
   if words.lower() not in same_words.keys() :

      same_words[words.lower()]=1
   else:
```

```
    same_words[words.lower()]=same_words[words.lower()]+1
for each in same_words.keys(): print ("word = ",each, ", count =
",same_words[each])
```

```
word =  b'it' , count =  10
word =  b'was' , count =  11
word =  b'the' , count =  14
word =  b'best' , count =  1
word =  b'of' , count =  12
word =  b'times' , count =  2
word =  b'worst' , count =  1
word =  b'age' , count =  2
word =  b'wisdom' , count =  1
word =  b'foolishness' , count =  1
word =  b'epoch' , count =  2
word =  b'belief' , count =  1
word =  b'incredulity' , count =  1
word =  b'season' , count =  2
word =  b'light' , count =  1
word =  b'darkness' , count =  1
word =  b'spring' , count =  1
word =  b'hope' , count =  1
word =  b'winter' , count =  1
word =  b'despair' , count =  1
word =  b'we' , count =  4
word =  b'had' , count =  2
word =  b'everything' , count =  1
word =  b'before' , count =  2
word =  b'us' , count =  2
word =  b'nothing' , count =  1
word =  b'were' , count =  2
word =  b'all' , count =  2
word =  b'going' , count =  2
word =  b'direct' , count =  2
word =  b'to' , count =  1
```

```
word =  b'heaven' , count =  1
word =  b'other' , count =  1
word =  b'way' , count =  1
word =  b'in' , count =  2
word =  b'short' , count =  1
word =  b'period' , count =  2
word =  b'so' , count =  1
word =  b'far' , count =  1
word =  b'like' , count =  1
word =  b'present' , count =  1
word =  b'that' , count =  1
word =  b'some' , count =  1
word =  b'its' , count =  2
word =  b'noisiest' , count =  1
word =  b'authorities' , count =  1
word =  b'insisted' , count =  1
word =  b'on' , count =  1
word =  b'being' , count =  1
word =  b'received' , count =  1
word =  b'for' , count =  2
word =  b'good' , count =  1
word =  b'or' , count =  1
word =  b'evil' , count =  1
word =  b'superlative' , count =  1
word =  b'degree' , count =  1
word =  b'comparison' , count =  1
word =  b'only' , count =  1
```

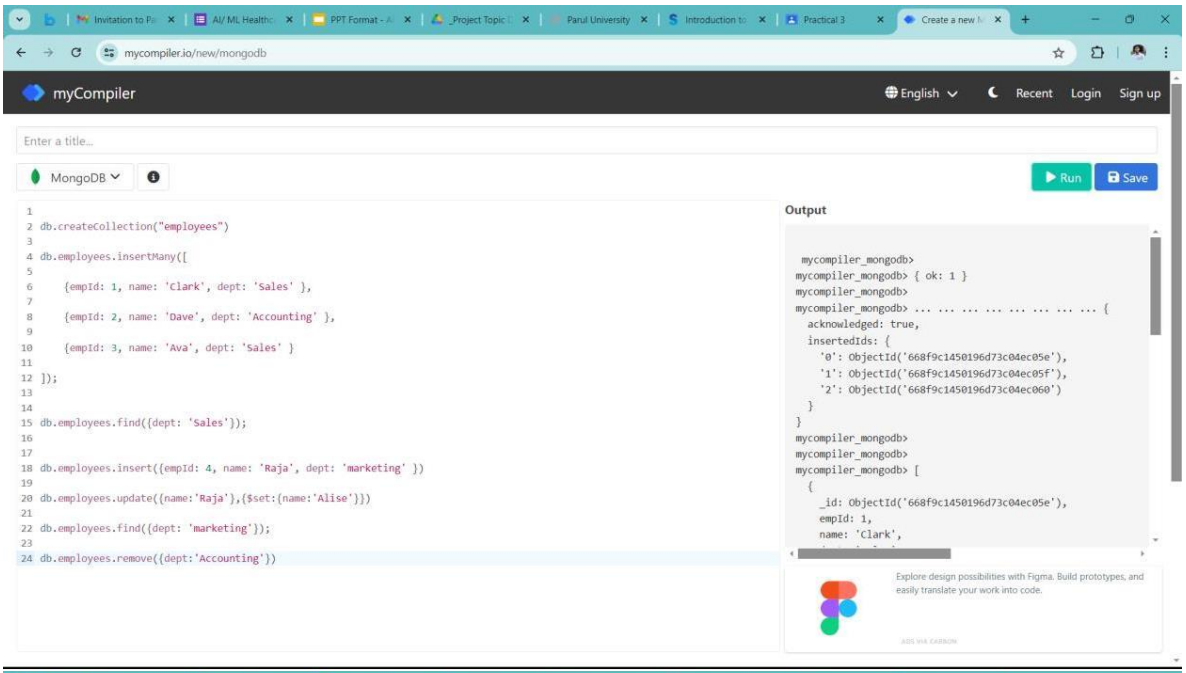
## Practical -3

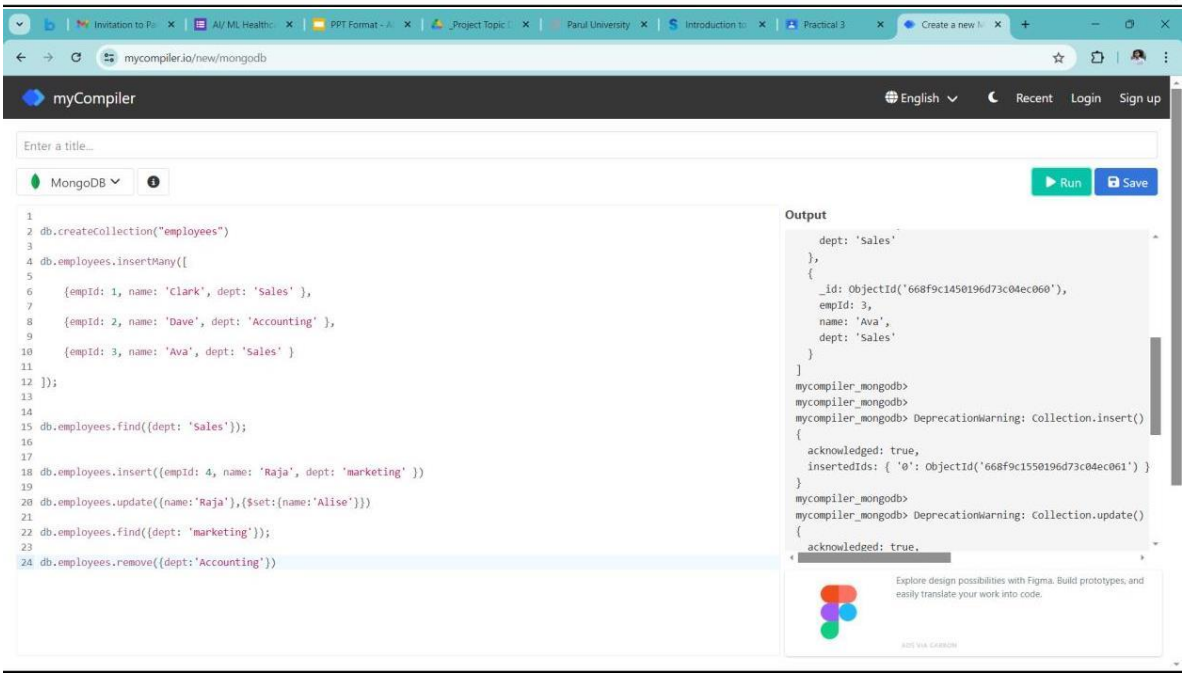
**Aim: Basic CRUD Operations in MongoDB.**

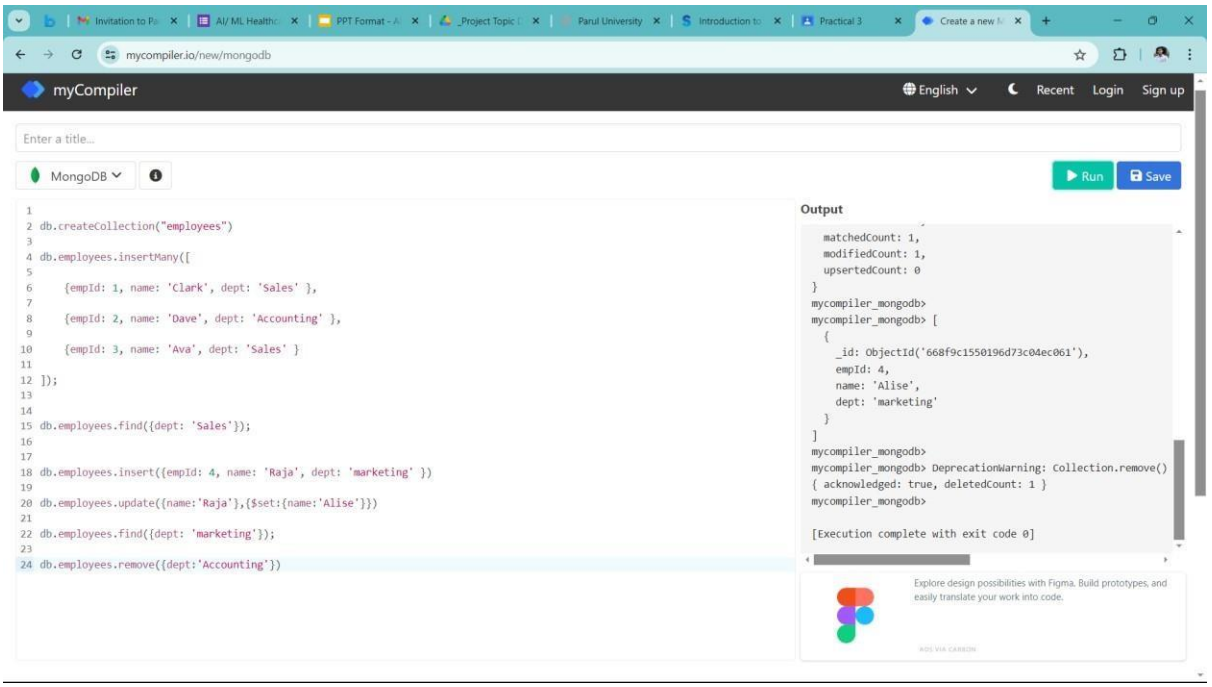
**Program -1:**

