Hindi Vidya Prachar Samiti's

# RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE (AUTONOMOUS)

GHATKOPAR (W), MUMBAI - 400 086

## DEPARTMENT OF STATISTICS

**2022 -2023**

## M. Sc. Statistics Part II Semester III

## RJSPGSTAPA301

## Big Data Technologies

# Name: Ashishkumar Rana

# Roll No.: 903

Hindi Vidya Prachar Samiti's

# RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE (AUTONOMOUS)

## GHATKOPAR (W), MUMBAI - 400 086

# *Certificate*

NAAC Accredited "A" Grade
तिद्या अमृतं अश्नुते
ESTD 1963
RJC
ISO 9001:2000
Best College
UNIVERSITY of MUMBAI

*This is to certify that Mr./Ms. Ashishkumar Rana*
*Roll No 903 of M.Sc. Statistics class has completed the required number of experiments in the subject of*
*Big Data Technologies in the Department of Statistics during the academic year 2022 - 2023.*

*Professor In-Charge*
*Prof. Bharati Bhole*

*Head of Department*
*Prof. Chaya Pinge*

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



# Index

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



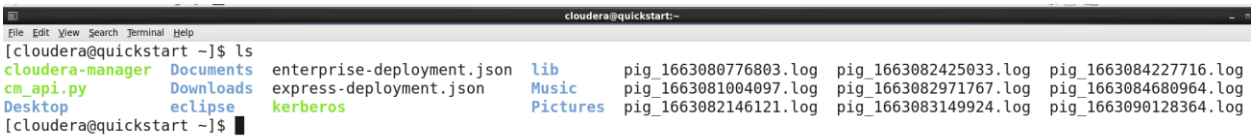
## Practical 1 - Basic Commands of Linux and Hadoop

a) Explain the syntax of following Linux and Hadoop commands and demonstrate the execution of each command.

### Linux Commands

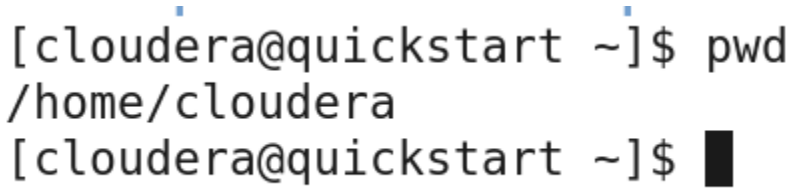
### 1. LS Command

**-> ls-** Command to display the list of Files and Directories in Cloudera. It Lists the contents of the directory specified by path, showing the names, and permissions.



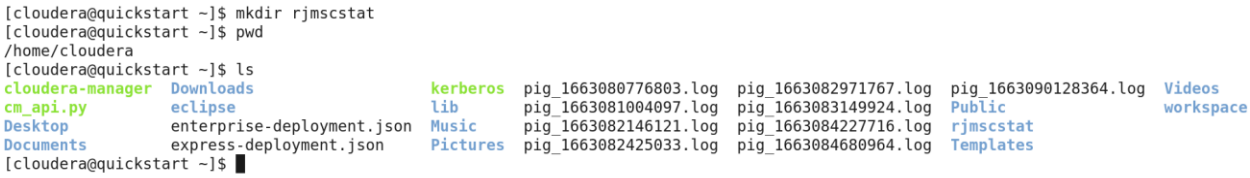

### 2. PWD Command

**-> pwd-** Command to display the current working directory.




### 3. MKDIR Command

Command to create the directory in cloudera.
Syntax:- mkdir directory_name
Here I am trying to create a directory named “rjmscstat” in cloudera.
After creating , Using the ls command, we can check for the directories in cloudera or Using the ls command we listed the directory ‘rjmscstat” created using mkdir.



4. CD Command

Command to change the directory from the current directory to another directory.

Syntax:- cd directory_name

Here I am trying to change the directory from Cloudera home to rjmscstat.

\

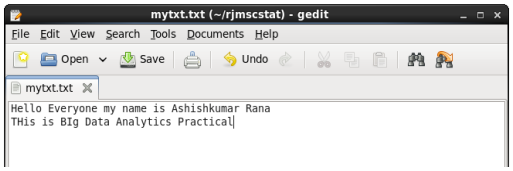
```
[cloudera@quickstart ~]$ cd rjmscstat
[cloudera@quickstart rjmscstat]$
```

5. Gedit Command

Command to create any txt file inside the directory. Here I am adding some text inside the file.

Syntax:- gedit mytxt.txt

```
[cloudera@quickstart rjmscstat]$ gedit mytxt.txt
```




6. Cat Command

Command to see the content inside the file. Here, I am trying to see the content inside the file mytxt.txt

Syntax:- cat mytxt.txt

```
[cloudera@quickstart rjmscstat]$ cat mytxt.txt
Hello Everyone my name is Ashishkumar Rana
THis is BIg Data Analytics Practical
[cloudera@quickstart rjmscstat]$
```

7. LS -L Command

Syntax:- ls -l

Using this command we come to know about the permission assigned to the respective directory

```
[cloudera@quickstart ~]$ ls -l
total 196
-rwxrwxr-x 1 cloudera cloudera  5141 Jun  9  2015 cloudera-manager
-rwxrwxr-x 1 cloudera cloudera  9922 Jun  9  2015 cm_api.py
drwxrwxr-x 2 cloudera cloudera  4096 Jun  9  2015 Desktop
drwxrwxr-x 4 cloudera cloudera  4096 Jun  9  2015 Documents
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Downloads
drwxrwsr-x 9 cloudera cloudera  4096 Feb 19  2015 eclipse
-rw-rw-r-- 1 cloudera cloudera 53819 Jun  9  2015 enterprise-deployment.json
-rw-rw-r-- 1 cloudera cloudera 50679 Jun  9  2015 express-deployment.json
-rwxrwxr-x 1 cloudera cloudera  5007 Jun  9  2015 kerberos
drwxrwxr-x 2 cloudera cloudera  4096 Jun  9  2015 lib
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Music
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Pictures
-rw-rw-r-- 1 cloudera cloudera  4135 Sep 14 01:52 pig_1663139566177.log
-rw-rw-r-- 1 cloudera cloudera  2583 Sep 14 03:00 pig_1663148736943.log
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Public
drwxrwxr-x 2 cloudera cloudera  4096 Sep 16 01:51 rjmscstat
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Templates
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Videos
drwxrwxr-x 4 cloudera cloudera  4096 Jun  9  2015 workspace
[cloudera@quickstart ~]$
```

## 8. Chmod Command

Using this command we change the mode of execution for a particular directory.
Syntax:- chmod ugo+rwx rjmscstat

```
[cloudera@quickstart ~]$ chmod ugo+rwx rjmscstat
[cloudera@quickstart ~]$ ls -l
total 196
-rwxrwxr-x 1 cloudera cloudera  5141 Jun  9  2015 cloudera-manager
-rwxrwxr-x 1 cloudera cloudera  9922 Jun  9  2015 cm_api.py
drwxrwxr-x 2 cloudera cloudera  4096 Jun  9  2015 Desktop
drwxrwxr-x 4 cloudera cloudera  4096 Jun  9  2015 Documents
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Downloads
drwxrwsr-x 9 cloudera cloudera  4096 Feb 19  2015 eclipse
-rw-rw-r-- 1 cloudera cloudera 53819 Jun  9  2015 enterprise-deployment.json
-rw-rw-r-- 1 cloudera cloudera 50679 Jun  9  2015 express-deployment.json
-rwxrwxr-x 1 cloudera cloudera  5007 Jun  9  2015 kerberos
drwxrwxr-x 2 cloudera cloudera  4096 Jun  9  2015 lib
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Music
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Pictures
-rw-rw-r-- 1 cloudera cloudera  4135 Sep 14 01:52 pig_1663139566177.log
-rw-rw-r-- 1 cloudera cloudera  2583 Sep 14 03:00 pig_1663148736943.log
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Public
drwxrwxrwx 2 cloudera cloudera  4096 Sep 16 01:51 rjmscstat
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Templates
drwxr-xr-x 2 cloudera cloudera  4096 Aug  5 00:31 Videos
drwxrwxr-x 4 cloudera cloudera  4096 Jun  9  2015 workspace
```

## 9. touch Command

Another Command to create any txt file inside the directory.
Syntax:- touch mytxt.txt

```
[cloudera@quickstart ~]$ touch one.txt
[cloudera@quickstart ~]$ ls
cloudera-manager  Desktop    Downloads  enterprise-deployment.json  kerberos  Music    Pictures              pig_1663148736943.log  rjmscstat  Videos
cm_api.py         Documents  eclipse    express-deployment.json     lib       one.txt  pig_1663139566177.log  Public                 Templates  workspace
[cloudera@quickstart ~]$
```

10. MV Command

Command to move any file from a directory to Another Directory. Here I am trying to move one.txt from cloudera home to inside the directory rjmscstat.

Syntax:- mv one.txt rjmscstat

```
[cloudera@quickstart ~]$ mv one.txt rjmscstat
[cloudera@quickstart ~]$ cd rjmscstat
[cloudera@quickstart rjmscstat]$ ls
mytxt.txt  one.txt
```

11. Echo Command

Another Command to add any content inside the txt file inside the directory. Here I am adding some text inside the file.

Syntax:- echo “text”

```
[cloudera@quickstart rjmscstat]$ echo "Hi Everyone. How are you?">> one.txt
[cloudera@quickstart rjmscstat]$ cat one.txt
Hi Everyone. How are you?
```

12. Remove Command

Command to delete txt file inside the directory. Or we can remove the entire directory.

Here I am removing the directory rjmscstat.

Syntax:- rm -r rjmscstat

```
[cloudera@quickstart ~]$ rm -r rjmscstat
rm: descend into directory `rjmscstat'? y
rm: remove regular file `rjmscstat/mytxt.txt'? y
rm: remove regular file `rjmscstat/one.txt'? y
rm: remove directory `rjmscstat'? y
[cloudera@quickstart ~]$
```

# Hadoop HDFS Command

## 1) Hadoop Version->hadoop version

The Hadoop fs shell command version prints the Hadoop version.

```
[cloudera@quickstart ~]$ hadoop version
Hadoop 2.6.0-cdh5.4.2
Subversion http://github.com/cloudera/hadoop -r 15b703c8725733b7b2813d2325659
d57e7a3f
Compiled by jenkins on 2015-05-20T00:03Z
Compiled with protoc 2.5.0
From source with checksum de74f1adb3744f8ee85d9a5b98f90d
This command was run using /usr/jars/hadoop-common-2.6.0-cdh5.4.2.jar
```

## 2) LS Command

### ->hdfs dfs -ls /

HDFS Command to display the list of Files and Directories in HDFS. It Lists the contents of the directory specified by path,showing the names,permissions, owner, size and modification date for each entry.

**hdfs dfs is the command that is specific to HDFS.**

```
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 5 items
drwxr-xr-x   - hbase supergroup          0 2022-02-10 20:00 /hbase
drwxr-xr-x   - solr  solr                0 2015-06-09 03:38 /solr
drwxrwxrwx   - hdfs  supergroup          0 2022-02-07 21:01 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2015-06-09 03:38 /user
drwxr-xr-x   - hdfs  supergroup          0 2015-06-09 03:36 /var
```

### ->hadoop fs -ls /

hadoop fs is more “generic” command that allows you to interact with multiple file systems
including Hadoop. we are using the ls command to enlist the files and directories present in
HDFS. The Hadoop fs shell command ls displays a list of the contents of a directory specified in
the path provided by the user. It shows the name, permissions, owner, size, and modification date
for each file or directories in the specified directory.

Using the ls command, we can check for the directories in HDFS.

**3) MKDIR Command**

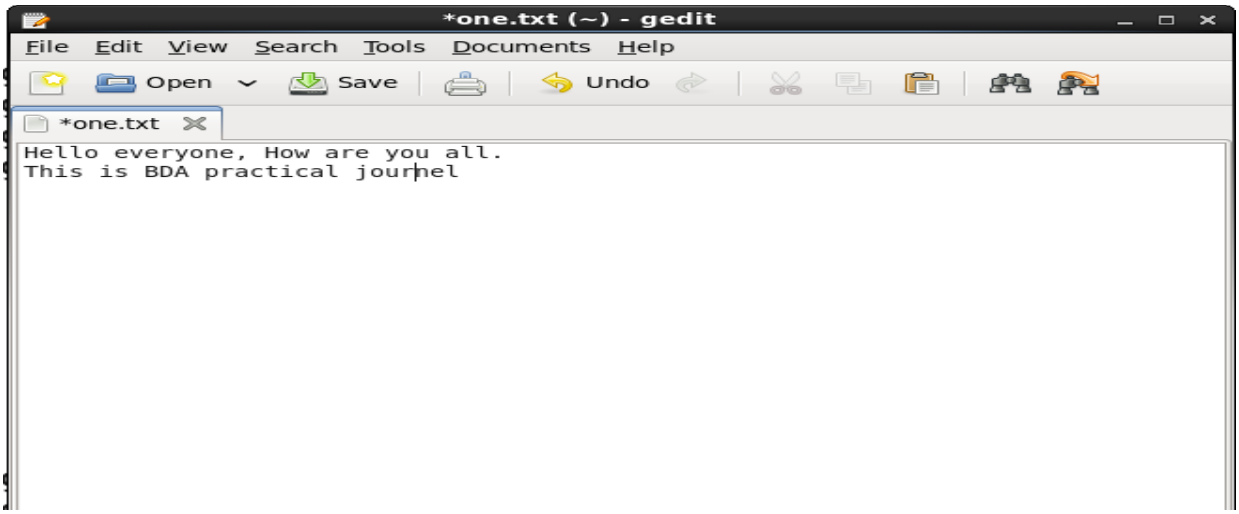
HDFS Command to create the directory in HDFS.
Usage: hdfs dfs –mkdir /directory_name
Here I am trying to create a directory named “rjc” in HDFS.
After creating ,Using the ls command, we can check for the directories in HDFS or Using ls command we listed the directory ‘rjc” created using mkdir.

```
[cloudera@quickstart ~]$ hdfs dfs -mkdir /rjc
[cloudera@quickstart ~]$ hadoop fs -ls /
Found 6 items
drwxr-xr-x   - hbase    supergroup          0 2022-02-14 18:34 /hbase
drwxr-xr-x   - cloudera supergroup          0 2022-02-14 18:44 /rjc
drwxr-xr-x   - solr     solr                0 2015-06-09 03:38 /solr
drwxrwxrwx   - hdfs     supergroup          0 2022-02-07 21:10 /tmp
drwxr-xr-x   - hdfs     supergroup          0 2015-06-09 03:38 /user
drwxr-xr-x   - hdfs     supergroup          0 2015-06-09 03:36 /var
[cloudera@quickstart ~]$ hdfs dfs -ls /
Found 6 items
drwxr-xr-x   - hbase    supergroup          0 2022-02-14 18:34 /hbase
drwxr-xr-x   - cloudera supergroup          0 2022-02-14 18:44 /rjc
drwxr-xr-x   - solr     solr                0 2015-06-09 03:38 /solr
drwxrwxrwx   - hdfs     supergroup          0 2022-02-07 21:10 /tmp
drwxr-xr-x   - hdfs     supergroup          0 2015-06-09 03:38 /user
drwxr-xr-x   - hdfs     supergroup          0 2015-06-09 03:36 /var
[cloudera@quickstart ~]$
```

**4) copyFromLocal Command**

First we will Create a text file in local.
And add some text into it.

Now I am trying to copy the ‘one.txt’ file present in the local file system to the ‘rjmscstat’ directory of Hadoop.
The below command copies the file from the local file system to HDFS.



```
[cloudera@quickstart ~]$ gedit one.txt
[cloudera@quickstart ~]$ cat one.txt
Hello everyone, How are you all.
This is BDA practical journel
[cloudera@quickstart ~]$
```

```
[cloudera@quickstart ~]$ hdfs dfs -copyFromLocal /home/cloudera/one.txt rjmscstat
[cloudera@quickstart ~]$ hdfs dfs -ls rjmscstat/
Found 1 items
-rw-r--r--   1 cloudera cloudera         63 2022-09-22 23:27 rjmscstat/one.txt
```

**4) If getting any error due to permissions**

Use-> export HADOOP_USER_NAME=hdfs

**5) Put Command**

-> hdfs dfs -put /home/cloudera/Desktop/ file_02 /rjc

Here in this example, we are trying to copy “file_02” of the local file system to the Hadoop file system.

The Hadoop fs shell command put is similar to the copyFromLocal, which copies files or directory from the local filesystem to the destination in the Hadoop filesystem.

```
[cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/txtfile.txt rjmscstat
[cloudera@quickstart ~]$ hdfs dfs -ls rjmscstat/
Found 2 items
-rw-r--r--   1 cloudera cloudera         63 2022-09-22 23:27 rjmscstat/one.txt
-rw-r--r--   1 cloudera cloudera         47 2022-09-22 23:43 rjmscstat/txtfile.txt
[cloudera@quickstart ~]$
```

**6) copyToLocal Command**

**copyToLocal** command copies the file from HDFS to the local file system

Here in this example, we are trying to copy the ‘file_01 file present in the rjc directory of HDFS

to the local file system.

