PRACTICAL –1

AIM: -To install Cloudera QuickStart VM

PROCEDURE: -

- The Cloudera QuickStart VMs are openly available as Zip archives in VirtualBox, VMware and KVM formats. To download the VM, search for https://www.cloudera.com/downloads.html, and select the appropriate version of CDH that you require.

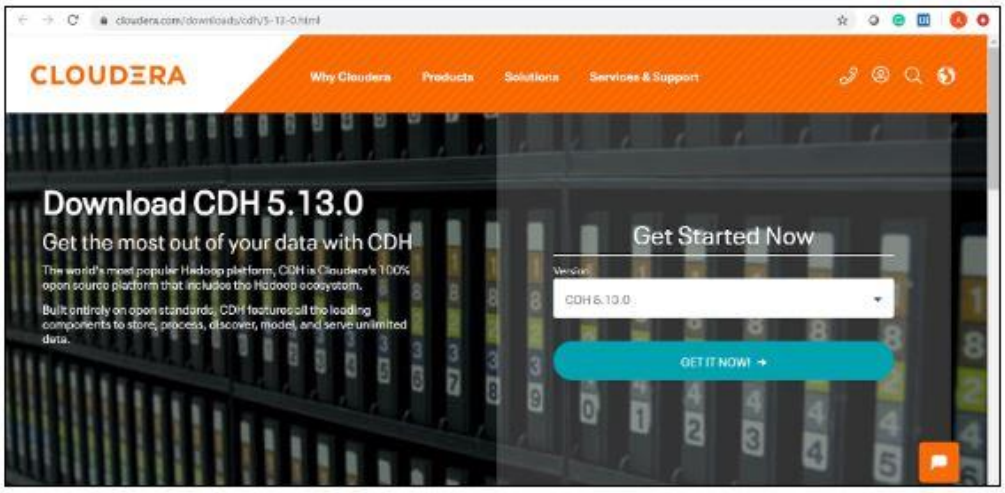


Fig: Download Cloudera QuickStart VM

- Click on the 'GET IT NOW' button, and it will prompt you to fill in your details.
- Once the file is downloaded, go to the download folder and unzip these files. It can then be used to set up a single node Cloudera cluster.
- Shown below are the two virtual images of Cloudera QuickStart VM.

- Now that the downloading process is done with, let's move forward with this Cloudera QuickStart VM Installation guide and see the actual process.

Cloudera QuickStart VM Installation

- Before setting up the Cloudera Virtual Machine, you would need to have a virtual machine such as VMware or Oracle VirtualBox on your system.
- In this case, we are using Oracle VirtualBox to set up the Cloudera QuickStart VM.
- In order to download and install the Oracle VirtualBox on your operating system, click on the following link: Oracle VirtualBox.
- To set up the Cloudera QuickStart VM in your Oracle VirtualBox Manager, click on ‘File’ and then select ‘Import Appliance’.