```
db.createCollection("employees")
db.employees.insertMany([
        {empId: 1, name: 'Clark', dept: 'Sales' }
        {empId: 2, name: 'Dave', dept: 'Accounting' },
        {empId: 3, name: 'Ava', dept: 'Sales' }
]);
db.employees.find({dept: 'Sales'});
db.employees.insert({empId: 4, name: 'Raja', dept: 'marketing' })
db.employees.update({name:'Raja'},{$set:{name:'Alise'}})
db.employees.find({dept: 'marketing'});

db.employees.remove({dept:'Accounting'})
```

## Information and Communication Technology



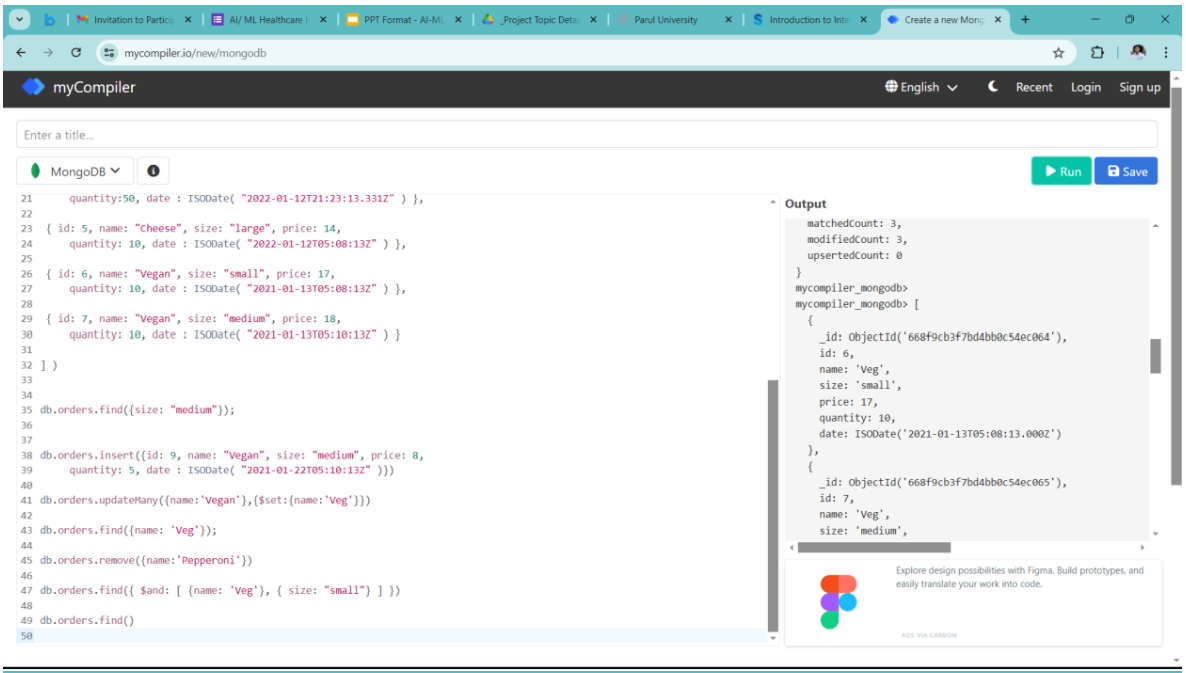


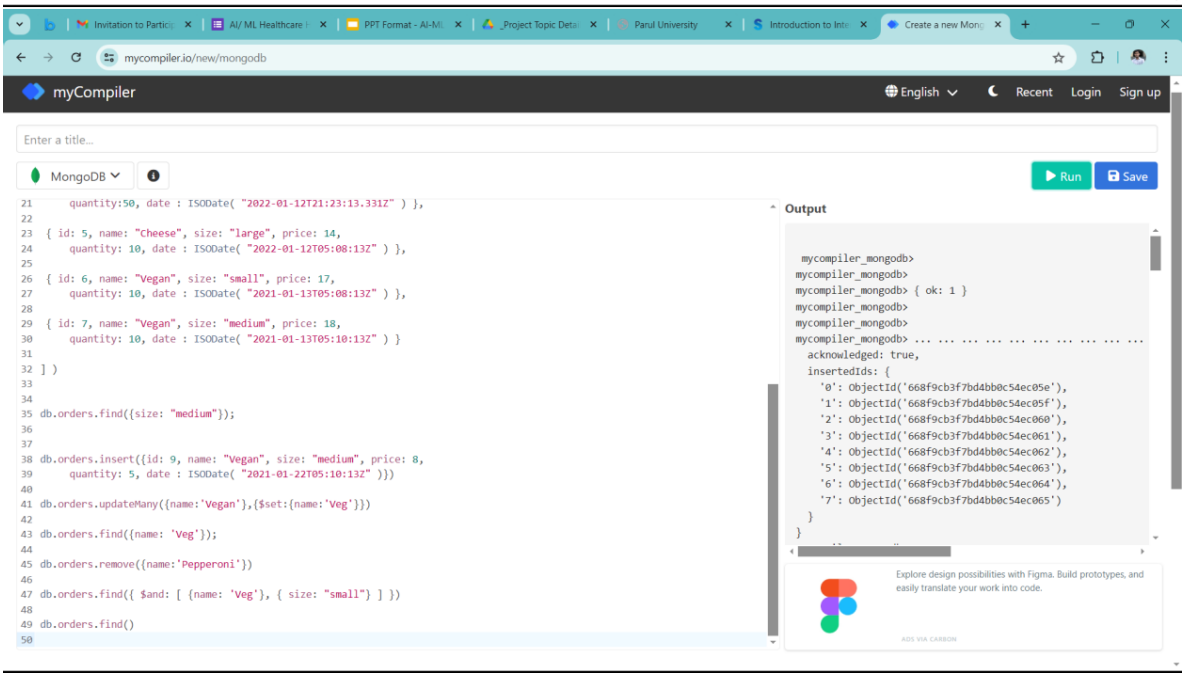

## Program -2 Create orders collection and items to the database.

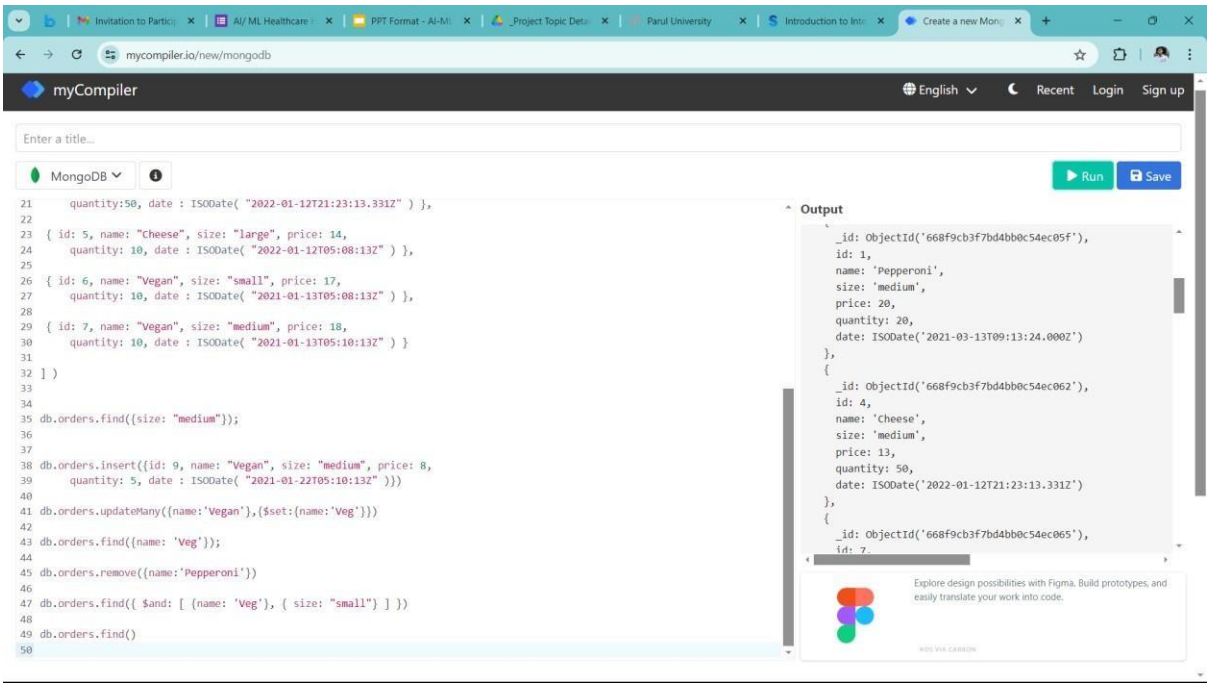
```
db.createCollection("orders")