Deleted Sample file from desktop. If it is already exist. And then again run the command.

```
[cloudera@quickstart ~]$ hdfs dfs -copyToLocal rjmscstat/one.txt
[cloudera@quickstart ~]$ ls
cloudera-manager  Desktop    Downloads  enterprise-deployment.json  kerberos  Music    Pictures              pig_1663148736943.log  rjmscstat  txtfile.txt   Videos
cm_api.py         Documents  eclipse    express-deployment.json     lib       one.txt  pig_1663139566177.log  Public                Templates  txtfile.txt~  workspace
[cloudera@quickstart ~]$
```

**7) CAT Command**

**->hdfs dfs -cat rjmscstat/one.txt**

we are using the cat command to display the content of the 'Sample_01' file present in rjc directory of HDFS

The cat command reads the file in HDFS and displays the content of the file on console or stdout**.**

```
[cloudera@quickstart ~]$ hdfs dfs -cat rjmscstat/one.txt
Hello everyone, How are you all.
This is BDA practical journel
```

## 8) Cp Command

```
[cloudera@quickstart ~]$ hdfs dfs -cat rjmscstat/one.txt
Hello everyone, How are you all.
This is BDA practical journel
[cloudera@quickstart ~]$ hdfs df -mkdir new_folder
Error: Could not find or load main class df
[cloudera@quickstart ~]$ hdfs dfs -mkdir new_folder
[cloudera@quickstart ~]$ hdfs dfs -ls
Found 6 items
drwxr-xr-x   - cloudera cloudera          0 2022-09-14 02:33 .Trash
drwxr-xr-x   - cloudera cloudera          0 2022-09-23 00:42 new_folder
drwxr-xr-x   - cloudera cloudera          0 2022-09-23 00:02 rjcnew
drwxr-xr-x   - cloudera cloudera          0 2022-09-22 23:43 rjmscstat
-rw-r--r--   1 cloudera cloudera         63 2022-09-23 00:40 rjnew
drwxr-xr-x   - cloudera cloudera          0 2022-09-14 00:04 training
[cloudera@quickstart ~]$ hdfs dfs -cp rjmscstat/one.txt new_folder
[cloudera@quickstart ~]$ hdfs dfs -ls new_folder
Found 1 items
-rw-r--r--   1 cloudera cloudera         63 2022-09-23 00:42 new_folder/one.txt
```

## 9) MV Command

```
[cloudera@quickstart ~]$ hdfs dfs -mv new_folder/one.txt move_folder
[cloudera@quickstart ~]$ hdfs dfs -ls move_folder
Found 1 items
-rw-r--r--   1 cloudera cloudera         63 2022-09-23 00:42 move_folder/one.txt
[cloudera@quickstart ~]$ hdfs dfs -ls new_folder
[cloudera@quickstart ~]$
```

## 10) RM Command

```
[cloudera@quickstart ~]$ hdfs dfs -rm -r new_folder
22/09/23 00:50:21 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted new_folder
[cloudera@quickstart ~]$ hdfs dfs -rm -r move_folder
22/09/23 00:50:39 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted move_folder
[cloudera@quickstart ~]$
```

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



## Practical 2 - HiveQL

a) List the existing database of hive.

```
hive> show databases;
OK
default
Time taken: 0.21 seconds, Fetched: 1 row(s)
hive>
```

b) Create a database named userdb and verify a database list.

```
hive> create database userdb;
OK
Time taken: 0.972 seconds
hive> show databases;
OK
default
userdb
Time taken: 0.017 seconds, Fetched: 2 row(s)
hive>
```

c) Display the schema of the userdb database.

```
hive> show tables;
OK
Time taken: 0.088 seconds
hive>
```

d) Delete the tables of the userdb database and drop the database.

```
hive> drop table student;
OK
Time taken: 0.027 seconds
hive> drop database userdb;
OK
Time taken: 0.15 seconds
hive>
```

e) Display the existing table of the userdb database.

```
hive> show tables;
OK
Time taken: 0.009 seconds
hive>
```

f) Demonstrate the alter command for the following tasks:

i) Rename the table.

```
hive> create table student(stud_id int, Name string, Roll_no int);
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. AlreadyExistsException(message:Table student already exists)
hive> alter table student RENAME TO stud_info;
OK
Time taken: 0.122 seconds
hive>
```

```
hive> show tables;
OK
stud_info
Time taken: 0.091 seconds, Fetched: 1 row(s)
hive>
```

ii) Add columns to the existing table.

```
hive> alter table stud_info ADD COLUMNS(Class string);
OK
Time taken: 0.089 seconds
hive> describe stud_info;
OK
stud_id                 int
name                    string
roll_no                 int
class                   string
Time taken: 0.062 seconds, Fetched: 4 row(s)
hive>
```

iii) Remove a column from the table.

```
hive> alter table stud_info REPLACE COLUMNS(Stud_id int, Name string, Roll_no int);
OK
Time taken: 0.091 seconds
hive> describe stud_info;
OK
stud_id                 int
name                    string
roll_no                 int
Time taken: 0.042 seconds, Fetched: 3 row(s)
hive>
```

iv) Renaming the column.

```
hive> describe stud_info;
OK
stud_id                 int
name                    string
roll_no                 int
Time taken: 0.042 seconds, Fetched: 3 row(s)
hive> alter table stud_info CHANGE stud_id id int;
OK
Time taken: 0.092 seconds
hive> describe stud_info;
OK
id                      int
name                    string
roll_no                 int
Time taken: 0.05 seconds, Fetched: 3 row(s)
hive>
```

v) Replacing the column.

```
hive> alter table stud_info CHANGE stud_id id int;
OK
Time taken: 0.092 seconds
hive> describe stud_info;
OK
id                      int
name                    string
roll_no                 int
Time taken: 0.05 seconds, Fetched: 3 row(s)
hive>
```

g) Assume we have the employee table as given below, with fields named Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000.

| ID | Name | Salary | Designation | Dept |
|---|---|---|---|---|
| 1201 | Gopal | 45000 | Technical manager | TP |
| 1202 | Manisha | 45000 | Proofreader | PR |
| 1203 | Masthanvali | 40000 | Technical writer | TP |
| 1204 | Krian | 40000 | Hr Admin | HR |
| 1205 | Kranthi | 30000 | Op Admin | Admin |

```
hive> CREATE TABLE employee(ID INT,Name STRING, Salary INT, Designation STRING, Dept STRING);
OK
Time taken: 0.135 seconds
hive> Describe employee;
OK
id                      int
name                    string
salary                  int
designation             string
dept                    string
Time taken: 0.138 seconds, Fetched: 5 row(s)
hive>
```

```
hive> SELECT * FROM employee;
OK
1201    Gopal   45000   Technical manager       TP
1202    Manisha 45000   Proofreader     PR
1203    Masthanvali     40000   Technical writer        TP
1204    Krian   40000   HR Admin        HR
1205    Kranthi 30000   OP Admin        Admin
Time taken: 0.28 seconds, Fetched: 5 row(s)
hive>
```

h) Using the above table solve the following queries.

i) Retrieve the employees details for the employees having salary greater than 30000.

```
hive> SELECT * FROM employee WHERE Salary>30000;
OK
1201    Gopal   45000   Technical manager       TP
1202    Manisha 45000   Proofreader     PR
1203    Masthanvali     40000   Technical writer        TP
1204    Krian   40000   HR Admin        HR
Time taken: 0.119 seconds, Fetched: 4 row(s)
hive>
```

ii) Retrieve the employees details in the sorted order of designation.

```
hive> SELECT * FROM employee ORDER BY Designation;
1204    Krian   40000   HR Admin        HR
1205    Kranthi 30000   OP Admin        Admin
1202    Manisha 45000   Proofreader     PR
1201    Gopal   45000   Technical manager       TP
1203    Masthanvali     40000   Technical writer        TP
Time taken: 47.722 seconds, Fetched: 5 row(s)
hive>
```

iii) Retrieve department wise employees details using group by clause.

```
hive> SELECT Dept, COUNT()FROM employee
    > GROUP BY dept;
Admin   1
HR      1
PR      1
TP      2
Time taken: 38.396 seconds, Fetched: 4 row(s)
hive>
```

iv) Retrieve the employee having maximum and minimum salary.

Minimum salary:

```
select min(salary) from employee;

OK
30000
Time taken: 32.775 seconds, Fetched: 1 row(s)
```

Maximum salary:

```
select max(salary) from employee;

OK
45000
Time taken: 33.182 seconds,
```

v) Get the list of employees in uppercase.

```
hive> select UPPER(name) from employee;
OK
GOPAL
MANISHA
MASTHANVALI
KRIAN
KRANTHI
Time taken: 0.099 seconds, Fetched: 5 row(s)
hive>
```

i) Create any two tables and demonstrate the various joining operations.

```
hive> SELECT * FROM employee;
OK
1201    Gopal   45000   Technical manager       TP
1202    Manisha 45000   Proofreader     PR
1203    Masthanvali     40000   Technical writer        TP
1204    Krian   40000   HR Admin        HR
1205    Kranthi 30000   OP Admin        Admin
Time taken: 0.129 seconds, Fetched: 5 row(s)
hive> describe employee;
OK
id                      int
name                    string
salary                  int
designation             string
dept                    string
Time taken: 0.16 seconds, Fetched: 5 row(s)
hive> CREATE TABLE Emp_loc(id int, Location String);
OK
Time taken: 0.148 seconds
hive> describe Emp_LOc;
OK
id                      int
location                string
Time taken: 0.115 seconds, Fetched: 2 row(s)
hive>
```

```
1201    Mumbai
1202    Delhi
1203    Ranchi
1204    Thane
1205    Patna
Time taken: 0.112 seconds, Fetched: 5 row(s)
hive>
```

**INNER JOIN**

```
hive> SELECT employee.id, Name, Location FROM employee
    > INNER JOIN Emp_Loc ON employee.id = Emp_Loc.id;
1201    Gopal   Mumbai
1202    Manisha Delhi
1203    Masthanvali     Ranchi
1204    Krian   Thane
1205    Kranthi Patna
Time taken: 48.975 seconds, Fetched: 5 row(s)
hive>
```

**LEFT OUTER JOIN**

```
hive> SELECT employee.id, Name, Location FROM employee
    > LEFT OUTER JOIN Emp_Loc ON employee.id = Emp_Loc.id;
1201    Gopal   Mumbai
1202    Manisha Delhi
1203    Masthanvali     Ranchi
1204    Krian   Thane
1205    Kranthi Patna
Time taken: 46.836 seconds, Fetched: 5 row(s)
hive>
```

**RIGHT OUTER JOIN**

```
hive> SELECT employee.id, Name, Location FROM employee
    > RIGHT OUTER JOIN Emp_Loc ON employee.id = Emp_Loc.id;
1201    Gopal   Mumbai
1202    Manisha Delhi
1203    Masthanvali     Ranchi
1204    Krian   Thane
1205    Kranthi Patna
Time taken: 49.105 seconds, Fetched: 5 row(s)
hive>
```

**FULL OUTER JOIN**

```
hive> SELECT employee.id, Name, Location FROM employee
    > FULL OUTER JOIN Emp Loc ON employee.id = Emp Loc.id;
```

```
1201    Gopal   Mumbai
1202    Manisha Delhi
1203    Masthanvali     Ranchi
1204    Krian   Thane
1205    Kranthi Patna
Time taken: 66.99 seconds, Fetched: 5 row(s)
hive>
```

j) Demonstrate the creation of an internal (managed) table, external table and temporary table.

**INTERNAL (Managed) TABLE**

```
hive> CREATE TABLE Student_info(Name string, Rol_no int, Gender string)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 0.437 seconds
hive> DESCRIBE FORMATED Student_info;
FAILED: SemanticException [Error 10001]: Table not found FORMATED
hive> DESCRIBE FORMATtED Student_info;
OK
# col_name              data_type               comment

name                    string
rol_no                  int
gender                  string

# Detailed Table Information
Database:               default
Owner:                  cloudera
CreateTime:             Fri Sep 23 11:49:19 PDT 2022
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/user/hive/warehouse/student_info
Table Type:             MANAGED_TABLE
Table Parameters:
        transient_lastDdlTime   1663958959
```

## EXTERNAL TABLE

```
hive> CREATE EXTERNAL TABLE Student_Ext(Name string, Rol_no int, Gender string)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 0.074 seconds
hive> DESCRIBE FORMATTED Student_Ext;
OK
# col_name              data_type               comment

name                    string
rol_no                  int
gender                  string

# Detailed Table Information
Database:               default
Owner:                  cloudera
CreateTime:             Fri Sep 23 12:08:02 PDT 2022
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/user/hive/warehouse/student_ext
Table Type:             EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                TRUE
        transient_lastDdlTime   1663960082
```

## TEMPORARY TABLE

```
hive> CREATE TEMPORARY TABLE Student_Ext(Name string, Rol_no int, Gender string)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 0.031 seconds
hive> DESCRIBE FORMATTED Student_Ext;
OK
# col_name              data_type               comment

name                    string
rol_no                  int
gender                  string

# Detailed Table Information
Database:               default
Owner:                  cloudera
CreateTime:             Fri Sep 23 12:10:21 PDT 2022
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/tmp/hive/cloudera/209a7ca1-7eaf-4b3e-83c0-1f4f
36f
Table Type:             MANAGED_TABLE
```

This temporary table is of type managed_table.

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



## Practical 3 - HBase

a) What is Hbase? Explain the data model of HBase.

**Hbase**

i. HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

ii. HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data.

**Hbase Data Model**

i. The Data Model in HBase is designed to accommodate semi-structured data that could vary in field size, data type, and columns.

ii. Additionally, the layout of the data model makes it easier to partition the data and distribute it across the cluster.

iii. The Data Model in HBase is made of different logical components such as Tables, Rows, Column Families, Columns, Cells, and Versions.

| Row Key | Customer | | Sales | |
|---|---|---|---|---|
| Customer Id | Name | City | Product | Amount |
| 101 | John White | Los Angeles, CA | Chairs | $400.00 |
| 102 | Jane Brown | Atlanta, GA | Lamps | $200.00 |
| 103 | Bill Green | Pittsburgh, PA | Desk | $500.00 |
| 104 | Jack Black | St. Louis, MO | Bed | $1600.00 |

Column Families

**Tables:** The HBase Tables are more like a logical collection of rows stored in separate partitions called Regions. As shown above, every Region is then served by exactly one Region Server. The figure above shows a representation of a Table.

**Rows:** A row is one instance of data in a table and is identified by a row key. Row keys are unique in a Table and are always treated as a byte[].
Column Families – Data in a row are grouped together as Column Families. Each Column Family has one more Column and these Columns in a family are stored together in a low-level storage file known as HFile. Column Families form the basic unit of physical storage to which certain HBase features like compression are applied. Hence, it's important that proper care be taken when designing the Column Families in the table. The table above shows Customer and Sales Column Families. The Customer Column Family is made up of 2 columns – Name and City, whereas the Sales Column Families is made up of 2 columns – Product and Amount.

**Columns:** A Column Family is made of one or more columns. A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon – example: columnfamily:columnname. There can be multiple Columns within a Column Family and Rows within a table can have a varied number of Columns.
Cell – A Cell stores data and is essentially a unique combination of rowkey, Column Family and the Column (Column Qualifier). The data stored in a Cell is called its value and the data type is always treated as byte[].

**Version:** The data stored in a cell is versioned and versions of data are identified by the timestamp. The number of versions of data retained in a column family is configurable and this value by default is 3.

b) Create the database tables in HBase and demonstrate the various DDL and DML commands.

**1) DDL Commands - Data Definition Language**

1. **Create Table**

```
hbase(main):022:0> create 'student','roll_no','name','place'
0 row(s) in 0.4380 seconds

=> Hbase::Table - student
```

2. **List**

```
hbase(main):006:0> list
TABLE
student
1 row(s) in 0.0560 seconds
```

3. **Disable**

```
=> ["student"]
hbase(main):007:0> disable 'student'
0 row(s) in 1.4130 seconds
```

4. **is_disabled**

```
hbase(main):008:0> is_disabled 'student'
true
0 row(s) in 0.0150 seconds
```

5. **enable**

```
hbase(main):009:0> enable 'student'
0 row(s) in 0.5290 seconds
```

6. **is_enabled**

```
hbase(main):010:0> is_enabled 'student'
true
0 row(s) in 0.0520 seconds
```

7. **Describe**

```
hbase(main):011:0> describe 'student'
Table student is ENABLED
student
COLUMN FAMILIES DESCRIPTION
{NAME => 'name', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', T
TL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'place', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'roll_no', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0'
, TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
3 row(s) in 0.0730 seconds
```

8. **Alter**

**Delete Column**

```
hbase(main):012:0> alter 'student','delete'=>'place'
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.4180 seconds
```

9. **exists - verifies whether table exists or not**

```
hbase(main):014:0> exists 'student'
Table student does exist
0 row(s) in 0.0390 seconds
```

10. **drop table**

```
hbase(main):015:0> disable 'student'
0 row(s) in 1.3070 seconds

hbase(main):016:0> drop 'student'
0 row(s) in 0.3650 seconds
```

**2) DML - Data Manipulation Language**

1. **Put Command**

```
hbase(main):024:0> put 'student','1','name','Ashish'
0 row(s) in 0.0200 seconds

hbase(main):025:0> put 'student','1','roll_no','903'
0 row(s) in 0.0400 seconds

hbase(main):026:0> scan 'student'
ROW                                   COLUMN+CELL
 1                                    column=name:, timestamp=1664099340685, value=Ashish
 1                                    column=roll_no:, timestamp=1664099358517, value=903
1 row(s) in 0.0810 seconds

hbase(main):027:0>
```

2. **Get Command**

```
hbase(main):030:0> get 'student','1','name'
COLUMN                                  CELL
 name:                                  timestamp=1664099340685, value=Ashish
1 row(s) in 0.0230 seconds
```

3. **Delete And scan**

```
hbase(main):031:0> delete 'student','1','roll_no'
0 row(s) in 0.0450 seconds

hbase(main):032:0> scan 'student'
ROW                                     COLUMN+CELL
 1                                      column=name:, timestamp=1664099340685, value=Ashish
1 row(s) in 0.0340 seconds
```

4. **Count - count and return the number of rows**

```
hbase(main):033:0> count 'student'
1 row(s) in 0.0420 seconds

=> 1
```

5. **truncate**

```
hbase(main):034:0> truncate 'student'
Truncating 'student' table (it may take a while):
 - Disabling table...
 - Truncating table...
0 row(s) in 1.5370 seconds

hbase(main):035:0> scan 'student'
ROW                                     COLUMN+CELL
0 row(s) in 0.3220 seconds
```

## Practical 4 - Pig

### a) What is Pig and Piglatin?

- **Apache Pig** is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.
- The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.
- The language used for Pig is Pig Latin. The Pig scripts get internally converted to Map Reduce jobs and get executed on data stored in HDFS.
- Apart from that, Pig can also execute its job in Apache Tez or Apache Spark.
- Pig can handle any type of data, i.e., structured, semi-structured or unstructured and stores the corresponding results into the Hadoop Data File System.
- Every task which can be achieved using PIG can also be achieved using java used in MapReduce.

❖ Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:

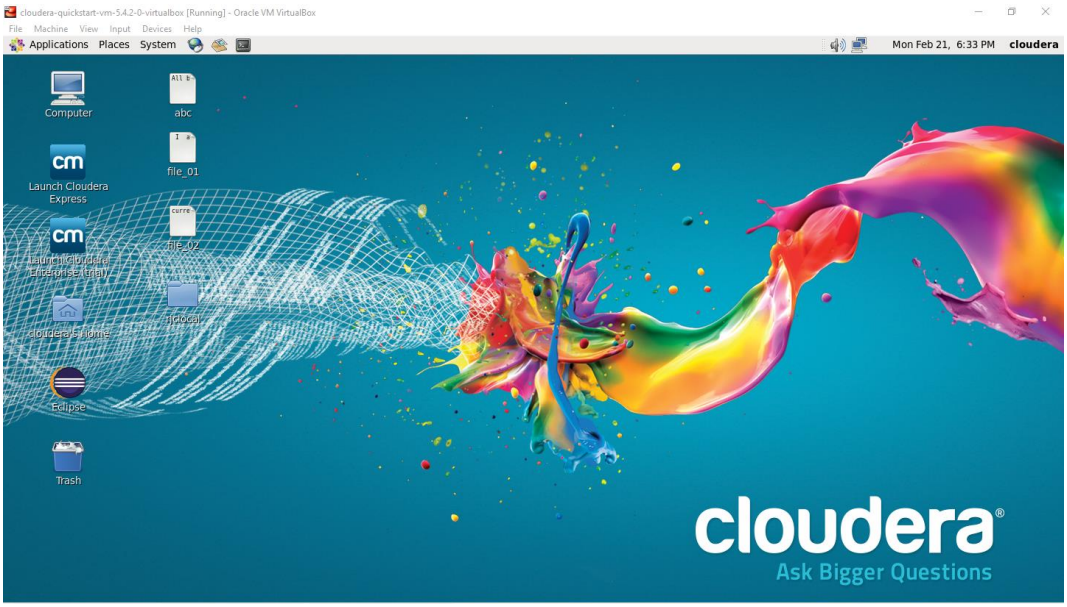
- ➢ **Ease of programming.** It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks composed of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.
- ➢ **Optimization opportunities.** The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
- ➢ **Extensibility.** Users can create their own functions to do special-purpose processing.