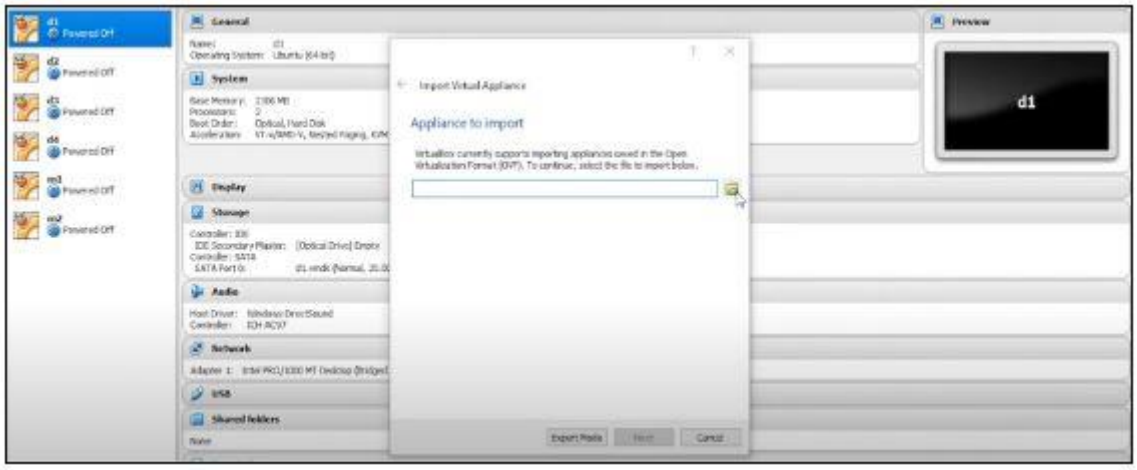
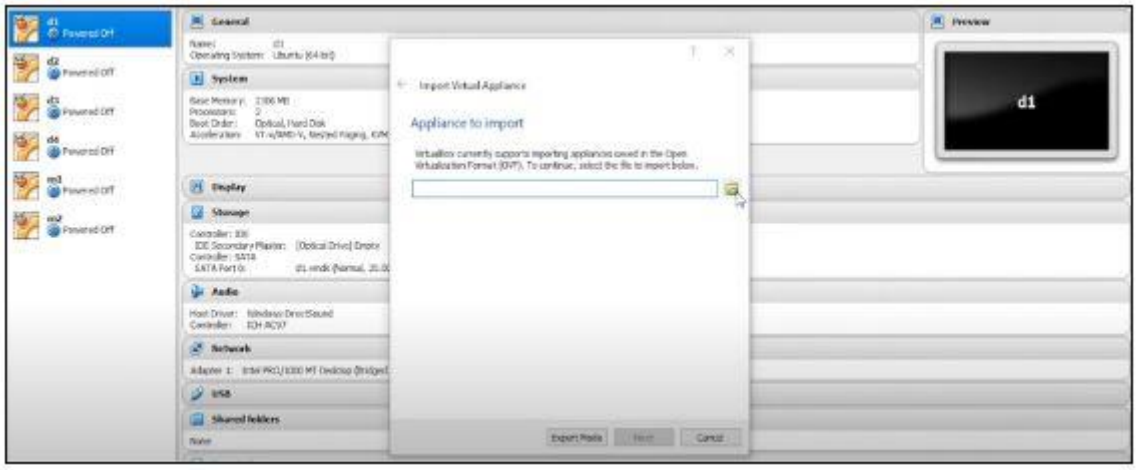


Fig: Importing the Cloudera QuickStart VM image

- Choose the QuickStart VM image by looking into your downloads. Click on ‘Open’ and then ‘Next’. Now you can see the specifications, then click on ‘Import’. This will start importing the virtual disk image .vmdk file into your VM box.

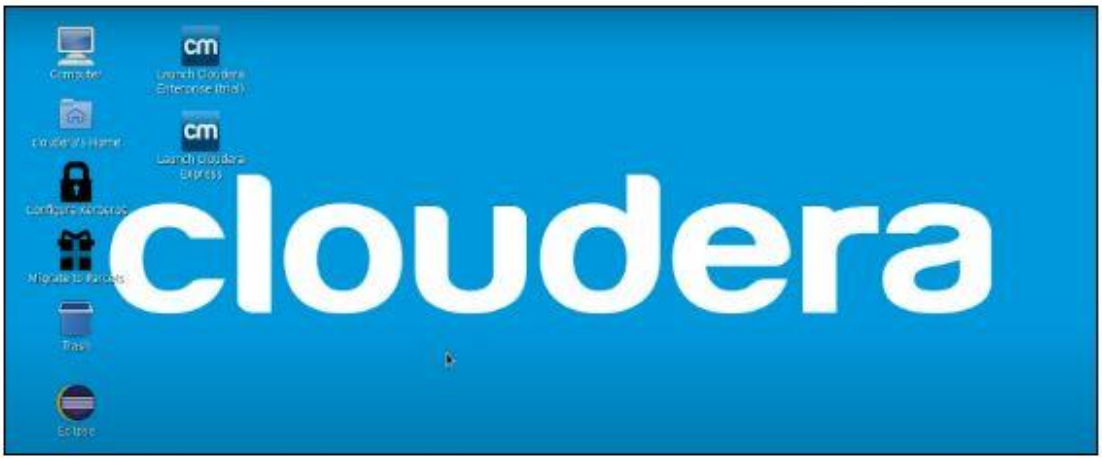


- Once this is done, we have to change the specifications of the machines to use. Here, we are giving 2 CPU cores and 5GB RAM. Wait for a while, as the importing finishes. The next step is to go ahead and set up a Cloudera QuickStart VM for practice. Once the importing is complete, you can see the Cloudera QuickStart VM on the left side panel.