db.orders.insertMany( [

  { id: 0, name: "Pepperoni", size: "small", price: 19,
    quantity: 10, date: ISODate( "2021-03-13T08:14:30Z" ) },
```

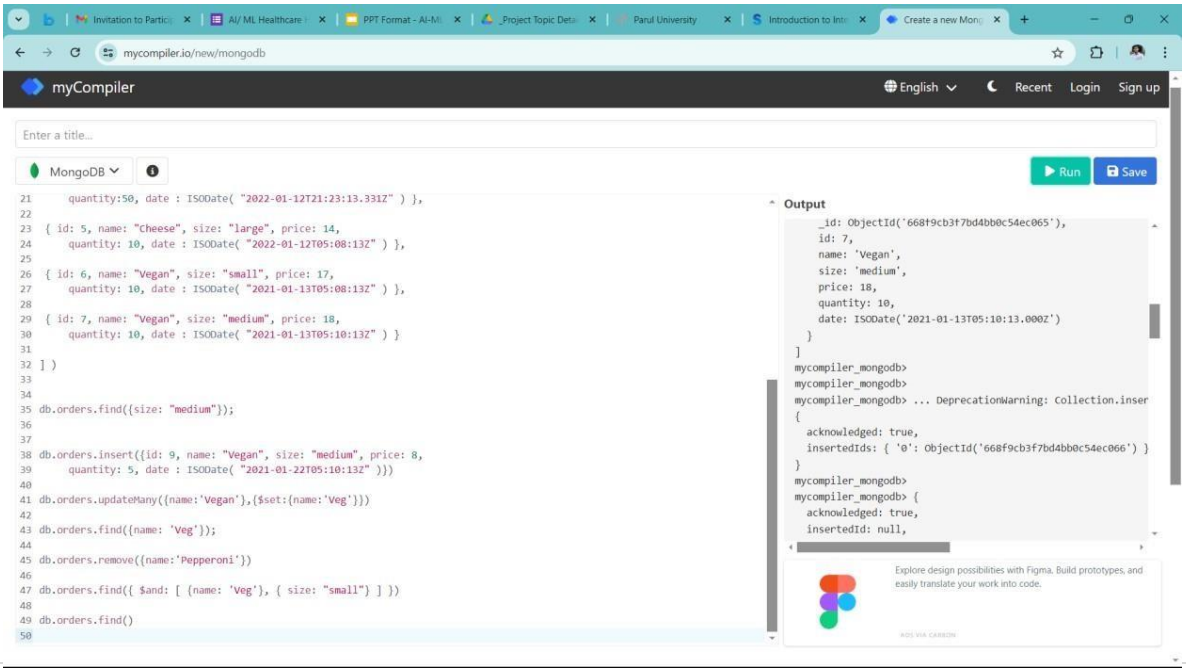
```
{ id: 1, name: "Pepperoni", size: "medium", price: 20,
   quantity: 20, date : ISODate( "2021-03-13T09:13:24Z" ) },

{ id: 2, name: "Pepperoni", size: "large", price: 21,
   quantity: 30, date : ISODate( "2021-03-17T09:22:12Z" ) },
```

```
 { id: 3, name: "Cheese", size: "small", price: 12,
   quantity: 15, date : ISODate( "2021-03-13T11:21:39.736Z" ) },

{ id: 4, name: "Cheese", size: "medium", price: 13,
   quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },

{ id: 5, name: "Cheese", size: "large", price: 14,
   quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },

{ id: 6, name: "Vegan", size: "small", price: 17,
   quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },

{ id: 7, name: "Vegan", size: "medium", price: 18,
   quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }

] )


db.orders.find({size: "medium"});


db.orders.insert({id: 9, name: "Vegan", size: "medium", price: 8,
   quantity: 5, date : ISODate( "2021-01-22T05:10:13Z" )})

db.orders.updateMany({name:'Vegan'},{$set:{name:'Veg'}})

db.orders.find({name: 'Veg'});

db.orders.remove({name:'Pepperoni'})

db.orders.find({ $and: [ {name: 'Veg'}, { size: "small"} ] })