**b) Demonstrate the pig commands with an example for each.**

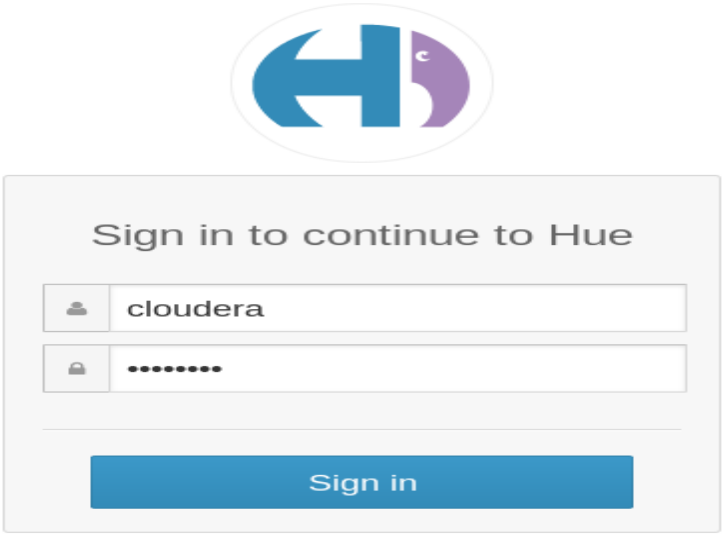
## To implement Word Count problem using Pig

## Steps:

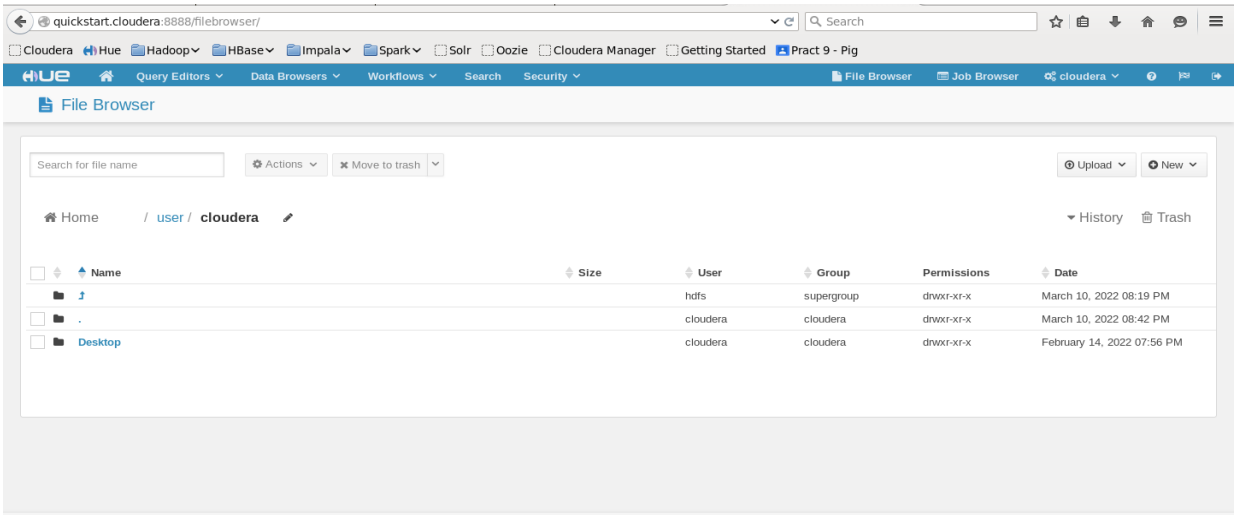
1. **Start the cloudera.**



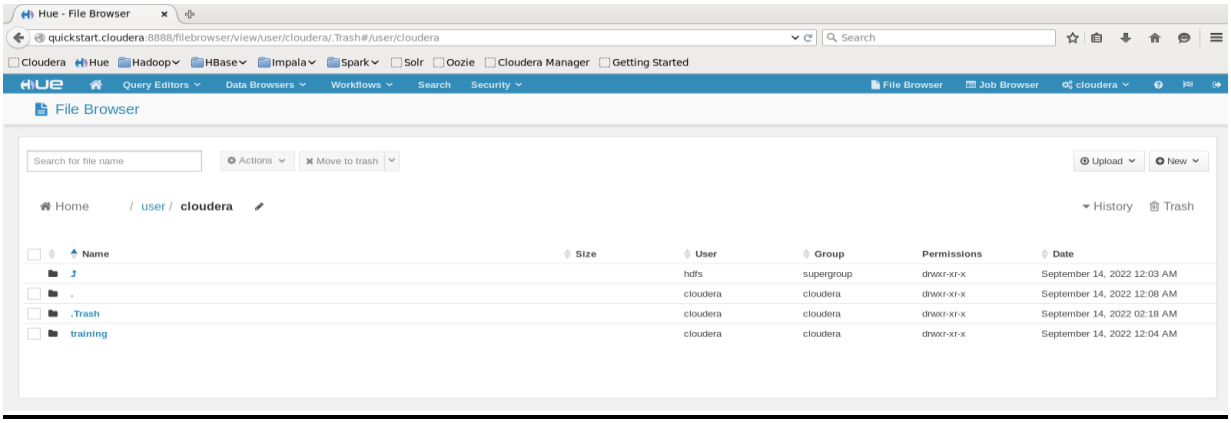

2. **Open the browser. And then open Hue and login.**



3. **In Hue Go to file browser and Now open the directory /user/cloudera**




4. **Now we are creating the directory as training inside /user/cloudera**



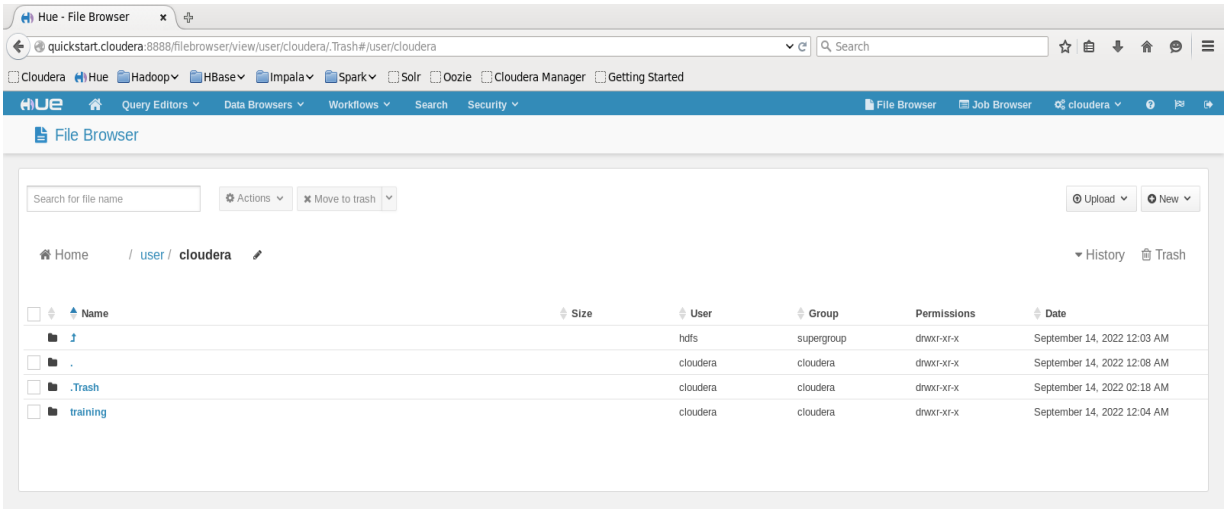

In File Browser we have New option in right corner

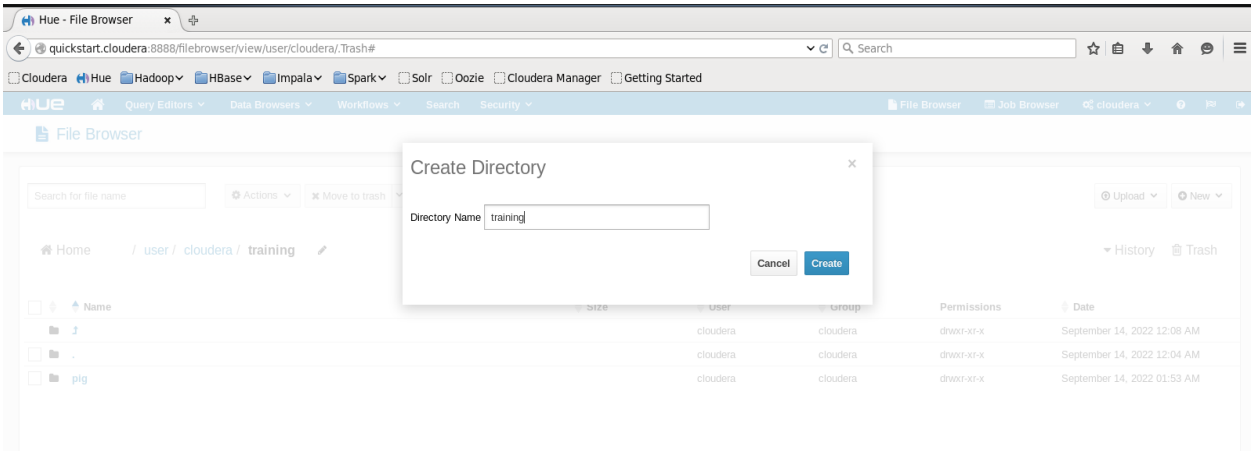
Click on New Directory

Give the directory name And click on Create

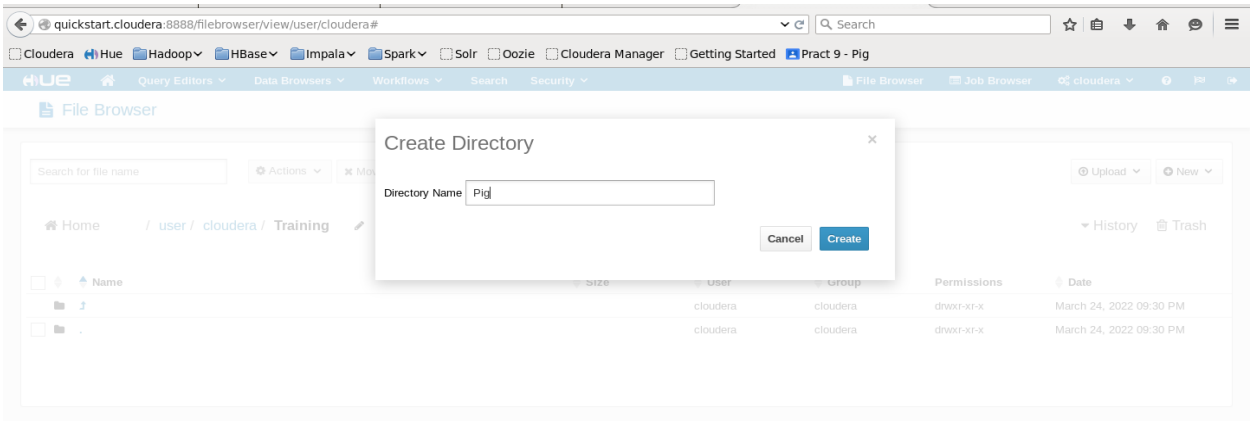
Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



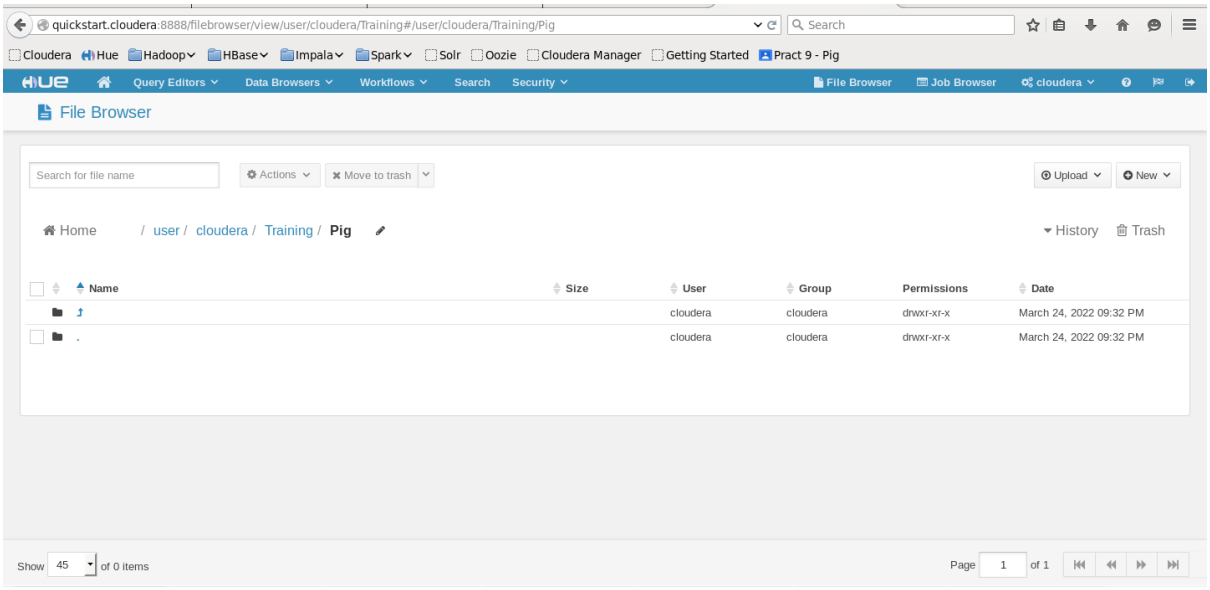



5. **<u>After creating training directory now creating the Pig directory inside training.</u>**

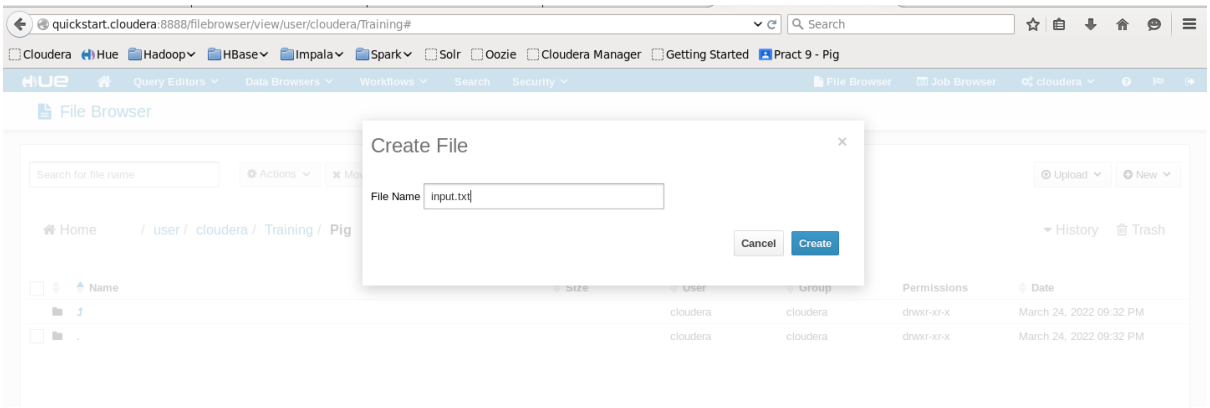


## 6. Pig directory has been created inside /user/cloudera/training



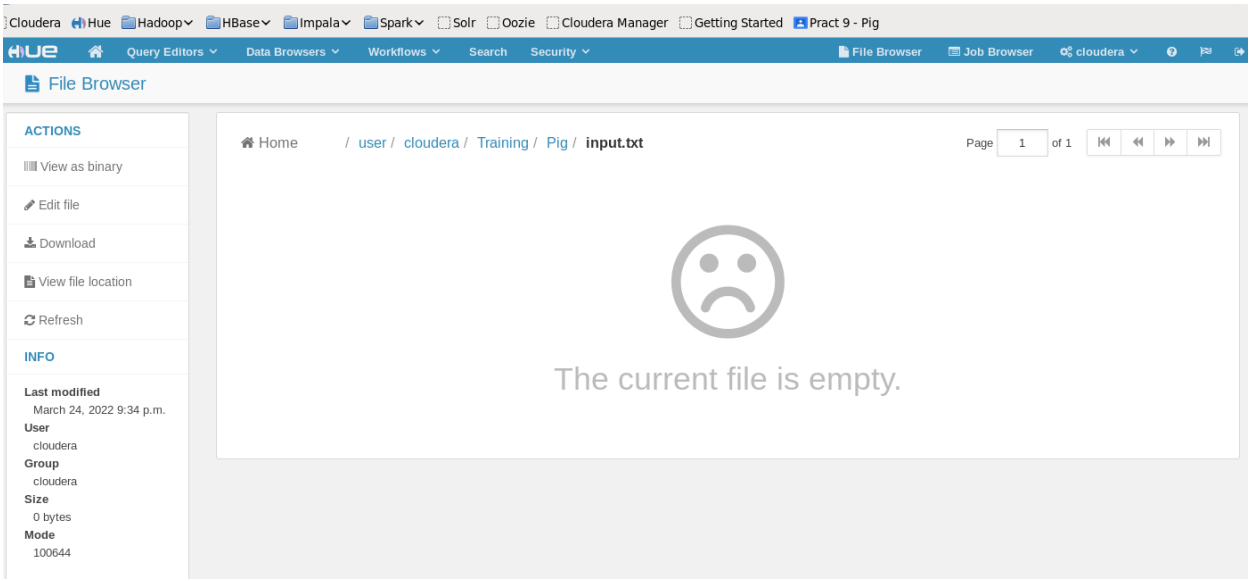

## 7. Creating input.txt file inside /usr/cloudera/training/Pig directory

Again inside the Pig directory click on New and create file as'input.txt'




Once the file has been created click on 'input.txt' to add the content in it

## 8. Adding some contents to this input.txt file.




For adding content in the input file, Click on ‘Edit file’ option then add the content.

Save the input.txt file

## 9. Now Open the terminal. And start Pig by typing pig on terminal.

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ pig
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2022-09-14 02:34:59,505 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.4.2 (rexported) compiled May 19 2015, 17:03:41
2022-09-14 02:34:59,505 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/cloudera/pig_1663148099486.log
2022-09-14 02:34:59,538 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/cloudera/.pigbootup not found
2022-09-14 02:34:59,991 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress
2022-09-14 02:34:59,992 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:34:59,992 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://quickstart.c
loudera:8020
2022-09-14 02:35:00,881 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress
2022-09-14 02:35:00,881 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to map-reduce job tracker at: localhost:8021
2022-09-14 02:35:00,881 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:35:00,940 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:35:00,942 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress


2022-09-14 02:35:01,199 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:35:01,199 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress
2022-09-14 02:35:01,265 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:35:01,266 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress
2022-09-14 02:35:01,344 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:35:01,344 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress
2022-09-14 02:35:01,415 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:35:01,415 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.ad
dress
grunt>
```

**10. Now we have to load that input file where ever it is stored. By typing the command**

```
grunt> input1 = LOAD '/user/cloudera/training/pig/input.txt' AS (f1:chararray);
grunt> DUMP input1;
```

**11. Now we are dumping the data.**

It will do the MapReduce task. The Dump operator is used to run the Pig Latin statements and display the results on the screen. It is generally used for debugging Purpose.