Fig: Cloudera VM set up successful

- Now, to give more RAM and CPU cores, click on ‘Settings’, followed by ‘System’, and increase the RAM to 5GB. Click on the processor and assign 2 CPU cores. Subsequently, select ‘Network’. The Adapter 1 settings should be NAT by default. Click on ‘OK’ next.
- Now you are required to start the machine, so that it uses 2 CPU cores, 5GB RAM, and brings up the Cloudera QuickStart VM.
- The next step will be going ahead and starting the machine by clicking the ‘Start’ symbol on top.

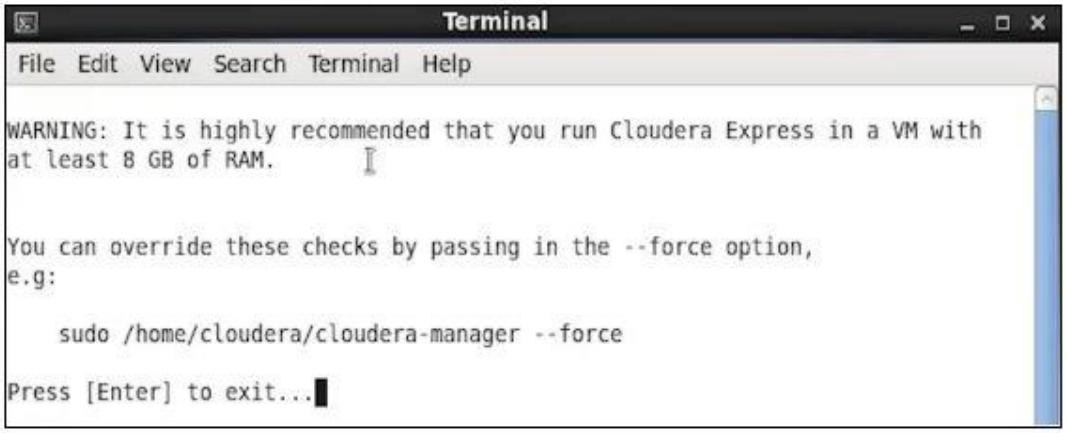
- Once your machine comes on, it will look like this:



- Next, we must follow a few steps to gain admin console access. You need to click on the terminal present on top of the desktop screen, and type in the following:

- Once you see that your HDFS access is working fine, you can close the terminal. Then, you have to click on the following icon that says ‘Launch Cloudera Express’.

- Once you click on the express icon, a screen will appear with the following command:



- You are required to copy the command, and run it on a separate terminal. Hence, open a new terminal, and use the below command to close the Cloudera based services. It will restart the services, after which you can access your admin console.