db.orders.find()
```


mycompiler.io/new/mongodb
myCompiler
English
Recent
Login
Sign up
Enter a title...
MongoDB
Run
Save
quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },
{ id: 5, name: "cheese", size: "large", price: 14,
quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },
{ id: 6, name: "Vegan", size: "small", price: 17,
quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },
{ id: 7, name: "Vegan", size: "medium", price: 18,
quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }
] )
db.orders.find({size: "medium"});
db.orders.insert({id: 9, name: "Vegan", size: "medium", price: 8,
quantity: 5, date : ISODate( "2021-01-22T05:10:13Z" )})
db.orders.updateMany({name:'Vegan'},{$set:{name:'Veg'}})
db.orders.find({name: 'Veg'});
db.orders.remove({name:'Pepperoni'})
db.orders.find({ $and: [ {name: 'Veg'}, { size: "small"} ] })
db.orders.find()
Output
matchedCount: 3,
modifiedCount: 3,
upsertedCount: 0
}
mycompiler_mongodb>
mycompiler_mongodb> [
{
_id: ObjectId('668f9cb3f7bd4bb0c54ec064'),
id: 6,
name: 'Veg',
size: 'small',
price: 17,
quantity: 10,
date: ISODate('2021-01-13T05:08:13.000Z')
},
{
_id: ObjectId('668f9cb3f7bd4bb0c54ec065'),
id: 7,
name: 'Veg',
size: 'medium',
Explore design possibilities with Figma. Build prototypes, and easily translate your work into code.


mycompiler.io/new/mongodb
myCompiler
English
Recent
Login
Sign up
Enter a title...
MongoDB
Run
Save
quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },
{ id: 5, name: "cheese", size: "large", price: 14,
quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },
{ id: 6, name: "Vegan", size: "small", price: 17,
quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },
{ id: 7, name: "Vegan", size: "medium", price: 18,
quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }
] )
db.orders.find({size: "medium"});
db.orders.insert({id: 9, name: "Vegan", size: "medium", price: 8,
quantity: 5, date : ISODate( "2021-01-22T05:10:13Z" )})
db.orders.updateMany({name:'Vegan'},{$set:{name:'Veg'}})
db.orders.find({name: 'Veg'});
db.orders.remove({name:'Pepperoni'})
db.orders.find({ $and: [ {name: 'Veg'}, { size: "small"} ] })
db.orders.find()
Output
mycompiler_mongodb>
mycompiler_mongodb>
mycompiler_mongodb> { ok: 1 }
mycompiler_mongodb>
mycompiler_mongodb>
mycompiler_mongodb> ... ... ... ... ... ... ... ... ... ...
acknowledged: true,
insertedIds: {
'0': ObjectId('668f9cb3f7bd4bb0c54ec05e'),
'1': ObjectId('668f9cb3f7bd4bb0c54ec05f'),
'2': ObjectId('668f9cb3f7bd4bb0c54ec060'),
'3': ObjectId('668f9cb3f7bd4bb0c54ec061'),
'4': ObjectId('668f9cb3f7bd4bb0c54ec062'),
'5': ObjectId('668f9cb3f7bd4bb0c54ec063'),
'6': ObjectId('668f9cb3f7bd4bb0c54ec064'),
'7': ObjectId('668f9cb3f7bd4bb0c54ec065')
}
}
Explore design possibilities with Figma. Build prototypes, and easily translate your work into code.


mycompiler.io/new/mongodb
myCompiler
English
Recent
Login
Sign up
Enter a title...
MongoDB
Run
Save
quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },
{ id: 5, name: "Cheese", size: "large", price: 14,
quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },
{ id: 6, name: "Vegan", size: "small", price: 17,
quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },
{ id: 7, name: "Vegan", size: "medium", price: 18,
quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }
] )
db.orders.find({size: "medium"});
db.orders.insert({id: 9, name: "Vegan", size: "medium", price: 8,
quantity: 5, date : ISODate( "2021-01-22T05:10:13Z" )})
db.orders.updateMany({name:'Vegan'},{$set:{name:'Veg'}})
db.orders.find({name: 'Veg'});
db.orders.remove({name:'Pepperoni'})
db.orders.find({ $and: [ {name: 'Veg'}, { size: "small"} ] })
db.orders.find()
Output
_id: ObjectId('668f9cb3f7bd4bb0c54ec05f'),
id: 1,
name: 'Pepperoni',
size: 'medium',
price: 20,
quantity: 20,
date: ISODate('2021-03-13T09:13:24.000Z')
_id: ObjectId('668f9cb3f7bd4bb0c54ec062'),
id: 4,
name: 'Cheese',
size: 'medium',
price: 13,
quantity: 50,
date: ISODate('2022-01-12T21:23:13.331Z')
_id: ObjectId('668f9cb3f7bd4bb0c54ec065'),
Explore design possibilities with Figma. Build prototypes, and easily translate your work into code.


mycompiler.io/new/mongodb
myCompiler
English
Recent
Login
Sign up
Enter a title...
MongoDB
Run
Save
quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },
{ id: 5, name: "Cheese", size: "large", price: 14,
quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },
{ id: 6, name: "Vegan", size: "small", price: 17,
quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },
{ id: 7, name: "Vegan", size: "medium", price: 18,
quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }
] )
db.orders.find({size: "medium"});
db.orders.insert({id: 9, name: "Vegan", size: "medium", price: 8,
quantity: 5, date : ISODate( "2021-01-22T05:10:13Z" )})
db.orders.updateMany({name:'Vegan'},{$set:{name:'Veg'}})
db.orders.find({name: 'Veg'});
db.orders.remove({name:'Pepperoni'})
db.orders.find({ $and: [ {name: 'Veg'}, { size: "small"} ] })
db.orders.find()
Output
_id: ObjectId('668f9cb3f7bd4bb0c54ec065'),
id: 7,
name: 'Vegan',
size: 'medium',
price: 18,
quantity: 10,
date: ISODate('2021-01-13T05:10:13.000Z')
]
mycompiler_mongodb>
mycompiler_mongodb>
mycompiler_mongodb> ... DeprecationWarning: Collection.inser
{
acknowledged: true,
insertedIds: { '0': ObjectId('668f9cb3f7bd4bb0c54ec066') }
}
mycompiler_mongodb>
mycompiler_mongodb> {
acknowledged: true,
insertedId: null,
Explore design possibilities with Figma. Build prototypes, and easily translate your work into code.

## PRACTICAL – 4

**Aim: Store the basic information about students such as roll no, name, date of birth, and address of student using various collection types such as List, Set and Hadoop.**

**Code:**

- Create database studentdb and Make Collection With name "student" and Follow Queries

  ✓ Created Database

  > use studentdb

  switched to DB studentdb

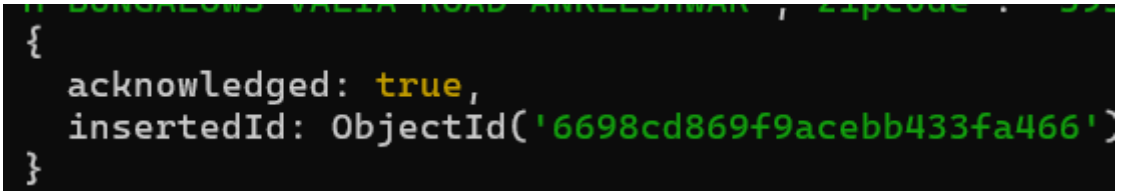
  ✓ Create Collection

  > db.createCollection("student")

✓ Insert Records Into student Collection using **insertOne()** method

> db.student.insertOne({no:1, name:"ANUJ MULHAR",dob: "2004-01-30",e_mail: "thefunvlogs0@gmail.com", phone:"9904857092",address: {"B": "118","Nr.Kosamdi": "KUMKUM BUNGALOWS VALIA ROAD ANKLESHWAR","zipcode": "393001"},Branch:"CSE", marks:[58,96,55]})

Output:




- ✓ Insert Records Into student Collection using **insertMany()** method.

db.student.insertMany([{no:2, name:"KUNAL NAKUM",dob: "2005-09-26",e_mail: "kunalnakum.123@gmail.com", phone:"9313564747",address: {"Kalam Bhawan-c ": "401","street": "WAGHODIA","zipcode": "391760"},Branch:"CSE", marks:[30,55,100]},{no:3, name:"Aditya patel",dob: "2002-10-21",e_mail: "adityapatel123@gmail.com", phone:"9313564747",address: {"Kalam Bhawan -C": "101","street": "Limda","zipcode": "391760"},Branch:"CSE", marks:[60,50,42]}])

Output:

- ✓ List out all Document

>db.student.find()

```
studentdb> db.student.find()
[
  {
    _id: ObjectId('6698cd869f9acebb433fa466'),
    no: 1,
    name: 'ANUJ MULHAR',
    dob: '2004-01-30',
    e_mail: 'thefunvlogs0@gmail.com',
    phone: '9904857092',
    address: {
      B: '118',
      'Nr.Kosamdi': 'KUMKUM BUNGALOWS VALIA ROAD ANKLESHWAR',
      zipcode: '393001'
    },
    Branch: 'CSE',
    marks: [ 58, 96, 55 ]
  },
  {
    _id: ObjectId('6698cf0d9f9acebb433fa467'),
    no: 2,
    name: 'KUNAL NAKUM',
    dob: '2005-09-26',
    e_mail: 'kunalnakum.123@gmail.com',
    phone: '9313564747',
    address: { 'Kalam Bhawan-c ': '401', street: 'WAGHODIA', zipcode: '391760' },
    Branch: 'CSE',
    marks: [ 30, 55, 100 ]
  },
  {
    _id: ObjectId('6698cf0d9f9acebb433fa468'),
    no: 3,
    name: 'Aditya patel',
    dob: '2002-10-21',
    e_mail: 'adityapatel123@gmail.com',
    phone: '9313564747',
    address: { 'Kalam Bhawan -C': '101', street: 'Limda', zipcode: '391760' },
    Branch: 'CSE',
    marks: [ 60, 50, 42 ]
  }
]
```

update name of no 1 as "kunal sinh"

studentdb> db.student.update({no:1},{$set:{name:"kunal sinh"}})

```
studentdb> db.student.update({no:1},{$set:{name:"kunal sinh"}})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