```
2022-09-14 02:40:47,021 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: UNKNOWN
2022-09-14 02:40:47,086 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParall
elSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInser
ter], RULES_DISABLED=[FilterLogicExpressionSimplifier, PartitionFilterOptimizer]}
2022-09-14 02:40:47,180 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false
2022-09-14 02:40:47,196 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1
2022-09-14 02:40:47,196 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1
2022-09-14 02:40:47,263 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2022-09-14 02:40:47,469 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig script settings are added to the job
2022-09-14 02:40:47,521 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.reduce.markreset.buffer.percent is deprecated. Instead, use mapreduce.reduce.markreset.buffer.perc
ent
2022-09-14 02:40:47,521 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - mapred.job.reduce.markreset.buffer.percent is not set, set to default 0.3
2022-09-14 02:40:47,521 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.output.compress is deprecated. Instead, use mapreduce.output.fileoutputformat.compress
2022-09-14 02:40:48,056 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - creating jar file Job6288332931726524521.jar
2022-09-14 02:40:50,421 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - jar file Job6288332931726524521.jar created
2022-09-14 02:40:50,421 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.jar is deprecated. Instead, use mapreduce.job.jar
2022-09-14 02:40:50,446 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.JobControlCompiler - Setting up single store job
2022-09-14 02:40:50,464 [main] INFO  org.apache.pig.data.SchemaTupleFrontend - Key [pig.schematuple] is false, will not generate code.
2022-09-14 02:40:50,464 [main] INFO  org.apache.pig.data.SchemaTupleFrontend - Starting process to move generated code to distributed cache
2022-09-14 02:40:50,464 [main] INFO  org.apache.pig.data.SchemaTupleFrontend - Setting key [pig.schematuple.classes] with classes to deserialize []
2022-09-14 02:40:50,502 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 1 map-reduce job(s) waiting for submission.
2022-09-14 02:40:50,503 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker.http.address is deprecated. Instead, use mapreduce.jobtracker.http.address
2022-09-14 02:40:50,503 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-09-14 02:40:50,525 [JobControl] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2022-09-14 02:40:50,599 [JobControl] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:40:50,933 [JobControl] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-09-14 02:40:50,933 [JobControl] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
2022-09-14 02:40:50,949 [JobControl] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths (combined) to process : 1
2022-09-14 02:40:50,995 [JobControl] INFO  org.apache.hadoop.mapreduce.JobSubmitter - number of splits:1
2022-09-14 02:40:51,163 [JobControl] INFO  org.apache.hadoop.mapreduce.JobSubmitter - Submitting tokens for job: job_1662450770751_0026
2022-09-14 02:40:51,338 [JobControl] INFO  org.apache.hadoop.yarn.client.api.impl.YarnClientImpl - Submitted application application_1662450770751_0026
2022-09-14 02:40:51,378 [JobControl] INFO  org.apache.hadoop.mapreduce.Job - The url to track the job: http://quickstart.cloudera:8088/proxy/application_1662450770751_0026/

2022-09-14 02:41:07,067 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2022-09-14 02:41:07,069 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:41:07,069 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-09-14 02:41:07,070 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2022-09-14 02:41:07,081 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-09-14 02:41:07,081 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Hello Everyone my name is Ashishkumar Rana)
(from Mumbai Maharashtra)
(Student of Ramniranjan Jhunjhunwala College)
(This is Big Data Analytics Practical)
grunt>
```

**12. Here we are counting the words in each line for that we are using the following command**

```
grunt> wordsInEachLine = FOREACH input1 GENERATE flatten(TOKENIZE(f1)) as word;
2022-09-14 02:49:50,715 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:49:50,715 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
grunt> DUMP wordsInEachLine;
```

**13. Again, we are dumping the data. It will do the MapReduce task. dump wordsInEachLine;**

```
2022-09-14 02:51:07,374 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2022-09-14 02:51:07,374 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 02:51:07,374 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-09-14 02:51:07,375 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2022-09-14 02:51:07,384 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-09-14 02:51:07,384 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Hello)
(Everyone)
(my)
(name)
(is)
(Ashishkumar)
(Rana)
(from)
(Mumbai)
(Maharashtra)
(Student)
(of)
(Ramniranjan)
(Jhunjhunwala)
(College)
(This)
(is)
(Big)
(Data)
(Analytics)
(Practical)
grunt>
```

## 14. Now grouping the words present in each line.

groupedWords = group wordsInEachLine by word;

```
grunt> groupedWords = group wordsInEachLine by word;
grunt> DUMP groupedWords;
```

And then dumping the data by the following command.

dump groupedWords;

```
2022-09-14 03:02:51,272 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2022-09-14 03:02:51,272 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 03:02:51,272 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-09-14 03:02:51,274 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2022-09-14 03:02:51,279 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-09-14 03:02:51,279 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(is,{(is),(is)})
(my,{(my)})
(of,{(of)})
(Big,{(Big)})
(Data,{(Data)})
(Rana,{(Rana)})
(This,{(This)})
(from,{(from)})
(name,{(name)})
(Hello,{(Hello)})
(Mumbai,{(Mumbai)})
(College,{(College)})
(Student,{(Student)})
(Everyone,{(Everyone)})
(Analytics,{(Analytics)})
(Practical,{(Practical)})
(Ashishkumar,{(Ashishkumar)})
(Maharashtra,{(Maharashtra)})
(Ramniranjan,{(Ramniranjan)})
(Jhunjhunwala,{(Jhunjhunwala)})
grunt>
```

## 15. Now we count those words. For each group we count words in each line.

countedWords = foreach groupedWords generate group, COUNT(wordsInEachLine);

```
grunt> countWords = FOREACH groupedWords GENERATE group, COUNT(wordsInEachLine);
grunt> DUMP countWords;
```

**16. After every counting of words commands,**

we are dumping the data dump countedWords;

Now the Final Output we are getting as word count for every word.

```
2022-09-14 03:11:20,445 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2022-09-14 03:11:20,446 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 03:11:20,446 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-09-14 03:11:20,446 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2022-09-14 03:11:20,452 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-09-14 03:11:20,452 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(is,2)
(my,1)
(of,1)
(Big,1)
(Data,1)
(Rana,1)
(This,1)
(from,1)
(name,1)
(Hello,1)
(Mumbai,1)
(College,1)
(Student,1)
(Everyone,1)
(Analytics,1)
(Practical,1)
(Ashishkumar,1)
(Maharashtra,1)
(Ramniranjan,1)
(Jhunjhunwala,1)
grunt>
```

**17. Now Exit from the grunt shell using quit command.**

```
grunt> quit
[cloudera@quickstart ~]$
```

**c) Write a pig script to count the occurrence of each word for the given text.**


```
2022-09-14 03:11:20,445 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2022-09-14 03:11:20,446 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2022-09-14 03:11:20,446 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-09-14 03:11:20,446 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2022-09-14 03:11:20,452 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2022-09-14 03:11:20,452 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(is,2)
(my,1)
(of,1)
(Big,1)
(Data,1)
(Rana,1)
(This,1)
(from,1)
(name,1)
(Hello,1)
(Mumbai,1)
(College,1)
(Student,1)
(Everyone,1)
(Analytics,1)
(Practical,1)
(Ashishkumar,1)
(Maharashtra,1)
(Ramniranjan,1)
(Jhunjhunwala,1)
grunt>
```

**Relations**

Ex- Create Student.txt file as

John,18,4.0F
Mary,19,3.8F
Bill,20,3.9F
Joe,18,3.8F

```
grunt> A = LOAD '/user/cloudera/Practice/Pig/student.txt' USING PigStorage(',') AS(name:chararray, age:int, gpa:float);
grunt> DUMP A;
(John,18,4.0)
(Mary,19,3.8)
(Bill,20,3.9)
(Joe,18,3.8)
grunt>
```

```
grunt> X = FOREACH A GENERATE name, $2;
grunt> DUMP X;
(John,4.0)
(Mary,3.8)
(Bill,3.9)
(Joe,3.8)
grunt>
```

**To add Age and gpa Column**

```
grunt> Y = FOREACH A GENERATE gpa+age;
grunt> DUMP Y;
2022-09-14 03:48:55,818 [main] INFO
(22.0)
(22.8)
(23.9)
(21.8)
```

**Tuple**

A tuple is an ordered set of fields.
Create a data.txt file as,

(3,8,9) (4,5,6)
(1,4,7) (3,7,5)
(2,5,8) (9,5,8)

```
grunt> A = LOAD '/user/cloudera/Practice/Pig/data' USING PigStorage(' ') AS
>> (t1:tuple(t1a:int,t1b:int,t1c:int), t2:tuple(t2a:int,t2b:int,t2c:int));
grunt> DUMP A;
((3,8,9),(4,5,6))
((1,4,7),(3,7,5))
((2,5,8),(9,5,8))
grunt>
```

## Practical 5 - Handling Database in MongoDB

### a) Installation of MongoDB

1. Write a note on NoSQL and MongoDB

NoSQL databases store data in documents rather than relational tables. Accordingly, we classify

them as "not only SQL" and subdivide them by a variety of flexible data models. Types of NoSQL

databases include pure document databases, key-value stores, wide-column databases, and

graph databases. NoSQL databases are built from the ground up to store and process vast

amounts of data at scale and support a growing number of modern businesses.

MongoDB is a source-available cross-platform document-oriented database program. Classified

as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License

which is deemed non-free by several distributions.

2. Differentiate between SQL and NoSQL

| SQL | NoSQL |
|---|---|
| RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS) | Non-relational or distributed database system. |
| These databases have fixed or static or predefined schema | They have dynamic schema |
| These databases are not suited for hierarchical data storage. | These databases are best suited for hierarchical data storage. |
| These databases are best suited for complex queries | These databases are not so good for complex queries |
| Vertically Scalable | Horizontally scalable |
| Follows ACID property | Follows CAP(consistency, availability, partition tolerance) |
| **Examples:** MySQL, PostgreSQL, Oracle, MS-SQL Server etc | **Examples:** MongoDB, GraphQL, HBase, Neo4j, Cassandra etc |

3. Types of databases in NoSQL

# Document databases

A document database stores data in JSON, BSON, or XML documents (not Word documents or Google Docs, of course). In a document database, documents can be nested. Particular elements can be indexed for faster querying.Documents can be stored and retrieved in a form that is much closer to the data objects used in applications, which means less translation is required to use the data in an application. SQL data must often be assembled and disassembled when moving back and forth between applications and storage.

# Key-value stores

The simplest type of NoSQL database is a key-value store. Every data element in the database is stored as a key value pair consisting of an attribute name (or "key") and a value. In a sense, a key-value store is like a relational database with only two columns: the key or attribute name (such as "state") and the value (such as "Alaska").

Use cases include shopping carts, user preferences, and user profiles.

# Column-oriented databases

While a relational database stores data in rows and reads data row by row, a column store is organized as a set of columns. This means that when you want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data. Columns are often of the same type and benefit from more efficient compression, making reads even faster. Columnar databases can quickly aggregate the value of a given column (adding up the total sales for the year, for example). Use cases include analytics.

# Graph databases

A graph database focuses on the relationship between data elements. Each element is stored as a node (such as a person in a social media graph). The connections between elements are called links or relationships. In a graph database, connections are first-class elements of the database, stored directly. In relational databases, links are implied, using data to express the relationships.

A graph database is optimized to capture and search the connections between data elements, overcoming the overhead associated with JOINing multiple tables in SQL.

4. Write a note on MongoDB Server, MongoDB Compass and MongoDB Shell

**MongoDB Server:**

MongoDB is an open source NoSQL database management program. NoSQL is used as an alternative to traditional relational databases. NoSQL databases are quite useful

for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.MongoDB supports various forms of data.MongoDB makes use of records which are made up of documents that contain a data structure composed of field and value pairs. Documents are the basic unit of data in MongoDB. The documents are similar to JavaScript Object Notation, but use a variant called Binary JSON (BSON). Values contained can be a variety of data types, including other documents, arrays and arrays of documents, according to the MongoDB user manual. Documents will also incorporate a primary key as a unique identifier.

**MongoDB Compass :**

MongoDB Compass is a much better alternative for the Mongo shell. Compass can carry out all the operations that Mongo Shell does and more, including:

- Visualize and explore data stored in your database
- Create databases and Insert, update, and delete data in your database
- Get immediate real-time server statistics
- Understand performance issues with visual explain plans
- Manage your indexes
- Validate your data with JSON schema validation rules
- Extendable via plugins

You can enjoy the benefits of MongoDB's powerful features by installing the full version of Compass. It's free to use for everyone and will make working with MongoDB easier than any other tool Now, let's install MongoDB Compass.

**MongoDB Shell :**

MongoDB Mongo shell is an interactive JavaScript interface that allows you to interact with MongoDB instances through the command line. The shell can be used for:

Data manipulation Administrative operations such as maintenance of database instancesMongoDB Mongo shell is the default client for the MongoDB database server. It's a command-line interface (CLI), where the input and output are all console-based. The Mongo shell is a good tool to manipulate small sets of data.

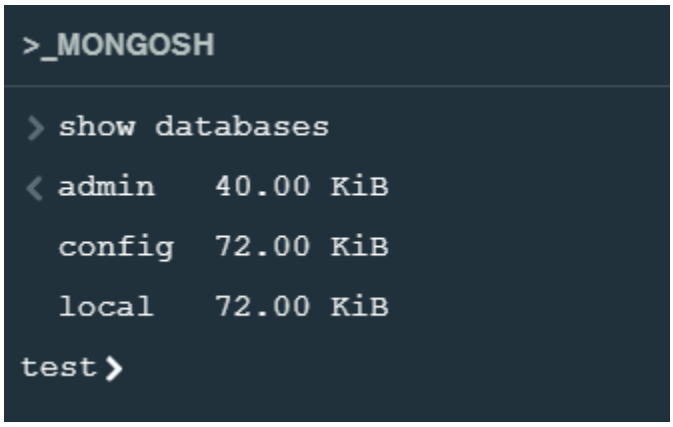
Here are the top features that Mongo shell offers:

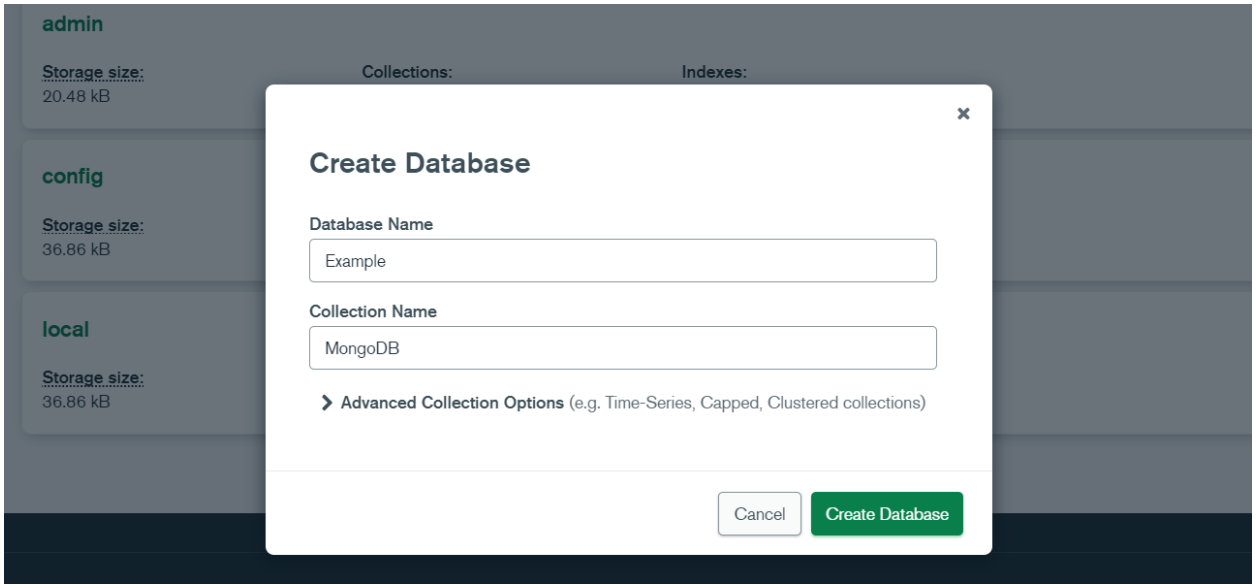
1. Run all MongoDB queries from the Mongo shell.
2. Manipulate data and perform administration operations.
3. Mongo shell uses JavaScript and a related API to issue commands.
4. See previous commands in the mongo shell with up and down arrow keys.
5. View possible command completions using the tab button after partially entering a command.
6. Print error messages, so you know what went wrong with your commands.

5. Install MongoDB Community edition.

**b) MongoDB Shell Commands**

1. List the existing databases and Create database named ‘Example’ and Check the list of databases again.






```
> use Example
< 'switched to db Example'
```

2. List the database again for the newly created database ‘Example’

```
> show databases
< Example   8.00 KiB
  admin    40.00 KiB
  config   96.00 KiB
  local    72.00 KiB
Example >
```

3. Use database ‘Example’.

```
> use Example
< 'switched to db Example'
```

4. List the existing Collections and Create Collection named ‘ExCollection’,

```
> show collections
< MongoDB
> db.createCollection("ExCollection")
< { ok: 1 }
> show collections
< ExCollection
  MongoDB
Example >
```

5. List the collections again for the newly created collection ‘ExCollection’.

```
> show collections
< ExCollection
  MongoDB
Example >
```

6. Insert the following Document into the ‘ExCollection’ Collection.

‘{field1:”Value1”, field2:”value2”}’

```
> db.ExCollection.insertOne({field1:"value1", field2:"value2"})
< { acknowledged: true,
    insertedId: ObjectId("632b2d501ef5504ea2862b1f") }
```

7. Delete Collection named ‘ExCollection’.

```
> db.ExCollection.drop()
< true
Example >
```

8. Delete database named ‘Example’.

```
> db.dropDatabase()
< { ok: 1, dropped: 'Example' }
Example>
```

9. Get the help on mongosh or mongo commands using help command?

i) Get the help on db command.

```
> db.help()
< Database Class

  getMongo                    Returns the current database connection
  getName                     Returns the name of the DB
  getCollectionNames          Returns an array containing the names of all collections in the current database.
  getCollectionInfos          Returns an array of documents with collection information, i.e. collection name and options, for the current database.
  runCommand                  Runs an arbitrary command on the database.
  adminCommand                Runs an arbitrary command against the admin database.
  aggregate                   Runs a specified admin/diagnostic pipeline which does not require an underlying collection.
  getSiblingDB                Returns another database without modifying the db variable in the shell environment.
  getCollection               Returns a collection or a view object that is functionally equivalent to using the db.<collectionName>.
  dropDatabase                Removes the current database, deleting the associated data files.
  createUser                  Creates a new user for the database on which the method is run. db.createUser() returns a duplicate user error if the user already exists on
                              the database.
  updateUser                  Updates the user's profile on the database on which you run the method. An update to a field completely replaces the previous field's values.
                              This includes updates to the user's roles array.
  changeUserPassword          Updates a user's password. Run the method in the database where the user is defined, i.e. the database you created the user.
  logout                      Ends the current authentication session. This function has no effect if the current session is not authenticated.
  dropUser                    Removes the user from the current database.
  dropAllUsers                Removes all users from the current database.
  auth                        Allows a user to authenticate to the database from within the shell.
  grantRolesToUser            Grants additional roles to a user.
  revokeRolesFromUser         Removes a one or more roles from a user on the current database.
  getUser                     Returns user information for a specified user. Run this method on the user's database. The user must exist on the database on which the method
                              runs.
```

```
  rotateCertificates                Calls the rotateCertificates command
  printCollectionStats              Prints the collection.stats for each collection in the db.
  getFreeMonitoringStatus           Calls the getFreeMonitoringStatus command
  disableFreeMonitoring             returns the db disableFreeMonitoring. uses the setFreeMonitoring command
  enableFreeMonitoring              returns the db enableFreeMonitoring. uses the setFreeMonitoring command
  getProfilingStatus                returns the db getProfilingStatus. uses the profile command
  setProfilingLevel                 returns the db setProfilingLevel. uses the profile command
  setLogLevel                       returns the db setLogLevel. uses the setParameter command
  getLogComponents                  returns the db getLogComponents. uses the getParameter command
  cloneDatabase                     deprecated, non-functional
  cloneCollection                   deprecated, non-functional
  copyDatabase                      deprecated, non-functional
  commandHelp                       returns the db commandHelp. uses the passed in command with help: true
  listCommands                      Calls the listCommands command
  getLastErrorObj                   Calls the getLastError command
  getLastError                      Calls the getLastError command
  printShardingStatus               Calls sh.status(verbose)
  printSecondaryReplicationInfo     Prints secondary replicaset information
  getReplicationInfo                Returns replication information
  printReplicationInfo              Formats sh.getReplicationInfo
  printSlaveReplicationInfo         DEPRECATED. Use db.printSecondaryReplicationInfo
  setSecondaryOk                    This method is deprecated. Use db.getMongo().setReadPref() instead
  watch                             Opens a change stream cursor on the database
  sql                               (Experimental) Runs a SQL query against Atlas Data Lake. Note: this is an experimental feature that may be subject to change in future
                                    releases.

Example>
```

ii) Get the help on collection related commands.