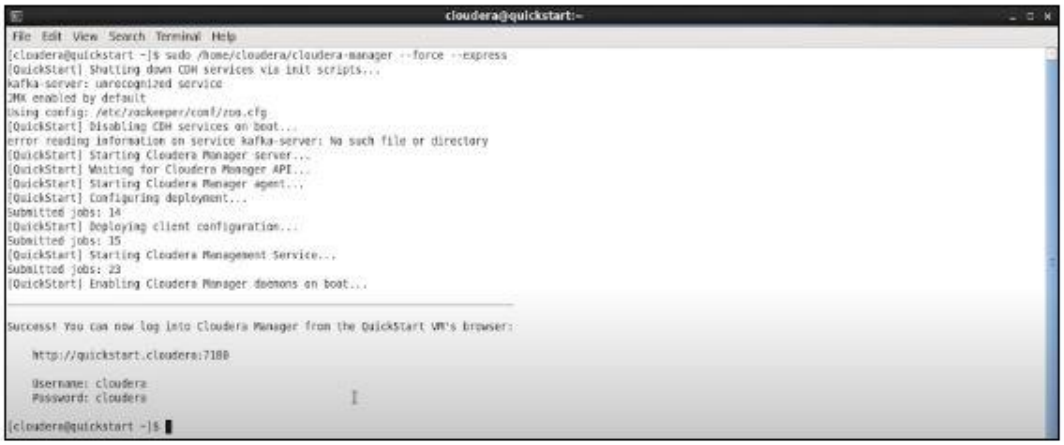


Fig: Restarting services on Cloudera QuickStart VM

- Now that our deployment has been configured, client configurations have also been deployed. Additionally, it has restarted the Cloudera Management Service, which gives access to the Cloudera QuickStart admin console with the help of a username and password.
    - Go on and open up the browser and change the port number to 7180.
    - You can log in to the Cloudera Manager by providing your username and password.

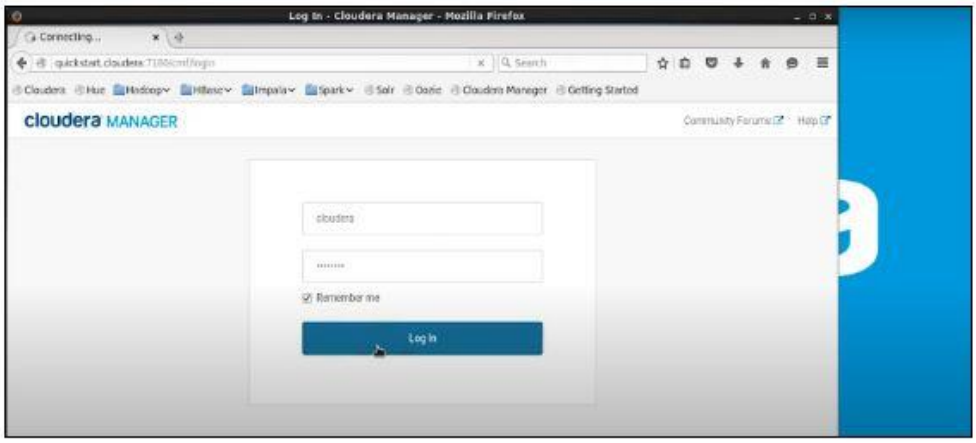


Fig: Logging in to Cloudera Manager

PRACTICAL –2

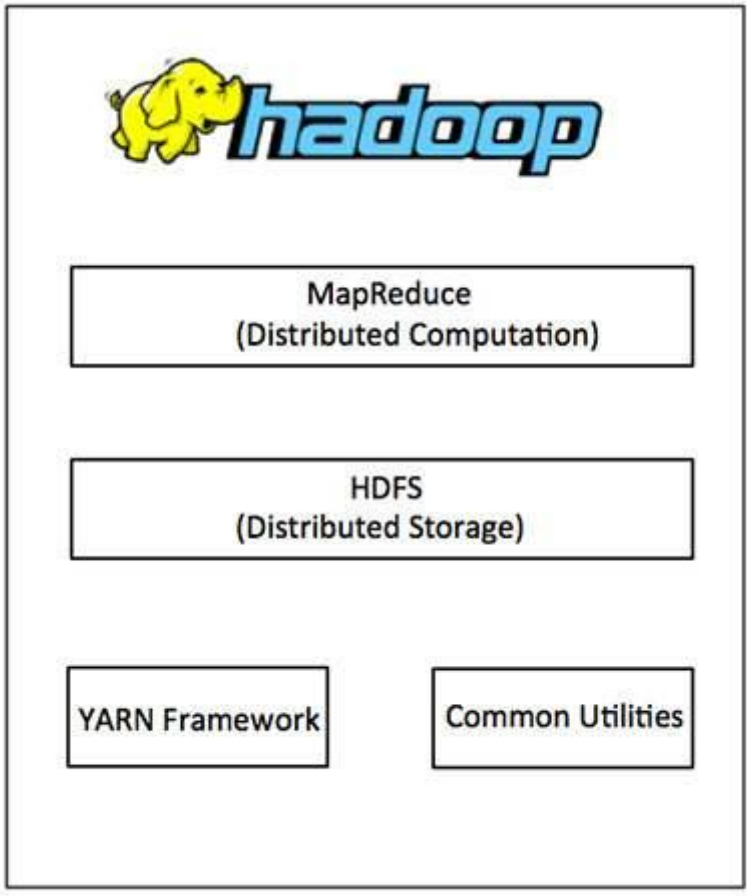
AIM: -Introduction to Hadoop.