- ✓ To Find Document From the student collection where name begins with A

studentdb> db.student.find({name:/^A/})

```
studentdb> db.student.find({name:/^A/})
[
  {
    _id: ObjectId('6698cf0d9f9acebb433fa468'),
    no: 3,
    name: 'Aditya patel',
    dob: '2002-10-21',
    e_mail: 'adityapatel123@gmail.com',
    phone: '9313564747',
    address: { 'Kalam Bhawan -C': '101', street: 'Limda', zipcode: '391760' },
    Branch: 'CSE',
    marks: [ 60, 50, 42 ]
  }
]
```

- ✓ To Find Document From the student collection where name ends

with
studentdb> db.student.find({name:/l$/})

```
studentdb> db.student.find({name:/l$/})
[
  {
    _id: ObjectId('6698cf0d9f9acebb433fa468'),
    no: 3,
    name: 'Aditya patel',
    dob: '2002-10-21',
    e_mail: 'adityapatel123@gmail.com',
    phone: '9313564747',
    address: { 'Kalam Bhawan -C': '101', street: 'Limda', zipcode: '391760' },
    Branch: 'CSE',
    marks: [ 60, 50, 42 ]
  }
]
```

To Find Document From the student collection where name has S in any position

>db.student.find({name:/k/})

```
studentdb> db.student.find({name:/k/})
[
  {
    _id: ObjectId('6698cd869f9acebb433fa466'),
    no: 1,
    name: 'kunal sinh',
    dob: '2004-01-30',
    e_mail: 'thefunvlogs0@gmail.com',
    phone: '9904857092',
    address: {
      B: '118',
      'Nr.Kosamdi': 'KUMKUM BUNGALOWS VALIA ROAD ANKLESHWAR',
      zipcode: '393001'
    },
    Branch: 'CSE',
    marks: [ 58, 96, 55 ]
  }
]
```

Count:

It is used to count the number of document in the collections

db.student.count()

```
studentdb> db.student.count()
DeprecationWarning: Collection
3
```

forEach() method is used to apply a JavaScript function for each document in a cursor.

db.student.find().forEach( function(myDoc) { print( "name: " + myDoc.name ); } );

```
studentdb> db.student.find().forEach( function(myDoc) { print( "name: " + myDoc.name ); } );
name: kunal sinh
name: KUNAL NAKUM
name: Aditya patel
```

---

limit() method is used to specify the maximum number of documents the cursor will return.

db.student.find().limit(2);

output

```
studentdb> db.student.find().limit(2);
[
  {
    _id: ObjectId('6698cd869f9acebb433fa466'),
    no: 1,
    name: 'kunal sinh',
    dob: '2004-01-30',
    e_mail: 'thefunvlogs0@gmail.com',
    phone: '9904857092',
    address: {
      B: '118',
      'Nr.Kosamdi': 'KUMKUM BUNGALOWS VALIA ROAD ANKLESHWAR',
      zipcode: '393001'
    },
    Branch: 'CSE',
    marks: [ 58, 96, 55 ]
  },
  {
    _id: ObjectId('6698cf0d9f9acebb433fa467'),
    no: 2,
    name: 'KUNAL NAKUM',
    dob: '2005-09-26',
    e_mail: 'kunalnakum.123@gmail.com',
    phone: '9313564747',
    address: { 'Kalam Bhawan-c ': '401', street: 'WAGHODIA', zipcode: '391760' },
    Branch: 'CSE',
    marks: [ 30, 55, 100 ]
  }
]
```

db.student.find().sort( { "name": -1 } )

```
studentdb> db.student.find().sort( { "name": -1 } )
[
  {
    _id: ObjectId('6698cd869f9acebb433fa466'),
    no: 1,
    name: 'kunal sinh',
    dob: '2004-01-30',
    e_mail: 'thefunvlogs0@gmail.com',
    phone: '9904857092',
    address: {
      B: '118',
      'Nr.Kosamdi': 'KUMKUM BUNGALOWS VALIA ROAD ANKLESHWAR',
      zipcode: '393001'
    },
    Branch: 'CSE',
    marks: [ 58, 96, 55 ]
  },
  {
    _id: ObjectId('6698cf0d9f9acebb433fa467'),
    no: 2,
    name: 'KUNAL NAKUM',
    dob: '2005-09-26',
    e_mail: 'kunalnakum.123@gmail.com',
    phone: '9313564747',
    address: { 'Kalam Bhawan-c ': '401', street: 'WAGHODIA', zipcode: '391760' },
    Branch: 'CSE',
    marks: [ 30, 55, 100 ]
  },
  {
    _id: ObjectId('6698cf0d9f9acebb433fa468'),
    no: 3,
    name: 'Aditya patel',
    dob: '2002-10-21',
    e_mail: 'adityapatel123@gmail.com',
    phone: '9313564747',
    address: { 'Kalam Bhawan -C': '101', street: 'Limda', zipcode: '391760' },
    Branch: 'CSE',
    marks: [ 60, 50, 42 ]
  }
]
```

$map

Applies an expression to each item in an array and returns an array with the applied results.

{ $map: { input: <expression>, as: <string>, in: <expression> } }

```
db.grades.insertMany([
 { _id: 1, quizzes: [ 5, 6, 7 ] },
 { _id: 2, quizzes: [ ] },
 { _id: 3, quizzes: [ 3, 8, 9 ] }
])
```

```
db.grades.aggregate(
  [
    { $project:
      { adjustedGrades:
        {
          $map:
            {
              input: "$quizzes",
              as: "grade",
              in: { $add: [ "$$grade", 2 ] }
            }
        }
      }
    }
  ]
```

)

output

{ "_id" : 1, "adjustedGrades" : [ 7, 8, 9 ] }
{ "_id" : 2, "adjustedGrades" : [ ] }
{ "_id" : 3, "adjustedGrades" : [ 5, 10, 11 ] }

```
studentdb> use grade
switched to db grade
grade> db.grades.insertMany([
...    { _id: 1, quizzes: [ 5, 6, 7 ] },
...    { _id: 2, quizzes: [ ] },
...    { _id: 3, quizzes: [ 3, 8, 9 ] }
... ])
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3 } }
grade> db.grades.aggregate(
...     [
...        { $project:
...           { adjustedGrades:
...              {
...                $map:
...                   {
...                     input: "$quizzes",
...                     as: "grade",
...                     in: { $add: [ "$$grade", 2 ] }
...                   }
...              }
...           }
...        }
...     ]
... )
[
  { _id: 1, adjustedGrades: [ 7, 8, 9 ] },
  { _id: 2, adjustedGrades: [] },
  { _id: 3, adjustedGrades: [ 5, 10, 11 ] }
]
```

**set**

Adds new fields to documents. $set outputs documents that contain all existing fields from the input documents and newly added fields.