```
> db.Ashish.help()
< Collection Class

  aggregate                       Calculates aggregate values for the data in a collection or a view.
  bulkWrite                       Performs multiple write operations with controls for order of execution.
  count                           Returns the count of documents that would match a find() query for the collection or view.
  countDocuments                  Returns the count of documents that match the query for a collection or view.
  deleteMany                      Removes all documents that match the filter from a collection.
  deleteOne                       Removes a single document from a collection.
  distinct                        Finds the distinct values for a specified field across a single collection or view and returns the results in an array.
  estimatedDocumentCount          Returns the count of all documents in a collection or view.
  find                            Selects documents in a collection or view.
  findAndModify                   Modifies and returns a single document.
  findOne                         Selects documents in a collection or view.
  renameCollection                Renames a collection.
  findOneAndDelete                Deletes a single document based on the filter and sort criteria, returning the deleted document.
  findOneAndReplace               Modifies and replaces a single document based on the filter and sort criteria.
  findOneAndUpdate                Updates a single document based on the filter and sort criteria.
  insert                          Inserts a document or documents into a collection.
  insertMany                      Inserts multiple documents into a collection.
  insertOne                       Inserts a document into a collection.
  isCapped                        Checks if a collection is capped
  remove                          Removes documents from a collection.
  replaceOne                      Replaces a single document within the collection based on the filter.
```

```
  totalIndexSize                  Reports the total size used by the indexes on a collection.
  reIndex                         Rebuilds all existing indexes on a collection.
  getDB                           Get current database.
  getMongo                        Returns the Mongo object.
  dataSize                        This method provides a wrapper around the size output of the collStats (i.e. db.collection.stats()) command.
  storageSize                     The total amount of storage allocated to this collection for document storage.
  totalSize                       The total size in bytes of the data in the collection plus the size of every index on the collection.
  drop                            Removes a collection or view from the database.
  exists                          Returns collection infos if the collection exists or null otherwise.
  getFullName                     Returns the name of the collection prefixed with the database name.
  getName                         Returns the name of the collection.
  runCommand                      Runs a db command with the given name where the first param is the collection name.
  explain                         Returns information on the query plan.
  stats                           Returns statistics about the collection.
  latencyStats                    returns the $latencyStats aggregation for the collection. Takes an options document with an optional boolean 'histograms' field.
  initializeOrderedBulkOp         Initializes an ordered bulk command. Returns an instance of Bulk
  initializeUnorderedBulkOp       Initializes an unordered bulk command. Returns an instance of Bulk
  getPlanCache                    Returns an interface to access the query plan cache for a collection. The interface provides methods to view and clear the query plan cache.
  mapReduce                       Calls the mapReduce command
  validate                        Calls the validate command. Default full value is false
  getShardVersion                 Calls the getShardVersion command
  getShardDistribution            Prints the data distribution statistics for a sharded collection.
  watch                           Opens a change stream cursor on the collection
  hideIndex                       Hides an existing index from the query planner.
  unhideIndex                     Unhides an existing index from the query planner.

Example >
```

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



## Practical 6 - Handling Collections in MongoDB

### a) MongoDB Collections

### 1. Explain the following types of Collec

1. Implicit Collection

In this method, we do not specify the create collection command. The collection is created automatically when data is imported from the application. This is the most common method used for creating collection in MongoDB. We can insert the data from the mongo shell or directly from the application.

2. Explicit Collection

The explicit method to create collection is mainly used in case there is need to create special collections like capped collections or collections with document validation rules.

3. Capped Collection

Capped collections are fixed size collections that start to overwrite the oldest log entries when the size reaches the maximum limit. The oplog that is used for MongoDB replication are capped collections.

4. Capped Collection with Document Validation

Documents with validation in MongoDB compare each insert and updates against the validation rules set for the collection. These rules are specified using validator option in the createCollection command. The row is inserted or updated if the validation is passed and rejected if the validation fails.

5. Clustered Collection

Clustered collections **store documents ordered by the clustered index key value**. You can only have one clustered index in a collection because the documents can be stored in only one order. Only collections with a clustered index store the data in sorted order.

**b) Implicit Creation of Collection**

1. Create an implicit collection named ‘MyCollection’ in a database named ‘MyDB’ by inserting the following document in to the collection ‘MyCollection’.

```
> use MyDB
< 'switched to db MyDB'
> db.MyCollection.insertOne({name:'Ashish'});
< { acknowledged: true,
    insertedId: ObjectId("632c8d22f54d089156d50ea2") }
MyDB>
```

**c) Explicit Creation of Collection**

1. Create an explicit collection named ‘MyCollection’ in a database named ‘MyDB’ by inserting the following document in to the collection ‘MyCollection’.

```
> db.createCollection('MyCollection')
< { ok: 1 }
MyDB>
```

```
> db.MyCollection.insertOne({name:'Ashish'})
< { acknowledged: true,
    insertedId: ObjectId("632c9936f54d089156d50ea3") }
MyDB>
```

**d) Capped Collection**

1. Create a capped collection named “MyLogCollection’ of size 524880. Also specify the maximum number of documents as 100.

db.createCollection(“MyLogCollection”,{capped:true,size:524880,max:100})

```
> db.createCollection('MyLogCollection',{capped:true,size:524880,max:100});
< { ok: 1 }
MyDB >
```

```
> show collections
< MyCollection
  MyLogCollection
MyDB >
```

2. Create a time series collection that captures weather data for the past 24 hours.

https://www.mongodb.com/docs/manual/core/timeseries/timeseries-procedures/#std-label-timeseries-create-query-procedures

```
> db.createCollection(
      "weather",
      {
         timeseries: {
            timeField: "timestamp",
            metaField: "metadata",
            granularity: "hours"
         }
      }
  );
< { ok: 1 }
```

**e) Creating Collection with Document Validation**

https://www.mongodb.com/docs/manual/core/schema-validation/specify-json-schema/

1. Create a collection named ‘validate’ with validation rules and try to insert data that does not meet the validation requirement.

*Given Validation Rules/ @jsonSchema:*

- BSON Type as Object.
- Phone’ field as required string field.

- Email' filed as a string filed with @ and . characters.
- Status' field with one of the two possible values 'Unknown' or 'InComplete'.

```
> db.createCollection("validate",{validator:{$jsonSchema:{
    bsonType:"object",required:["phone","email","status"],properties:{
      phone:{bsonType:"string",description:"phone is not required"},
      email:{bsonType:"string",description:"email required"},
      status:{bsonType:"string",enum:["unknown","incomplete"],
      description:"Must be either unknown or incomplete"}
    }}}})
< { ok: 1 }
> db.validate.insertOne({phone:"abc",email:"ashish",status:"unknown"})
< { acknowledged: true,
    insertedId: ObjectId("633beb1fc073f33dd5d30d0c") }
> db.validate.insertOne({phone:"abc",email:"ashishranarjcollege@gmail.com",status:"unknown"})
< { acknowledged: true,
    insertedId: ObjectId("633beb2bc073f33dd5d30d0d") }
> db.validate.insertOne({phone:"abc",email:"pqr",status:"xyz"})
▸ MongoServerError: Document failed validation
```

## f) Clustered Collection

https://www.mongodb.com/docs/v5.3/core/clustered-collections/

**1.** Create a clustered collection named 'stocks' with the following features.

i. Cluster index key as 1.

ii. Unique clustered index key value.

iii. Clustered index name as 'stockCluserKey'.

```
> db.runCommand( {
    create: "products",
    clusteredIndex: { "key": { _id: 1 }, "unique": true, "name": "products clustered key" }
  } )
< { ok: 1 }
```

## Practical 7 - Insert/import Operations in MongoDB

### a) MongoDB Data Insertion Methods.

1. Explain the following MongoDB Methods with the syntax:

1. Insert()

In MongoDB, the insert() method **inserts a document or documents into the collection**. It takes two parameters, the first parameter is the document or array of the document that we want to insert and the remaining are optional. Using this method you can also create a collection by inserting documents.

2. insertOne()

In MongoDB, insertOne() method inserts a document into the collection. This method inserts only one document at a time. Using this method you can also create a collection by inserting documents. You can insert documents with or without _id field.

3. insertMany()

The insertMany() method inserts one or more documents in the collection. It takes array of documents to insert in the collection. By default, documents are inserted in the given order if you want to insert documents in unordered, then set the value of ordered to false.

4. Compare insert() and insertOne() Method

In MongoDB, the insert() method **inserts a document or documents into the collection**. It takes two parameters, the first parameter is the document or array of the document that we want to insert and the remaining are optional. Using this method you can also create a collection by inserting documents.

In MongoDB, insertOne() method inserts a document into the collection. This method inserts only one document at a time. Using this method you can also create a collection by inserting documents. You can insert documents with or without _id field.

**b) MongoDB Data Types**

Integer, Boolean, Double, String, Arrays, Object, Null, Date and TimeStamp

1. Explain the following data types of MongoDB.

Integer, Boolean, Double, String, Arrays, Object, Null, Date and TimeStamp

**Integer**: In MongoDB, the integer data type is used to store an integer value. We can store integer data type in two forms **32 -bit signed integer and 64 – bit signed integer**.

**Boolean –** This type is used to store a boolean (true/ false) value.

**Double –** This type is used to store floating point values.

**String:** This is the most commonly used data type in MongoDB to store data, BSON strings are of UTF-8. So, the drivers for each programming language convert from the string format of the language to UTF-8 while serializing and de-serializing BSON. The string must be a valid UTF-8.

**Array:** The Array is the set of values. It can store the same or different data types values in it. In MongoDB, the array is created using square brackets([]).

**Object:** Object data type stores embedded documents. Embedded documents are also known as nested documents. Embedded documents or nested documents are those types of documents which contain a document inside another document.

**Null:** The null data type is used to store the null value.

**Date –** This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating an object of Date and passing day, month, year into it.

**Timestamp –** timestamp. This can be handy for recording when a document has been modified or added.

**c) insert() Method**

1. Insert the following information into the ‘Student’ Collection of ‘College’ database using insert() method.

Name:”Amit”, Class:”TY BSc IT”, College:”RJC” and PerMarks:85.00%

```
> use college
< 'switched to db college'
> db.createCollection('student')
< { ok: 1 }
```

```
> db.student.insert({name:'Amit',college:'RJC',Percentage:'85%'});
< 'DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.'
< { acknowledged: true,
    insertedIds: { '0': ObjectId("632c9f68f54d089156d50ea5") } }
college>
```

**d) insertOne() Method**

1. Insert the following information into the ‘Student’ Collection of ‘College’ database using insertOne() method.

Name:”Pranali”, Class:”TY BSc IT”, College:”RJC” and PerMarks:90.00%

```
> db.student.insertOne({name:'Amit',college:'RJC',class:'TYBSC IT',Percentage:'85%'});
< { acknowledged: true,
    insertedId: ObjectId("632ca031f54d089156d50ea6") }
college>
```

**e) insertMany() Method**

1. Insert the following information into the ‘Student’ Collection of ‘College’ database using insertMany() method.

Name:”Lalita”, Class:”TY BSc IT”, College:”KC” and PerMarks:89.00%

Name:”Sonali”, Class:”TY BSc IT”, College:”RJC” and PerMarks:58.00%

Name:”Suresh”, Class:”TY BSc IT”, College:”MCC” and PerMarks:75.00%

```
> db.student.insertMany([{name:'Lalita',college:'KC',class:'TYBSC IT',Percentage:'89%'},
  {name:'Sonali', college:'RJC', class:'TYBSC IT',Percentage:"58%"},
  {nmae:'Suresh', college:'MCC',class:'TYBSC IT', Percentage:'75%'}]
  );
< { acknowledged: true,
    insertedIds:
     { '0': ObjectId("632ca2e1f54d089156d50ea7"),
       '1': ObjectId("632ca2e1f54d089156d50ea8"),
       '2': ObjectId("632ca2e1f54d089156d50ea9") } }
college>
```

## f) Embedded Documents

1. Insert the following information into the 'Student' Collection of 'College' database with the embedded document for the field of address.

Name:"Prakash", Class:"TY BSc IT", College:"MCC", Address:{BulidNo:1,Street:"JN Road", City"Mumbai"} and PerMarks:75.00%