db.scores.insertMany([

{ _id: 1, student: "Mulhar", homework: [ 10, 5, 10 ], quiz: [ 10, 8 ], extraCredit: 0 },

{ _id: 2, student: "ANUJ", homework: [ 5, 6, 5 ], quiz: [ 8, 8 ], extraCredit: 8 }])

```
parul> use scores
switched to db scores
scores> db.scores.insertMany([
...     { _id: 1, student: "Mulhar", homework: [ 10, 5, 10 ], quiz: [ 10, 8 ], extraCredit: 0 },
...     { _id: 2, student: "ANUJ", homework: [ 5, 6, 5 ], quiz: [ 8, 8 ], extraCredit: 8 }
... ])
{ acknowledged: true, insertedIds: { '0': 1, '1': 2 } }
scores>
```

Aggregate the homework and quiz to the collection scores using the set metho

```
db.scores.aggregate( [
  {
    $set: {
      totalHomework: { $sum: "$homework" },
```

```
      totalQuiz: { $sum: "$quiz" }
    }
  },
  {
    $set: {

  totalScore: { $add: [ "$totalHomework", "$totalQuiz", "$extraCredit" ] } }
  }
] )
```

```
scores> db.scores.aggregate( [
...     {
...       $set: {
...          totalHomework: { $sum: "$homework" },
...          totalQuiz: { $sum: "$quiz" }
...       }
...     },
...     {
...       $set: {
...          totalScore: { $add: [ "$totalHomework", "$totalQuiz", "$extraCredit" ] } }
...     }
... ] )
[
  {
    _id: 1,
    student: 'Mulhar',
    homework: [ 10, 5, 10 ],
    quiz: [ 10, 8 ],
    extraCredit: 0,
    totalHomework: 25,
    totalQuiz: 18,
    totalScore: 43
  },
  {
    _id: 2,
    student: 'ANUJ',
    homework: [ 5, 6, 5 ],
    quiz: [ 8, 8 ],
    extraCredit: 8,
    totalHomework: 16,
    totalQuiz: 16,
    totalScore: 40
  }
]
scores>
```

**Lab Experiment**

Designing an employee management system using MongoDB as a storage engine. employee management system must have the capacity to store many differed types of objects with different sets of attributes. These kinds of data collections are quite compatible with MongoDB's data model: Use a single MongoDB collection to store all the employees data of the parul university like the employee name, ID and salary. MongoDB's dynamic schema means that each document need not conform to the same schema.

1. Create parul Database using MongoDB and create a employee collection in the parul Database.

2. Write the basic salary ,HRA, DA, Income Tax , Professional Tax of an employee and calculate its Gross salary of employee using aggregate functions (Gross salary=Basic Salary + HRA + DA - Income Tax –Professional Tax).

Code :

```
parul> db.employee.insertMany([
... {
```

```
...     name: "MULHAR ANUJ SURESH",
...     employeeID: "E001",
...     basicSalary: 40000,
...     HRA: 40000,
...     DA: 2500,
...     incomeTax: 5500,
...     professionalTax: 3000
...  },
...  {
...     name: "ANUJ SURESH MULHAR",
...     employeeID: "E002",
...     basicSalary: 70000,
...     HRA: 15000,
...     DA: 7000,
...     incomeTax: 6000,
...     professionalTax: 4500
...  }
... ])
```

```
parul> db.employee.insertMany([
...   {
...     name: "MULHAR ANUJ SURESH",
...     employeeID: "E001",
...     basicSalary: 40000,
...     HRA: 40000,
...     DA: 2500,
...     incomeTax: 5500,
...     professionalTax: 3000
...   },
...   {
...     name: "ANUJ SURESH MULHAR",
...     employeeID: "E002",
...     basicSalary: 70000,
...     HRA: 15000,
...     DA: 7000,
...     incomeTax: 6000,
...     professionalTax: 4500
...   }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6698d22c9f9acebb433fa469'),
    '1': ObjectId('6698d22c9f9acebb433fa46a')
  }
}
```

aggregate

Use MongoDB's aggregation framework to calculate the gross salary for each employee:

```
parul> db.employee.aggregate([
...  {
...    $addFields: {
...      grossSalary: {
...        $subtract: [
...          { $add: ["$basicSalary", "$HRA", "$DA"] },
...          { $add: ["$incomeTax", "$professionalTax"] }
...        ]
...      }
...    }
...  }
... ])
```

Information and Communication Technology

```
parul> db.employee.aggregate([
...   {
...     $addFields: {
...       grossSalary: {
...         $subtract: [
...           { $add: ["$basicSalary", "$HRA", "$DA"] },
...           { $add: ["$incomeTax", "$professionalTax"] }
...         ]
...       }
...     }
...   }
... ])
[
  {
    _id: ObjectId('6698cb1e9f9acebb433fa464'),
    name: 'John Doe',
    employeeID: 'E001',
    basicSalary: 50000,
    HRA: 10000,
    DA: 5000,
    incomeTax: 5000,
    professionalTax: 2000,
    grossSalary: 58000
  },
  {
    _id: ObjectId('6698cb1e9f9acebb433fa465'),
    name: 'Jane Smith',
    employeeID: 'E002',
    basicSalary: 60000,
    HRA: 12000,
    DA: 6000,
    incomeTax: 6000,
    professionalTax: 2500,
    grossSalary: 69500
  },
  {
    _id: ObjectId('6698d22c9f9acebb433fa469'),
    name: 'MULHAR ANUJ SURESH',
    employeeID: 'E001',
    basicSalary: 40000,
    HRA: 40000,
    DA: 2500,
    incomeTax: 5500,
    professionalTax: 3000,
    grossSalary: 74000
```

# PRACTICAL-5

**AIM: - Basic commands available for the Hadoop distributed file system.**

HDFS provide a set of essential commands to interact with and manage files stored in the distributed environment. Here are some of the most commonly used commands.