```
> db.student.insertOne({name:'Prakash',college:'MCC', class:'TYBSC IT',
  Address:{BuildNo:1,Street:'JN Road', city:'Mumbai'},
  percentage:'75%'});
< { acknowledged: true,
    insertedId: ObjectId("632ca474f54d089156d50eaa") }
college>
```

## g) Importing data from .csv file

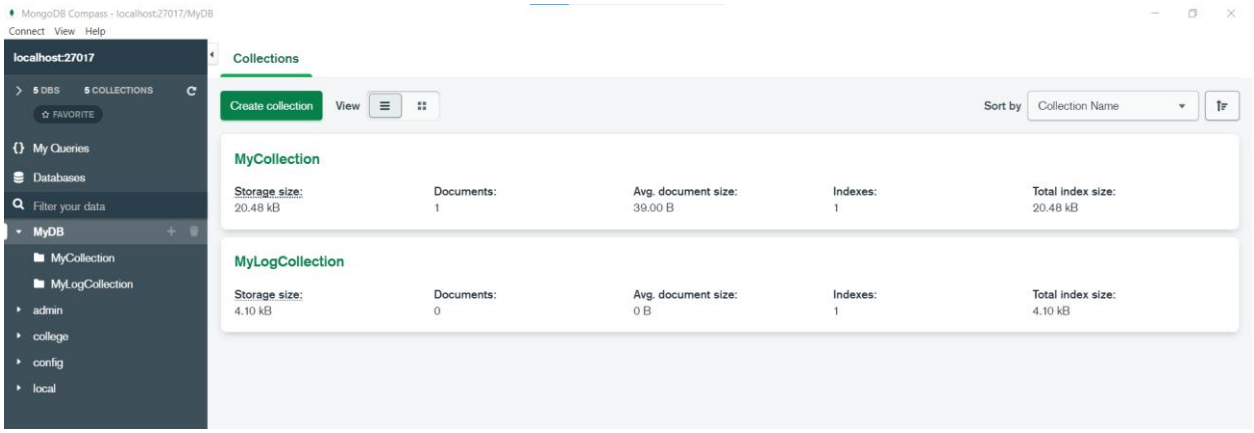
1. Import the data of datacsv.csv file into MongoDB Collection named 'Attendance'.

Given, datacsv.csv file

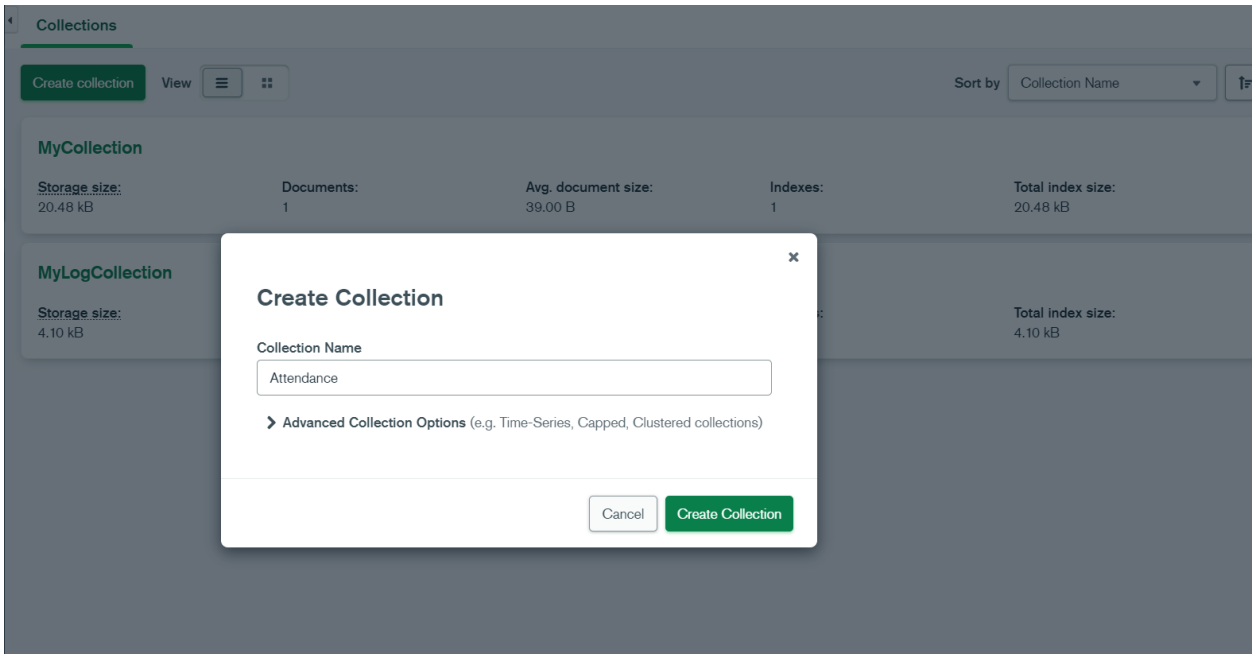
| Timestamp | Email Address | Roll No | First Name | Last Name |
|---|---|---|---|---|
| 6/14/2022 9:17:18 | umeshupadhay@gail.com | 5 | Umesh | upadhyay |
| 6/14/2022 9:18:35 | jeetyaddav@gmail.com | 21 | Jeet | Yadav |

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



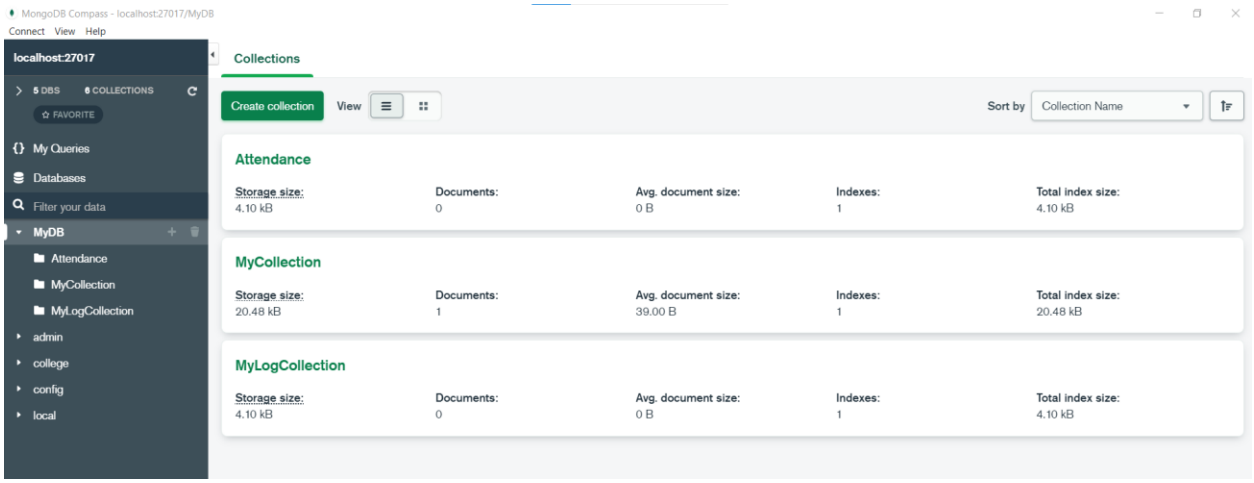



Here I will create a collection name as ‘Attendance’.

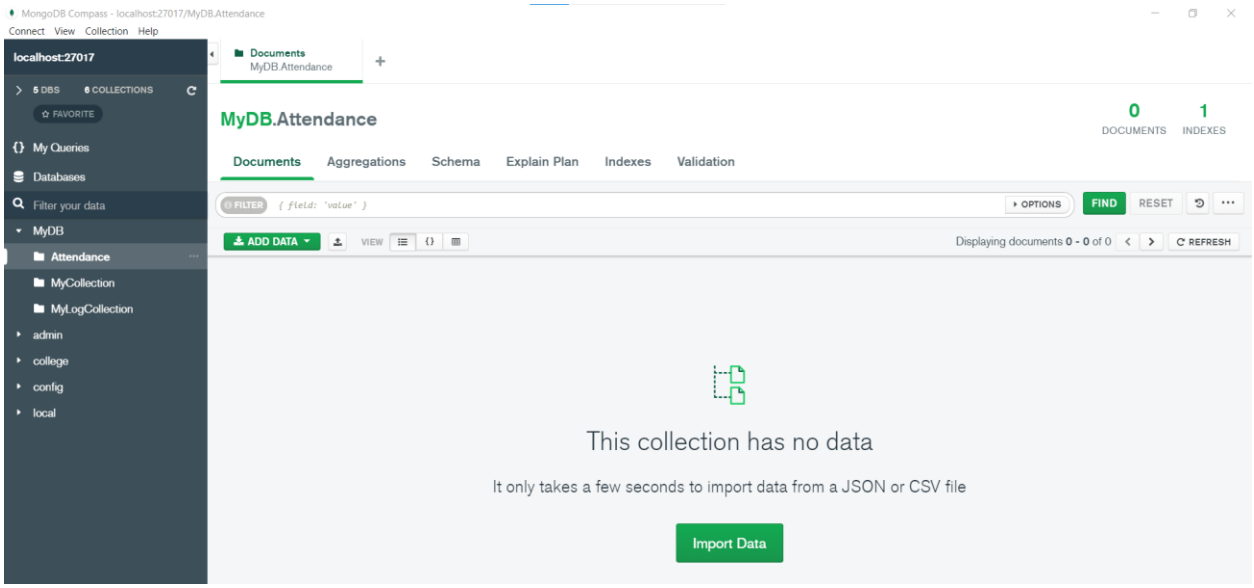


Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



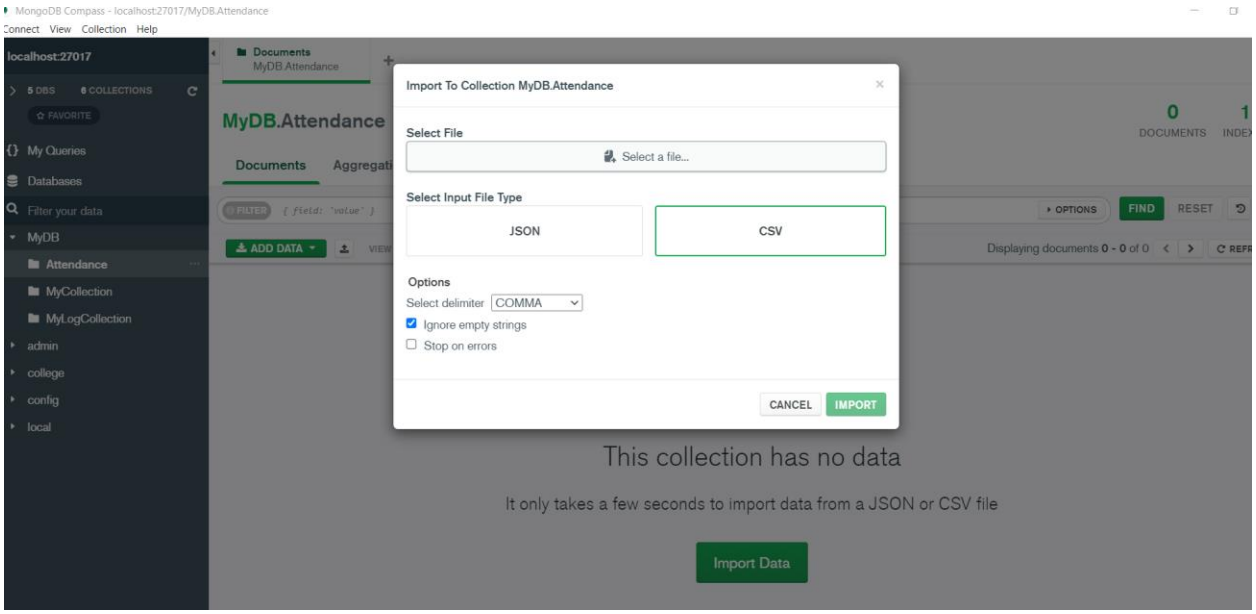



After creating the Collection, import the data

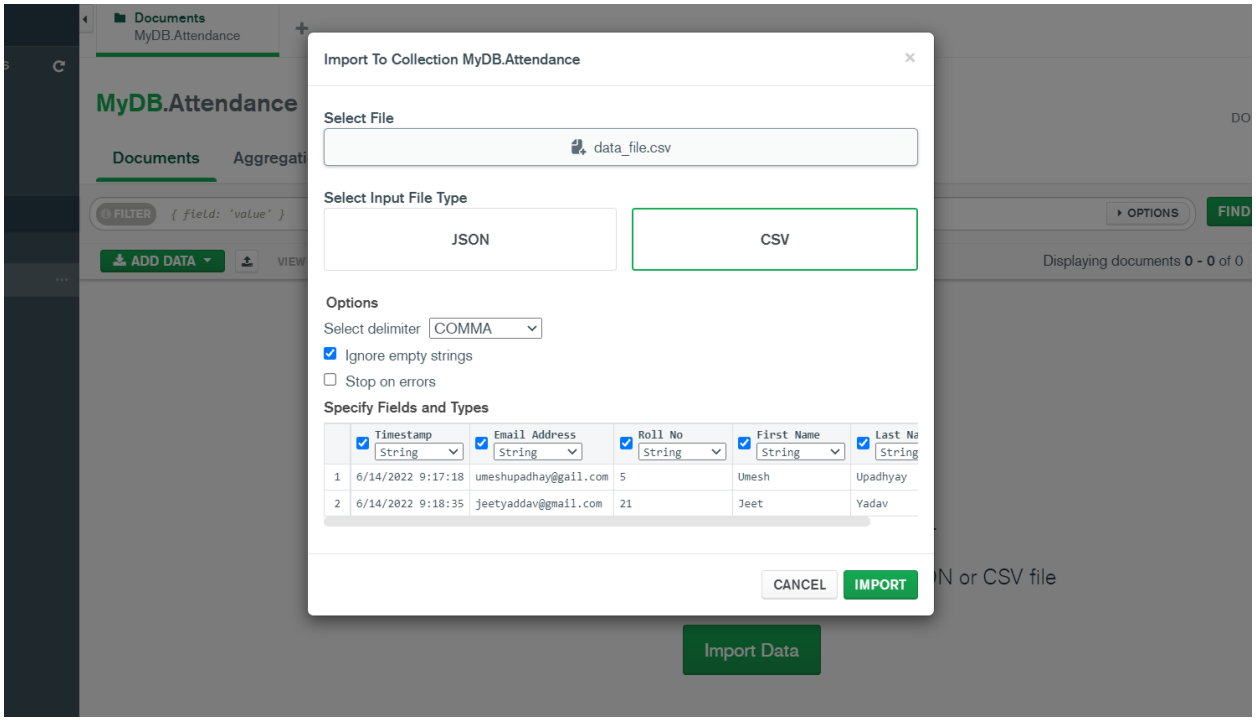


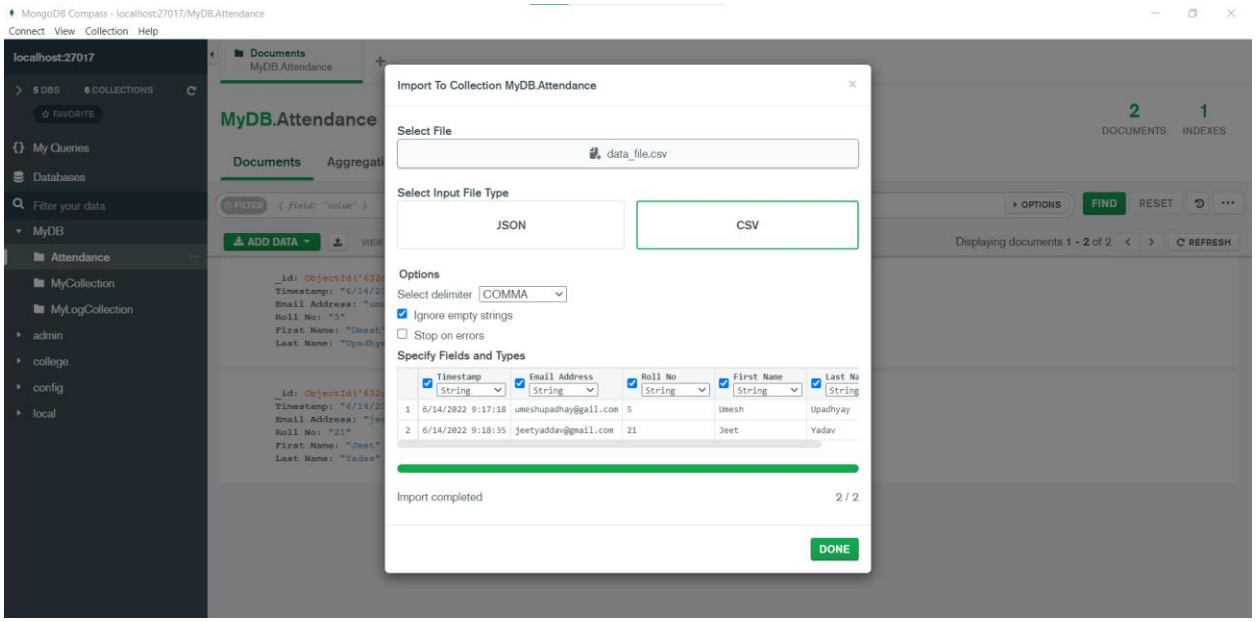
Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



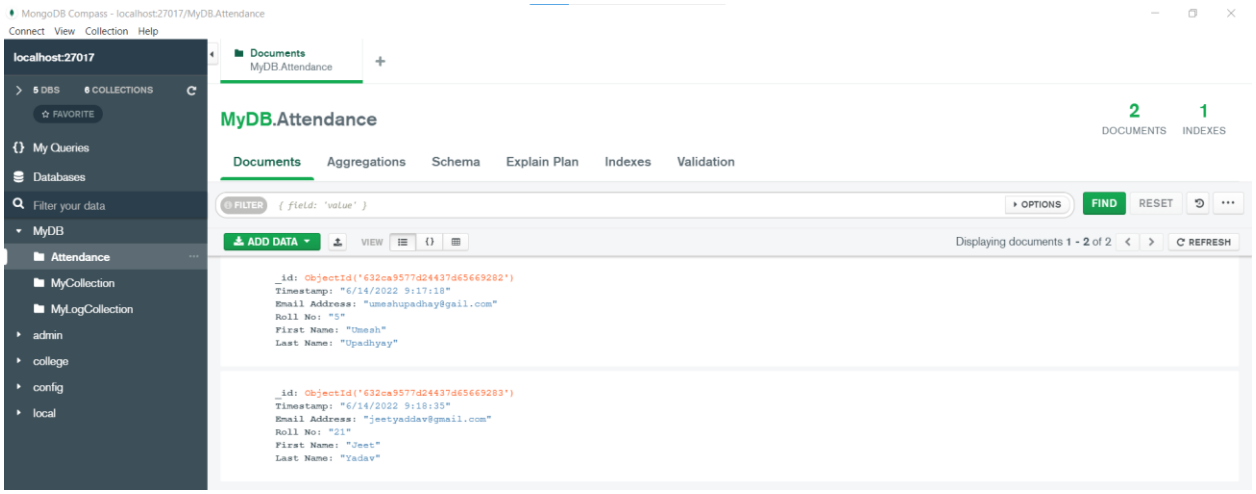



Here I want to import a CSV file so I will choose the option CSV, and then select the file.






Here the data is imported successfully.




## h) Importing data from .json file

1. Import the data from datajson.json file into the MongoDB Collection named ‘StudInfo’.

Given, datajson.json file.

```
[
{
```

"Name" : "Aditya",

"Roll No" : 3030,

"PLace" : "Matunga"

},

{

"Name" : "Suman",

"Roll No" : 3012,

"PLace" : "Kurla"

},

{

"Name" : "Rohan",

"Roll No" : 3013,

"PLace" : "Dombivali"

},

{

"Name" : "Shubham",

"Roll No" : 3002,

"PLace" : "Ghatkoper"

},

{

"Name" : "Durgesh",

"Roll No" : 3034,

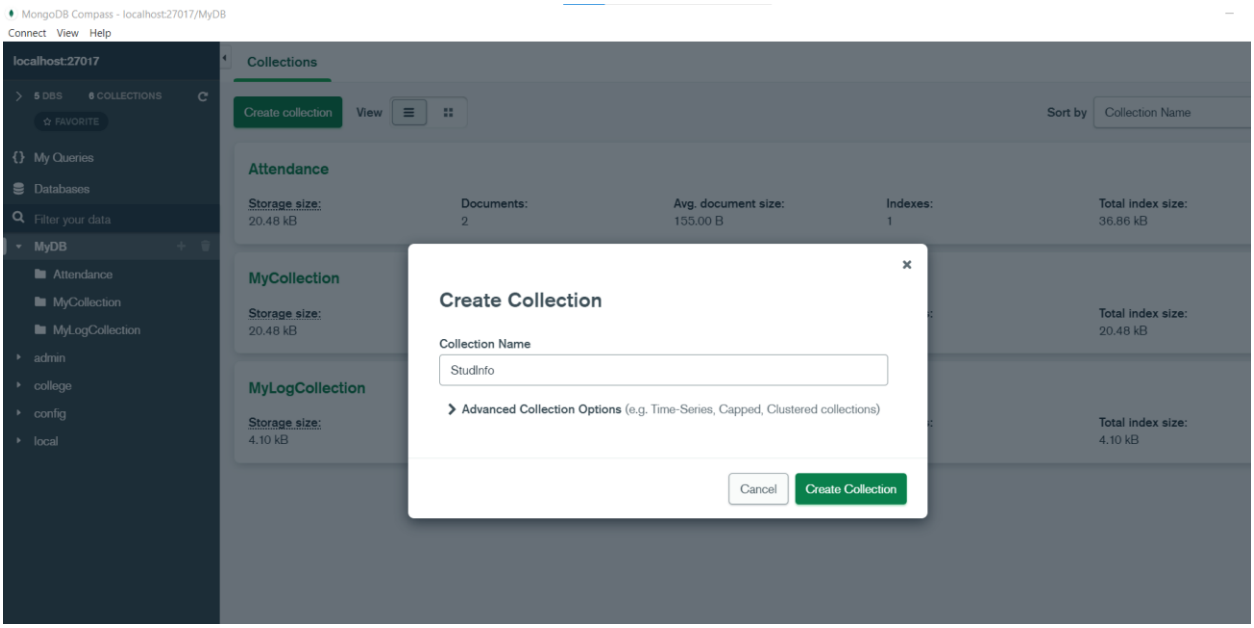
"PLace" : "Asalpha"

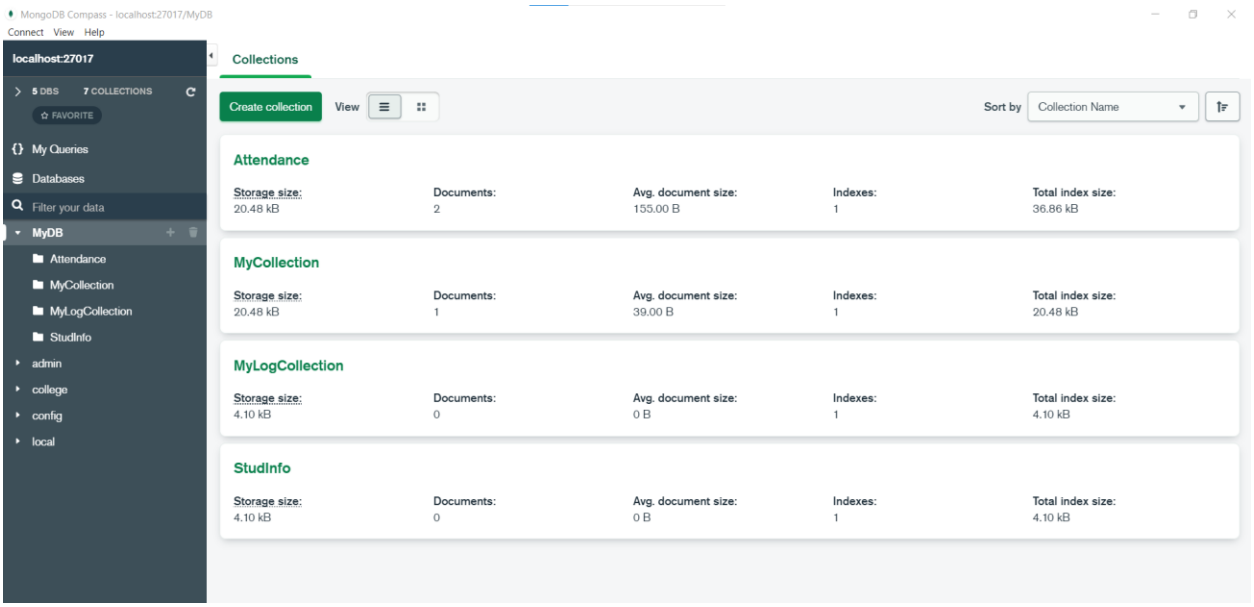
}

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



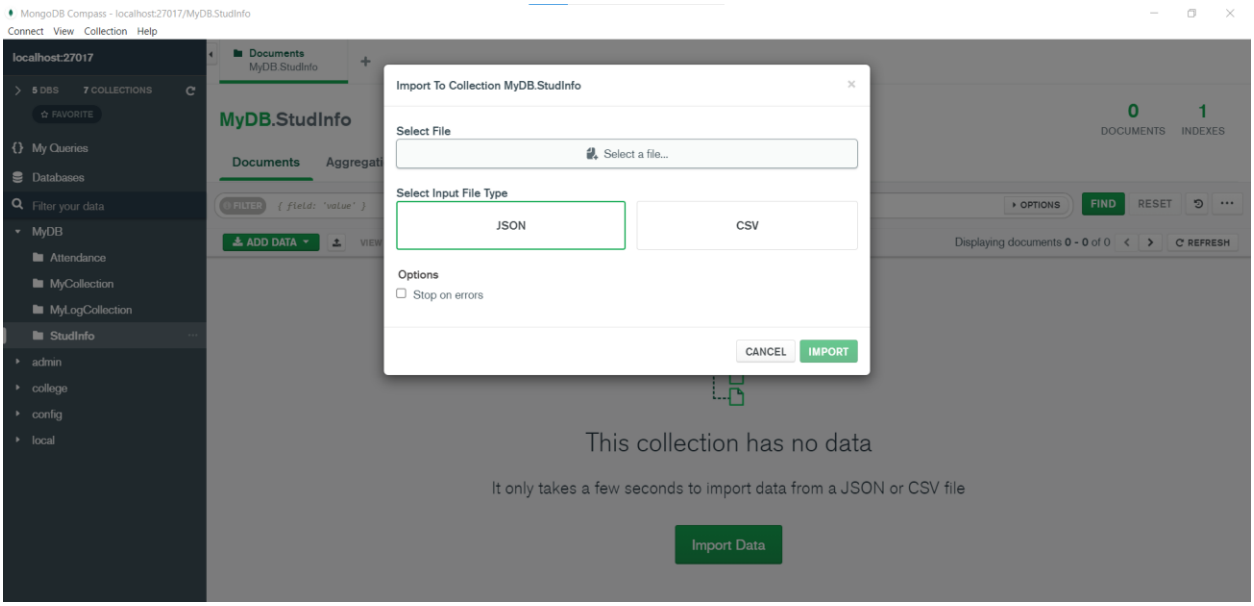
]

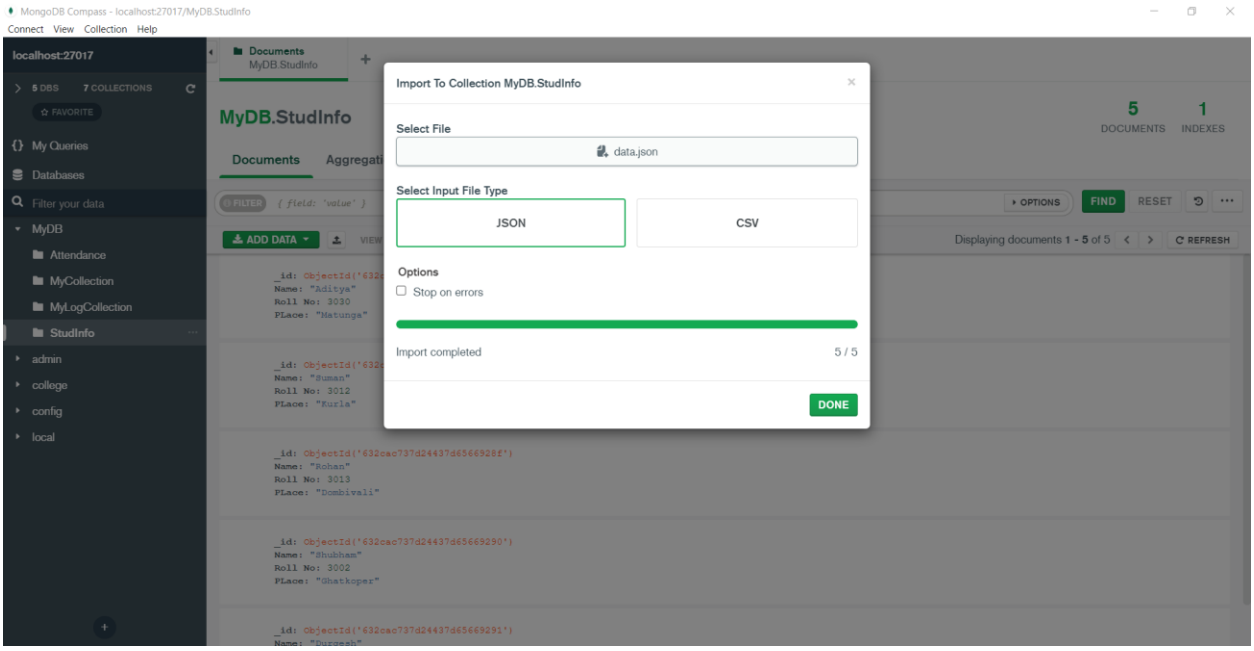


Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics

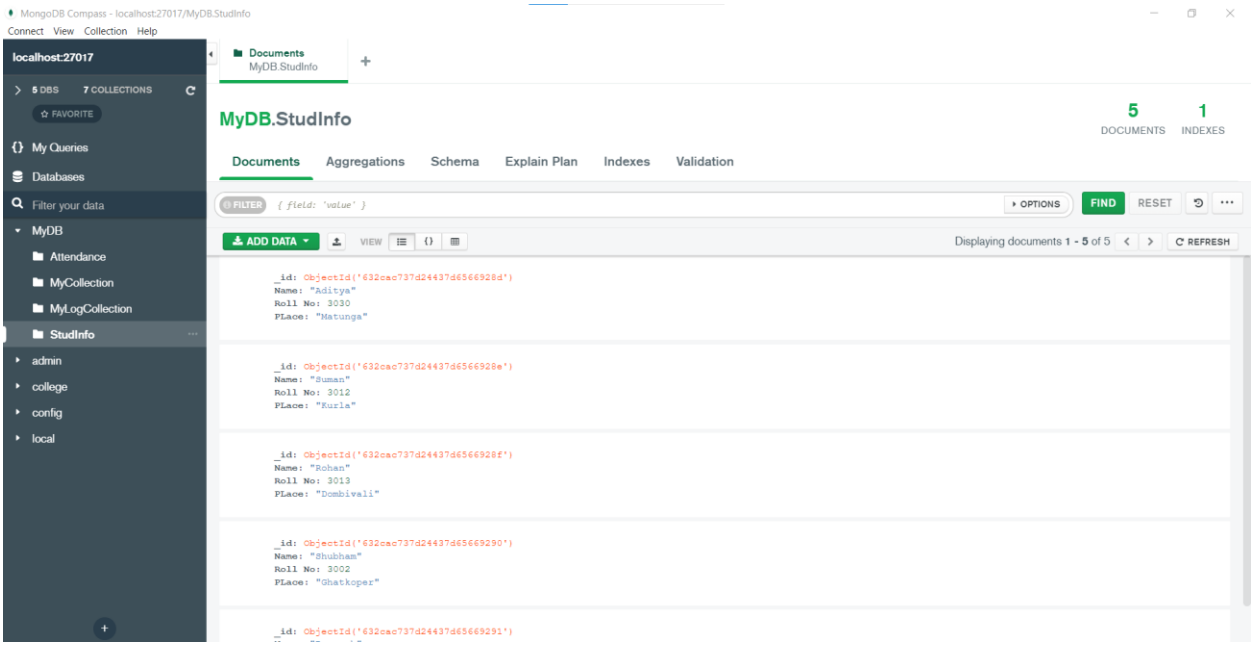




Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics





**Practical 8 - Querying MongoDB Database**

Create the employee collection with the given documents and solve the following queries.

Given, Documents for ‘employee’ collection.

```
db.employees.insertMany([
   {
      _id:1,
      firstName: "John",
      lastName: "King",
      email: "john.king@abc.com",
      salary: 5000
   },
   {
      _id:2,
      firstName: "Sachin",
      lastName: "T",
      email: "sachin.t@abc.com",
      salary: 8000
   },
   {
      _id:3,
      firstName: "James",
      lastName: "Bond",
      email: "jamesb@abc.com",
      salary: 7500
```

```
    },
    {
        _id:4,
        firstName: "Steve",
        lastName: "J",
        email: "steve.j@abc.com",
        salary: 9000
    },
    {
        _id:5,
        firstName: "Kapil",
        lastName: "D",
        email: "kapil.d@abc.com",
        salary: 4500
    },
    {
        _id:6,
        firstName: "Amitabh",
        lastName: "B",
        email: "amitabh.b@abc.com",
        salary: 11000
    },
{
        _id:7,
```

firstName: "John",
lastName: "King",
email: "john.king@abc.com",
salary: 5000,
skills: [ "Angular", "React", "MongoDB" ],
department: {
"name":"IT"
}
},
{
_id:8,
firstName: "Sachin",
lastName: "T",
email: "sachin.t@abc.com",
salary: 8000,
skills: [ "Accounting", "Tax" ],
department: {
"name":"Finance"
}
},
{
_id:9,
firstName: "James",
lastName: "Bond",

```
    email: "jamesb@abc.com",
    salary: 7500,
    skills: [ "Sales", "Marketing" ],
    department: {
        "name":"Marketing"
    }
},
{
    _id:10,
    firstName: "Steve",
    lastName: "J",
    email: "steve.j@abc.com",
    salary: 7000
},
{
    _id:11,
    firstName: "Kapil",
    lastName: "D",
    email: "kapil.d@abc.com",
    salary: 4500,
    skills: [ "Accounting", "Tax" ],
    department: {
        "name":"Finance"
    }
```

```
  },
  {
    _id:12,
    firstName: "Amitabh",
    lastName: "B",
    email: "amitabh.b@abc.com",
    salary: 7000
  }
])
```

```
> db.employees.insertMany([
      {
          _id:1,
          firstName: "John",
          lastName: "King",
          email: "john.king@abc.com",
          salary: 5000
      },
      {
          _id:2,
          firstName: "Sachin",
          lastName: "T",
          email: "sachin.t@abc.com",
          salary: 8000
      },
      {
          _id:3,
          firstName: "James",
          lastName: "Bond",
          email: "jamesb@abc.com",
          salary: 7500
      },
```

```
        firstName: "Steve",
        lastName: "J",
        email: "steve.j@abc.com",
        salary: 7000
    },
    {
        _id:11,
        firstName: "Kapil",
        lastName: "D",
        email: "kapil.d@abc.com",
        salary: 4500,
        skills: [ "Accounting", "Tax" ],
        department: {
                    "name":"Finance"
                }
    },
    {
        _id:12,
        firstName: "Amitabh",
        lastName: "B",
        email: "amitabh.b@abc.com",
        salary: 7000
    }
]);
```

```
{ acknowledged: true,
  insertedIds:
   { '0': 1,
     '1': 2,
     '2': 3,
     '3': 4,
     '4': 5,
     '5': 6,
     '6': 7,
     '7': 8,
     '8': 9,
     '9': 10,
     '10': 11,
     '11': 12 } }
MyDB>
```

a) find() and fineOne() Method

1. Display all documents of the ‘employee’ collection using find() method.

```
> db.employees.find();
{ _id: 1,
  firstName: 'John',
  lastName: 'King',
  email: 'john.king@abc.com',
  salary: 5000 }
{ _id: 2,
  firstName: 'Sachin',
  lastName: 'T',
  email: 'sachin.t@abc.com',
  salary: 8000 }
```

```
  { _id: 12,
    firstName: 'Amitabh',
    lastName: 'B',
    email: 'amitabh.b@abc.com',
    salary: 7000 }
MyDB >
```

2. Display the first document of the ‘employee’ collection using findOne() method.

```
> db.employees.findOne();
< { _id: 1,
    firstName: 'John',
    lastName: 'King',
    email: 'john.king@abc.com',
    salary: 5000 }
MyDB >
```

b) pretty() Method

3. Display all documents of the ‘employee’ collection in proper format.

c) Filtering criteria in MongoDB Queries / Selection Queries.

4. Display the first occurrence of the document where the firstName is ‘Kapil’.

```
> db.employees.find({firstName:'Kapil'});
< { _id: 5,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500 }
  { _id: 11,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
MyDB >
```

5. Get the mail id of an employee having a first name as ‘John’.

```
> db.employees.find({firstName:'John'},{email:1});
< { _id: 1, email: 'john.king@abc.com' }
  { _id: 7, email: 'john.king@abc.com' }
MyDB >
```

6. Get the list of employees having salaries greater than 7000 in ascending order of firstName and lastName.

```
> db.employees.find({salary:{$gt:7000}}).sort({
< { _id: 6,
    firstName: 'Amitabh',
    lastName: 'B',
    email: 'amitabh.b@abc.com',
    salary: 11000 }
  { _id: 3,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500 }
  { _id: 8,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
  { _id: 4,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 9000 }
MyDB >
```

```
> db.employees.find({salary:{$gt:7000}}).sort({lastName:1});
< { _id: 6,
    firstName: 'Amitabh',
    lastName: 'B',
    email: 'amitabh.b@abc.com',
    salary: 11000 }
  { _id: 3,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500 }
  { _id: 9,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500,
    skills: [ 'Sales', 'Marketing' ],
    department: { name: 'Marketing' } }
  { _id: 4,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 9000 }
```

```
{ _id: 2,
  firstName: 'Sachin',
  lastName: 'T',
  email: 'sachin.t@abc.com',
  salary: 8000 }
{ _id: 8,
  firstName: 'Sachin',
  lastName: 'T',
  email: 'sachin.t@abc.com',
  salary: 8000,
  skills: [ 'Accounting', 'Tax' ],
  department: { name: 'Finance' } }
MyDB>
```

7. Get the list of employees having salaries greater than 7000 in ascending order of firstName and lastName. Also exclude _id field from the result.

   db.employees.find({salaries: {$gt:7000} },{_id:0});

```
{ firstName: 'James',
  lastName: 'Bond',
  email: 'jamesb@abc.com',
  salary: 7500,
  skills: [ 'Sales', 'Marketing' ],
  department: { name: 'Marketing' } }
MyDB>
```

d) Using operators in queries.

8. Get an employee having a salary greater than 7000.($gt operator)

```
> db.employees.find({salary: {$gt:7000} });
< { _id: 2,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000 }
  { _id: 3,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500 }
  { _id: 4,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 9000 }
  { _id: 6,
    firstName: 'Amitabh',
    lastName: 'B',
    email: 'amitabh.b@abc.com',
    salary: 11000 }
```

```
  { _id: 9,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500,
    skills: [ 'Sales', 'Marketing' ],
    department: { name: 'Marketing' } }
MyDB >
```

9. Get the name of all employees having a salary greater than 7000 and less than 10000. (Multiple query operator)

```
> db.employees.find({salary:{$gt:7000,$lt:10000}});
< { _id: 2,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000 }
  { _id: 3,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500 }
```

```
  { _id: 9,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500,
    skills: [ 'Sales', 'Marketing' ],
    department: { name: 'Marketing' } }
MyDB >
```

10. Get the list of all employees from the ‘Finance’ Department. (Querying embedded documents)

```
> db.employees.find({department:{name:'Finance'}});
< { _id: 8,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
  { _id: 11,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
MyDB >
```

11. Get the list of employees having a salary 7000 or 9000 or 11000. ($in operator)

```
> db.employees.find({salary:{$in:[7000,9000,10000]}});
< { _id: 4,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 9000 }
  { _id: 10,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 7000 }
  { _id: 12,
    firstName: 'Amitabh',
    lastName: 'B',
    email: 'amitabh.b@abc.com',
    salary: 7000 }
MyDB >
```

12. List all the employees with the ‘Tax’ skill. (Querying array element)

```
> db.employees.find({skills:{$all:['Tax']}});
< { _id: 8,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
  { _id: 11,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
MyDB >
```

13. List all the employees with the ‘Tax’ skill or ‘Sales’ skill. ($in operator)

```
> db.employees.find({skills:{$in:['Tax','Sales']}});
< { _id: 8,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
  { _id: 9,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500,
    skills: [ 'Sales', 'Marketing' ],
    department: { name: 'Marketing' } }
  { _id: 11,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
MyDB >
```

14. List all the employees with the ‘Tax’ and ‘Accounting’ skills. ($all operator)

```
> db.employees.find({skills:{$all:['Tax','Accounting']}});
< { _id: 8,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
  { _id: 11,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
MyDB >
```

15. List all the employees with the 3 skills. ($size operator)

```
> db.employees.find({skills:{$size:3}});
< { _id: 7,
    firstName: 'John',
    lastName: 'King',
    email: 'john.king@abc.com',
    salary: 5000,
    skills: [ 'Angular', 'React', 'MongoDB' ],
    department: { name: 'IT' } }
MyDB >
```

e) Projection Queries

16. List the name of an employee who has a first name as ‘Sachin’. (Selection & Projection).

```
> db.employees.find({firstName:'Sachin'});
< { _id: 2,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000 }
  { _id: 8,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
MyDB >
```

f) limit() Method

17. Get the first five employee documents.

```
> db.employees.find().limit(5);
< { _id: 1,
    firstName: 'Morgan',
    lastName: 'King',
    email: 'john.king@abc.com',
    salary: 5500 }
  { _id: 2,
    firstName: 'Sachin',
    lastName: 'T',
    email: 'sachin.t@abc.com',
    salary: 8000 }
  { _id: 3,
    firstName: 'James',
    lastName: 'Bond',
    email: 'jamesb@abc.com',
    salary: 7500 }
  { _id: 4,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 9000 }
  { _id: 5,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500 }
MyDB >
```

18. Get the first five employee documents with their first name and last name.

```
> db.employees.find({},{firstName:1,lastName:1}).limit(5);
< { _id: 1, firstName: 'Morgan', lastName: 'King' }
  { _id: 2, firstName: 'Sachin', lastName: 'T' }
  { _id: 3, firstName: 'James', lastName: 'Bond' }
  { _id: 4, firstName: 'Steve', lastName: 'J' }
  { _id: 5, firstName: 'Kapil', lastName: 'D' }
MyDB >
```

**g)** skip() Method

19. Get the five employee documents with their first name and last name except first and second employee.

```
> db.employees.find({},{firstName:1,lastName:1}).limit(5).skip(1,2);
< { _id: 2, firstName: 'Sachin', lastName: 'T' }
  { _id: 3, firstName: 'James', lastName: 'Bond' }
  { _id: 4, firstName: 'Steve', lastName: 'J' }
  { _id: 5, firstName: 'Kapil', lastName: 'D' }
  { _id: 6, firstName: 'Amitabh', lastName: 'B' }
MyDB >
```

**h)** Regular Expressions in MongoDB

20. List the employees having mail id of j@abc.com

```
> db.employees.find({email:{$regex:/j@abc.com/}});
< { _id: 4,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 9000 }
  { _id: 10,
    firstName: 'Steve',
    lastName: 'J',
    email: 'steve.j@abc.com',
    salary: 7000 }
MyDB >
```

## Practical 9 - Updating and Deleting Operations in MongoDB

Using the ‘employee’ collection that is created in Practical 3, solve the following queries.

a) updateOne and updateMany() Methods

1. Change the firstName of the first employee to ‘Morgan’.($set operator)

```
> db.employees.updateOne(
    {_id: 1},
    {$set:
      {firstName: 'Morgan'


      }
    }
    )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
MyDB >
```

```
> db.employees.find().limit(1)
< { _id: 1,
    firstName: 'Morgan',
    lastName: 'King',
    email: 'john.king@abc.com',
    salary: 5000 }
MyDB >
```

2. Update the location of employees having firstName as ‘Steve’ to ‘USA’.

```
> db.employees.updateOne(
    {_id:1},
    {$inc:
    {
    salary:500
    }
    }
    )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.employees.find({_id:1})
< { _id: 1,
    firstName: 'Morgan',
    lastName: 'King',
    email: 'john.king@abc.com',
    salary: 5500 }
MyDB >
```

3. Increase the salary of the first employee by 500. ($inc operator).

```
> db.employees.updateOne(
    {_id:1},
    {$inc:
    {
    salary:500
    }
    }
    )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.employees.find({_id:1})
< { _id: 1,
    firstName: 'Morgan',
    lastName: 'King',
    email: 'john.king@abc.com',
    salary: 5500 }
MyDB >
```

4. Set the salary of an employee to 80000 having firstName as ‘Aarti’. (Add the document if not exist)

```
> db.employees.updateOne(
    {firstName: 'Aarti'},
    {$set:
    {
      salary:80000
    }
    }
    )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0 }
> db.employees.find({firstName:'Aarti'})
<
MyDB >
```

**Document is not exist.**
**Adding document for firstName Aarti**

```
> db.employees.insertOne({firstName:'Aarti', salary:80000})
< { acknowledged: true,
    insertedId: ObjectId("632f410d38c73ff5e81aa35e") }
MyDB >
```

5. Add the appraisal date to the current date for all employees and increase the salary by 5000 of each employee. ($currentDate operator)

6. List an employee with the highest salary. ($max operator)

```
> db.employees.find().sort({salary:-1}).limit(1)
< { _id: ObjectId("632f410d38c73ff5e81aa35e"),
    firstName: 'Aarti',
    salary: 80000 }
> db.employees.aggregate([{$group:{_id:null,max:{$max:"salary"}}}])
< { _id: null, max: 'salary' }
```

7. List an employee with the lowest salary and ($min operator)

```
> db.employees.find().sort({salary:1}).limit(1)
< { _id: 5,
    firstName: 'Kapil',
    lastName: 'D',
    email: 'kapil.d@abc.com',
    salary: 4500 }
> db.employees.aggregate([{$group:{_id:null,min:{$min:"salary"}}}])
< { _id: null, min: 'salary' }
```

8. Calculate the 20 times the salary of the first employee. ($mul operator)

```
> db.employees.updateOne({_id:1},{$mul:{salary:20}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.employees.find().limit(1)
< { _id: 1,
    firstName: 'Morgan',
    lastName: 'King',
    email: 'john.king@abc.com',
    salary: 110000 }
```

9. Remove the email id field from the collection. ($unset operator)

```
> db.employees.updateMany({},{$unset:{email:""}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 13,
    modifiedCount: 12,
    upsertedCount: 0 }
```

```
> db.employees.find()
< { _id: 1, firstName: 'Morgan', lastName: 'King', salary: 110000 }
  { _id: 2, firstName: 'Sachin', lastName: 'T', salary: 8000 }
  { _id: 3, firstName: 'James', lastName: 'Bond', salary: 7500 }
  { _id: 4, firstName: 'Steve', lastName: 'J', salary: 9000 }
  { _id: 5, firstName: 'Kapil', lastName: 'D', salary: 4500 }
  { _id: 6, firstName: 'Amitabh', lastName: 'B', salary: 11000 }
  { _id: 7,
    firstName: 'John',
    lastName: 'King',
    salary: 5000,
    skills: [ 'Angular', 'React', 'MongoDB' ],
    department: { name: 'IT' } }
  { _id: 10, firstName: 'Steve', lastName: 'J', salary: 7000 }
  { _id: 11,
    firstName: 'Kapil',
    lastName: 'D',
    salary: 4500,
    skills: [ 'Accounting', 'Tax' ],
    department: { name: 'Finance' } }
  { _id: 12, firstName: 'Amitabh', lastName: 'B', salary: 7000 }
  { _id: ObjectId("632f410d38c73ff5e81aa35e"),
    firstName: 'Aarti',
    salary: 80000 }
```

10. Rename the field ‘firstName’ to ‘FirstName’. ($rename operator)

```
> db.employees.updateMany({},{$rename:{firstName:"FirstName"}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 13,
    modifiedCount: 13,
    upsertedCount: 0 }
```

```
> db.employees.find().limit(1)
< { _id: 1, lastName: 'King', salary: 110000, FirstName: 'Morgan' }
```

11. Set the lastName of an employee ‘Aarti’ to ‘Shetty’ and salary to 90000. (updating multiple fields).