**File and Directory management**

- **ls:** List the contents of a directory.

hadoop fs -ls/path/to/directory

- **mkdir:** Creates a new directory.

hadoop fs -mkdir/path/to/new/directory

- **rm:** Deletes a file or directory.

hadoop fs – rm/path/to/file

- **mv:** Moves or rename a file or directory.

hadoop fs -mv/source/path/destination/path

- **cp:** Copies a file or directory.

hadoop fs -cp/source/path/destination/path

- **touchz:** Creates an empty file.

hadoop fs -touchz/path/to/file

- **cat:** display the content of a filr

hadoop fs -cat/opath/to/file

- **du:** Display disk usage statistics for a path

hadoop fs -du -h/path/to/directory

**Data transfer**

- **copyFormLocal:** Copies files from local file system to HDFS.

hadoop fs -copyFromLocal/local/file/hdfs/destination

- **copyToLocal:** Copies files from HDFS to local file system.

hadoop fs -copyToLocal/hdfs/sourcs/local/destination

- **moveFromLocal:** Moves files from local file system to HDFS.

hadoop fs -moveFromLocal/local/file/hdfs/destination

**Other Useful Commands**

- **df:** Displays HDFS filesystem status.

hadoop fs -df -h

- **chmod:** Changes file or directory permission.

hadoop fs -chmod777/path/to/file

- **chown:** Changes file or directory ownership.

hadoop fs -chown user:group/path/to/file

- rm: This command deletes a file from HDFS.

hadoop fs -rm<filename/directoryName>

- **du:** It will give the size of each file in directory.

hadoop fs -du <dirName>

- **dus:** This command will give the total size of directory/file.

hadoop fs -dus <dirName>

- **stat:** It will give the last modification time of directory or path. In short it will give stats of directory or file.

hadoop fs -stat <hdfs file>

- **setrep:** This command is used to change the replication factor of a file/directory in hdfs.by default,it is 3 for anything which stroed in HDFS(as set in hdfs coresite.xml).

- **test**

The test command is used for file test operations.

| Options | Description |
|---|---|
| **-d** | **Check whether the path given by the user is a directory or not, return 0 if it is a directory.** |
| **-e** | **Check whether the path given by the user exists or not, return if the path exists.** |
| **-f** | **Check whether the path given by the user is a file or not, return 0 if it is a file.** |
| **-s** | **Check if the path is not empty, return 0 if a path is not empty.** |
| **-r** | **Return 0 if the path exists and read permission is granted** |
| **-w** | **Return 0 if the exists and write permission is granted.** |
| **-z** | **Checks whether the file size is 0 byte or not, return 0 if the file is of 0 bytes.** |

- **getmerge:**

  getmerge command merges a list of files in a directory on the HDFS

filesystem into a single local file on the local filesystem.

- **stat** prints the statistics about the file or directory in the specification format.

**Formats:**

**%b- file size in bytes**

**%g- group name of owner**

**%n- file name**

**%o- block size**

**%r- replication**

**%u- user name of owner**

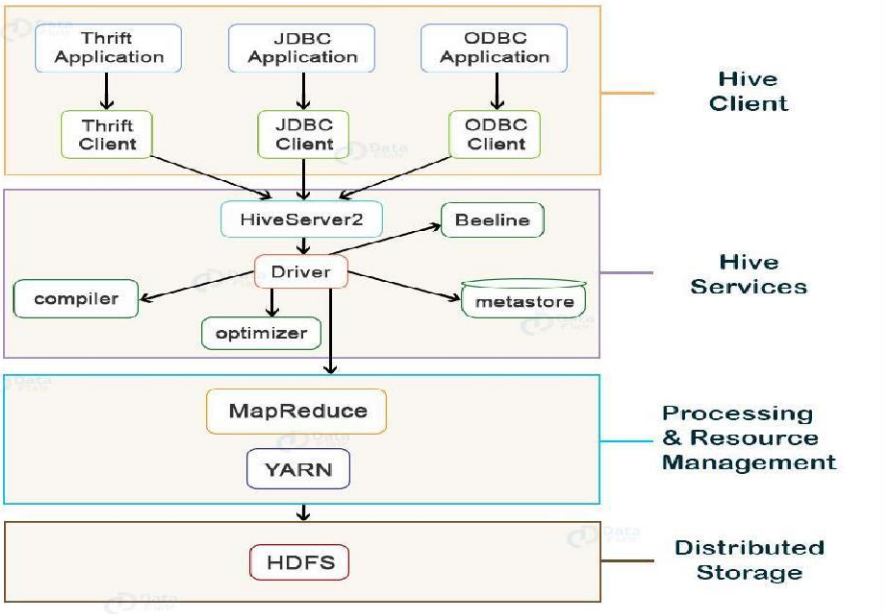
**%y- notification date**

## Practical -6

**Aim: To study basic commands available for HIVE Query Language.**

**Description:**

Apache Hive is an open-source data warehousing tool for performing distributed processing and data analysis. It was developed by Facebook to reduce the work of writing the Java MapReduce program. Apache Hive uses a Hive Query language, which is a declarative language similar to SQL. Hive translates the hive queries into MapReduce programs. It supports developers to perform processing and analyses on structured and semi-structured data by replacing complex java MapReduce programs with hive queries. One who is familiar with SQL commands can easily write the hive queries.




**Hive Architecture & Its Components**

Hive supports applications written in any language like Python, Java, C++, Ruby, etc. using JDBC, ODBC, and Thrift drivers, for performing queries on the Hive. Hence, one can easily write a hive client application in any language of its own choice.

**Hive clients are categorized into three types:**

**1. Thrift Clients**

The Hive server is based on Apache Thrift so that it can serve the request from a thrift client.

**2. JDBC client**

Hive allows for the Java applications to connect to it using the JDBC driver. JDBC driver uses Thrift to communicate with the Hive Server.

**3. ODBC client**

Hive ODBC driver allows applications based on the ODBC protocol to connect to Hive. Similar to the JDBC driver, the ODBC driver uses Thrift to communicate with the Hive Server.

- create table teacher(ID int primary key, Name text, department TEXT, hiredate TEXT, category TEXT, gender TEXT,salary int);
- insert into teacher
- values(1,'Taniya','social studies','3/17/1994','TGT','F','25000');
- insert into teacher
- values(2,'abhishek','Art','2/12/1990','PGT','M','20000');
- insert into teacher
- values(3,'sanjana',' english ','5/16/1980','PGT','F','30000');

insert into teacher

values(4,'vishwajeet','english','10/16/1989','TGT','M','25000');

**OUTPUT:**

| id | name | department | hiredate | category | gender | salary |
|---|---|---|---|---|---|---|
| 1 | Taniya | social studies | 1994-03-17 | TGT | F | 25000 |
| 2 | Abhishek | Art | 1990-02-12 | PGT | M | 20000 |
| 3 | Sanjana | English | 1980-05-16 | PGT | F | 30000 |
| 4 | Vishwajeet | English | 1989-10-16 | TGT | M | 25000 |

- select * from teacher;
- select max (hiredate), gender from teacher
- group by gender;

**OUTPUT:**

| id | name | department | hiredate | category | gender | salary |
|---|---|---|---|---|---|---|
| 1 | Taniya | social studies | 1994-03-17 | TGT | F | 25000 |
| 2 | Abhishek | Art | 1990-02-12 | PGT | M | 20000 |
| 3 | Sanjana | English | 1980-05-16 | PGT | F | 30000 |
| 4 | Vishwajeet | English | 1989-10-16 | TGT | M | 25000 |

| max_hiredate | gender |
|---|---|
| 1994-03-17 | F |
| 1990-02-12 | M |