```
> db.employees.updateOne({FirstName:"Aarti"},{$set:{lastName:"shetty", salary:90000}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.employees.find({FirstName:"Aarti"})
< { _id: ObjectId("632f410d38c73ff5e81aa35e"),
    salary: 90000,
    FirstName: 'Aarti',
    lastName: 'shetty' }
```

12. Update the skill of ‘James’ to ‘Sales’ and ‘Accounting’ instead of ‘Sales’ and ‘Marketing’. (Updating arrays).

```
> db.employees.find({FirstName:"James"})
< { _id: 3, lastName: 'Bond', salary: 7500, FirstName: 'James' }
  { _id: 9,
    lastName: 'Bond',
    salary: 7500,
    skills: [ 'Sales', 'Marketing' ],
    department: { name: 'Marketing' },
    FirstName: 'James' }
> db.employees.updateOne({_id:9},{$set:{skills:["Sales","Accounting"]}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.employees.find({_id:9})
< { _id: 9,
    lastName: 'Bond',
    salary: 7500,
    skills: [ 'Sales', 'Accounting' ],
    department: { name: 'Marketing' },
    FirstName: 'James' }
```

13. Update the ‘Marketing’ skill to ‘Advertising’ for all documents.

```
> db.employees.updateMany({skills:'marketing'},
  {$set: {"skills.$":'Advertising'}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0 }
MyDB >
```

14. Update the skills element ‘GST’ to ‘Tax’.

```
> db.employees.updateMany({skills:'GST'},
  {$set: {"skills.$":'Tax'}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0 }
MyDB >
```

15. Add ‘sports’ skill to all the existing documents.

```
> db.employees.updateMany({},
  {$push: {skills:'sports'}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 13,
    modifiedCount: 13,
    upsertedCount: 0 }
MyDB >
```

16. Add ‘Acting’ and ‘Reading’ skills to all the existing documents. ($push and $each operators)

```
> db.employees.updateMany({},
  {$push: {skills: {$each: ["Acting","Reading"]}}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 13,
    modifiedCount: 13,
    upsertedCount: 0 }
MyDB >
```

17. Remove the ‘sports’ skill from all the documents.

```
> db.employees.updateMany({},
  {$pull: {skills: "sports"}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 13,
    modifiedCount: 13,
    upsertedCount: 0 }
MyDB >
```

b) save() Method

18. Replace document 1 with a new document using save() method

https://www.mongodb.com/docs/v4.2/reference/method/db.collection.save/

db.products.save( { _id: 1, firstName: "Amit", lastName: "B", email: "amitabh.b@abc.com", salary: 11000 } )

.

c) deleteOne() and deleteMany and remove() Methods

19. Delete an employee having a salary of 7000.

db.employees.deleteMany( { salary: 7000 } )

```
> db.employees.deleteOne({salary:7000})
< { acknowledged: true, deletedCount: 1 }
MyDB >
```

The above command deletes the first matching document even if multiple documents match with the specified criteria.

```
> db.employees.deleteMany({salary:7000})
< { acknowledged: true, deletedCount: 1 }
MyDB >
```

The above command deletes all the documents that match with the specified filter criteria in a collection.

20. Delete all the employees having salaries greater than 70000.

```
> db.employees.deleteMany({salary:{$gt:7000}})
< { acknowledged: true, deletedCount: 8 }
MyDB >
```

21. Remove the employee document having id 2 using remove() method.

```
> db.employees.remove({_id:2})
< 'DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.'
< { acknowledged: true, deletedCount: 0 }
MyDB >
```

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



## Practical 10 - Aggregation Operations in MongoDB

Create the collection named “MyCollection” and insert the following documents and solve the following queries..

Given documents,

db.mycollection.insert([

{ title: 'MongoDB Overview', description: 'MongoDB is no sql database', by: 'tutorials point', url: 'http://www.tutorialspoint.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100 },

{ title: 'NoSQL Database', description: "NoSQL database doesn't have tables", by: 'tutorials point', url: 'http://www.tutorialspoint.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 20, comments: [ { user:'user1', message: 'My first comment', dateCreated: new Date(2013,11,10,2,35), like: 0 } ] } ])

```
{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("633d3659ee41dc3bb9eabfd5"),
     '1': ObjectId("633d3659ee41dc3bb9eabfd6") } }
```

### a) aggregate() Method

1. Write a MongoDB query to use sum, avg, min and max expressions.

   Example:

   a. sum

   db.mycollection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])

```
> db.mycollection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
{ _id: null, num_tutorial: 120 }
```

   b. avg

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

```
> db.mycollection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
< { _id: null, num_tutorial: 60 }
```

c. min

db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])

```
> db.MyCollection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
< { _id: null, num_tutorial: 20 }
```

d. max

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

```
> db.mycollection.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
< { _id: null, num_tutorial: 100 }
```

2. Write a MongoDB query to use push and add ToSet expressions.
   a. Push - Inserts the value to an array in the resulting document.

      db.mycollection.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])

```
> db.mycollection.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
< { _id: null,
    url:
     [ 'http://www.tutorialspoint.com',
       'http://www.tutorialspoint.com' ] }
```

   b. addToSet - Inserts the value to an array in the resulting document but does not create duplicates.

db.mycollection.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])

```
> db.mycollection.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
< { _id: null,
    url:
     [ 'http://www.tutorialspoint.com',
       'http://www.tutorialspoint.com' ] }
```

3. Write a MongoDB query to use the first and last expression.
    a. First

db.mycollection.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])

```
> db.mycollection.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
< { _id: null, first_url: 'http://www.tutorialspoint.com' }
```

    b. Last

db.mycollection.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

```
> db.mycollection.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])
< { _id: null, last_url: 'http://www.tutorialspoint.com' }
```

b) MapReduce

Create a sample collection orders with these documents:

db.orders.insertMany([
{ _id: 1, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-01"), price: 25, items: [ { sku: "oranges", qty: 5, price: 2.5 }, { sku: "apples", qty: 5, price: 2.5 } ], status: "A" },

{ _id: 2, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-08"), price: 70, items: [ { sku: "oranges", qty: 8, price: 2.5 }, { sku: "chocolates", qty: 5, price: 10 } ], status: "A" },

{ _id: 3, cust_id: "Busby Bee", ord_date: new Date("2020-03-08"), price: 50, items: [ { sku: "oranges", qty: 10, price: 2.5 }, { sku: "pears", qty: 10, price: 2.5 } ], status: "A" },

{ _id: 4, cust_id: "Busby Bee", ord_date: new Date("2020-03-18"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },

{ _id: 5, cust_id: "Busby Bee", ord_date: new Date("2020-03-19"), price: 50, items: [ { sku: "chocolates", qty: 5, price: 10 } ], status: "A"},

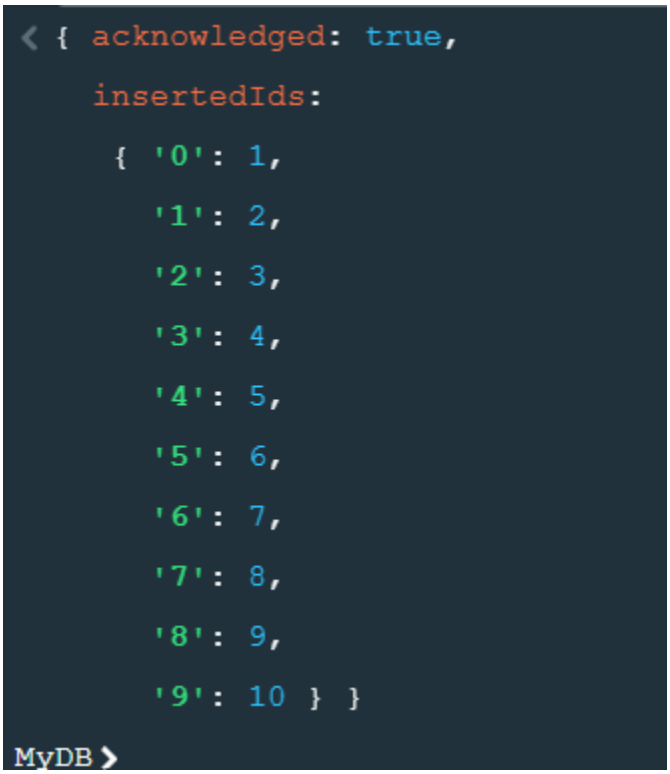
{ _id: 6, cust_id: "Cam Elot", ord_date: new Date("2020-03-19"), price: 35, items: [ { sku: "carrots", qty: 10, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 } ], status: "A" },

{ _id: 7, cust_id: "Cam Elot", ord_date: new Date("2020-03-20"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },

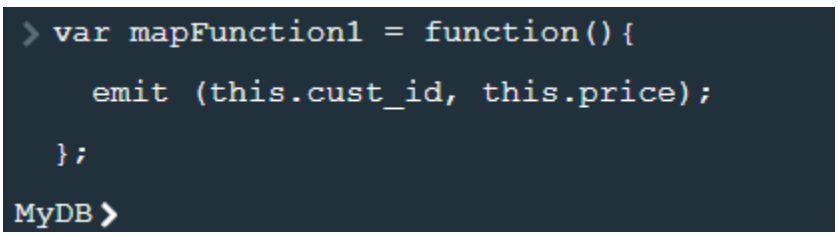
{ _id: 8, cust_id: "Don Quis", ord_date: new Date("2020-03-20"), price: 75, items: [ { sku: "chocolates", qty: 5, price: 10 }, { sku: "apples", qty: 10, price: 2.5 } ], status: "A" },

{ _id: 9, cust_id: "Don Quis", ord_date: new Date("2020-03-20"), price: 55, items: [ { sku: "carrots", qty: 5, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 }, { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" },

{ _id: 10, cust_id: "Don Quis", ord_date: new Date("2020-03-23"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ], status: "A" }
])




Define the map function to process each input document:




Define the corresponding reduce function with two arguments keyCustId and valuesPrices:

```
> var reduceFunction1 = function(keyCustId, valuesPrices){
    return Array.sum(valuesPrices);
  };
MyDB>
```

Perform map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function:

Using the available aggregation pipeline operators, you can rewrite the map-reduce operation without defining custom functions:

```
> db.orders.aggregate([
    {$group: {_id: "$cust_id",value:{$sum: "$price"}}},
    {$out: "agg_alternative_1"}
    ])
<
> db.agg_alternative_1.find().sort({_id:1})
< { _id: 'Ant O. Knee', value: 95 }
  { _id: 'Busby Bee', value: 125 }
  { _id: 'Cam Elot', value: 60 }
  { _id: 'Don Quis', value: 155 }
MyDB>
```

Name: Ashishkumar Rana
Roll no: 903
Subject: Big Data Analytics



## Practical 11 - Sorting and Indexing MongoDB database

a. sort() Method

1. Create a collection named ‘bookDetails’ and insert some documents with the fields like bookid, bookName, bookAuthor, bookPub, bookPrice and bookEdition.

```
db.bookDetails.insertMany([
  {bookid: 1, bookName: 'It ends with us', bookAuthor: 'colleen Hoover', bookPub: 'Atria', bookPrice: 300, bookEdition: 1},

  {bookid: 2, bookName: 'Murakami', bookAuthor: 'Haruki Murakami', bookPub: 'Kodansha', bookPrice: 250, bookEdition: 1},

  {bookid: 3, bookName: 'Atomic Habits', bookAuthor: 'James Clear', bookPub: 'Avery', bookPrice: 400, bookEdition: 3},

  {bookid: 4, bookName: 'All good people here', bookAuthor: 'Ashley Flowers', bookPub: 'Bantam', bookPrice: 375, bookEdition: 2}
  ])
```

```
> db.bookDetails.insertMany([
    {bookid: 1, bookName: 'It ends with us', bookAuthor: 'colleen Hoover', bookPub: 'Atria', bookPrice: 300, bookEdition: 1},

    {bookid: 2, bookName: 'Murakami', bookAuthor: 'Haruki Murakami', bookPub: 'Kodansha', bookPrice: 250, bookEdition: 1},

    {bookid: 3, bookName: 'Atomic Habits', bookAuthor: 'James Clear', bookPub: 'Avery', bookPrice: 400, bookEdition: 3},

    {bookid: 4, bookName: 'All good people here', bookAuthor: 'Ashley Flowers', bookPub: 'Bantam', bookPrice: 375, bookEdition: 2}
    ])
< { acknowledged: true,
    insertedIds:
     { '0': ObjectId("636168c61b7aa4532977f6fd"),
       '1': ObjectId("636168c61b7aa4532977f6fe"),
       '2': ObjectId("636168c61b7aa4532977f6ff"),
       '3': ObjectId("636168c61b7aa4532977f700") } }
MyDB >
```

2. Sort the documents based on bookAuthor.

```
> db.bookDetails.find().sort({bookAuthor:1})
< { _id: ObjectId("636168c61b7aa4532977f700"),
    bookid: 4,
    bookName: 'All good people here',
    bookAuthor: 'Ashley Flowers',
    bookPub: 'Bantam',
    bookPrice: 375,
    bookEdition: 2 }
  { _id: ObjectId("636168c61b7aa4532977f6fe"),
    bookid: 2,
    bookName: 'Murakami',
    bookAuthor: 'Haruki Murakami',
    bookPub: 'Kodansha',
    bookPrice: 250,
    bookEdition: 1 }
  { _id: ObjectId("636168c61b7aa4532977f6ff"),
    bookid: 3,
    bookName: 'Atomic Habits',
    bookAuthor: 'James Clear',
    bookPub: 'Avery',
    bookPrice: 400,
    bookEdition: 3 }
```

```
  { _id: ObjectId("636168c61b7aa4532977f6fd"),
    bookid: 1,
    bookName: 'It ends with us',
    bookAuthor: 'colleen Hoover',
    bookPub: 'Atria',
    bookPrice: 300,
    bookEdition: 1 }
MyDB>
```

3. Sort the documents on bookName and bookAuthor.

```
> db.bookDetails.aggregate([{$sort: {bookName: 1, bookAuthor: 1}}])
```

OR

```
> db.bookDetails.find().sort({bookName: 1,
    bookAuthor : 1
  })
< { _id: ObjectId("636168c61b7aa4532977f700"),
    bookid: 4,
    bookName: 'All good people here',
    bookAuthor: 'Ashley Flowers',
    bookPub: 'Bantam',
    bookPrice: 375,
    bookEdition: 2 }
  { _id: ObjectId("636168c61b7aa4532977f6ff"),
    bookid: 3,
    bookName: 'Atomic Habits',
    bookAuthor: 'James Clear',
    bookPub: 'Avery',
    bookPrice: 400,
    bookEdition: 3 }
```

```
{ _id: ObjectId("636168c61b7aa4532977f6fd"),
  bookid: 1,
  bookName: 'It ends with us',
  bookAuthor: 'colleen Hoover',
  bookPub: 'Atria',
  bookPrice: 300,
  bookEdition: 1 }
```

```
{ _id: ObjectId("636168c61b7aa4532977f6fe"),
  bookid: 2,
  bookName: 'Murakami',
  bookAuthor: 'Haruki Murakami',
  bookPub: 'Kodansha',
  bookPrice: 250,
  bookEdition: 1 }
```

4. Sort the documents on bookPub in descending order.

```
{ _id: ObjectId("636168c61b7aa4532977f6fd"),
  bookid: 1,
  bookName: 'It ends with us',
  bookAuthor: 'colleen Hoover',
  bookPub: 'Atria',
  bookPrice: 300,
  bookEdition: 1 }
MyDB>
```

b. Metadata Sort Technique

5. Search the documents for any one bookName and sort the query result documents using the metadata ‘testscore’.

Example

db.books.find({$text:{$search: "python"}}, {score:{$meta: "textScore"}, _id: 0}).sort({sort_example:{$meta: "textScore"}})

c. ensureIndex() and createIndex() Method

>> db.collection.ensureIndex() has been replaced by db.collection.createIndex()

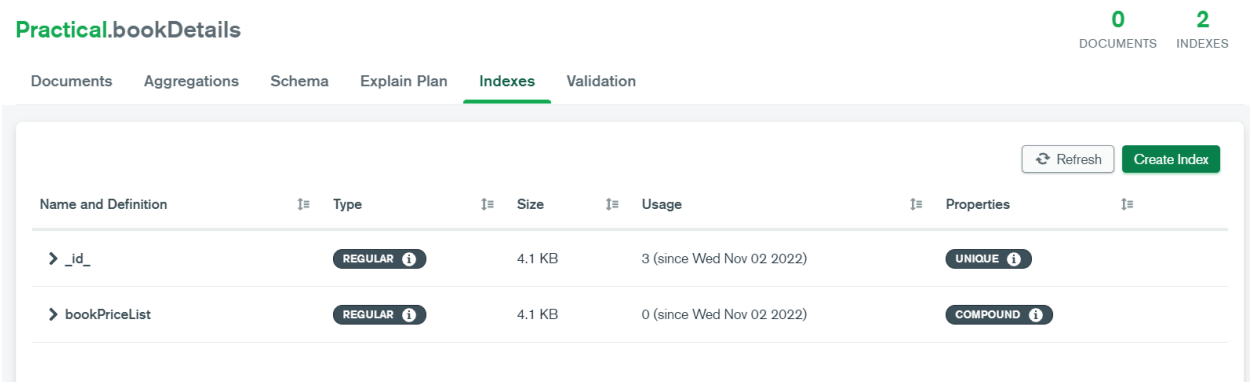
6. Check for the existence og the index named ‘bookPriceList’.

```
> db.bookDetails.getIndexes()
< [ { v: 2, key: { _id: 1 }, name: '_id_' } ]
MyDB >
```

7. Create an index named ‘bookPriceList’ on the bookName in ascending order and bookPrice in decreasing order.

Example

db.nooks.createIndex({ bookName: 1, bookPrice: -1 } ,{ name: "bookPriceList" })



8. Get the list of books sorted by ‘bookPriceList’.

```
> db.bookDetails.find().sort({bookPriceList:1})
```