PROCEDURE: -Hadoop is an Apache open-source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed *storage* and *computation* across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

At its core, Hadoop has two major layers namely −

- Processing/Computation layer (MapReduce), and
- Storage layer (Hadoop Distributed File System).



MapReduce

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data.sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules −

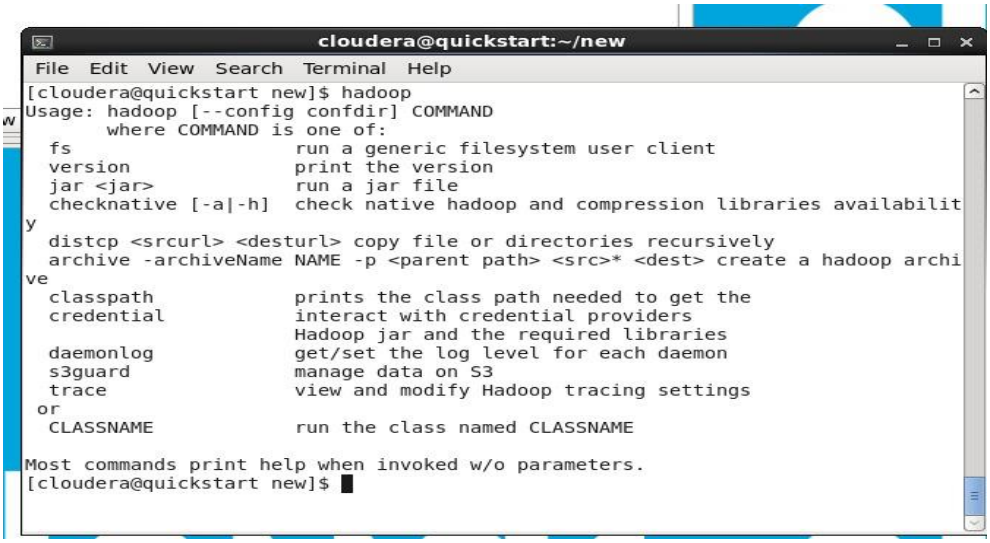
- **Hadoop Common** − These are Java libraries and utilities required by other Hadoop modules.
- **Hadoop YARN** − This is a framework for job scheduling and cluster resource management.

PRACTICAL –3

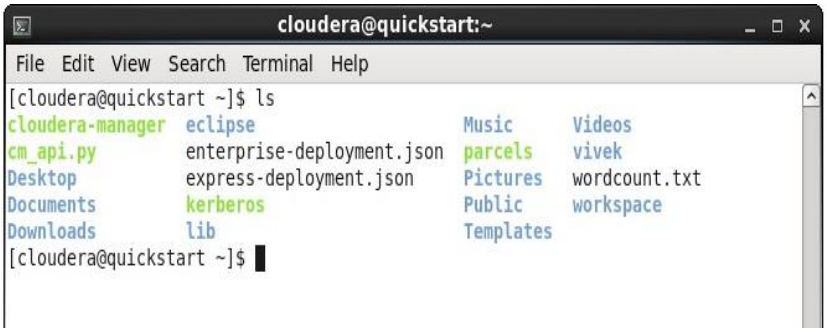
AIM: -To Execute HDFS Commands.

PROCEDURE: - Hdfs commands are as follows:

- **Hadoop:** With the help of the HDFS command, we can perform Hadoop HDFS file operations like changing the file permissions, viewing the file contents, creating files or directories, copying file/directory from the local file system to HDFS or vice-versa, etc.



- **ls:** This command is used to list all the files. Use *lsr* for recursive approach. It is useful when we want a hierarchy of a folder.
  Syntax:
  bin/hdfs dfs -ls <path>



- **mkdir:** To create a directory. In Hadoop dfs there is no home directory by default. So, let's first create it.

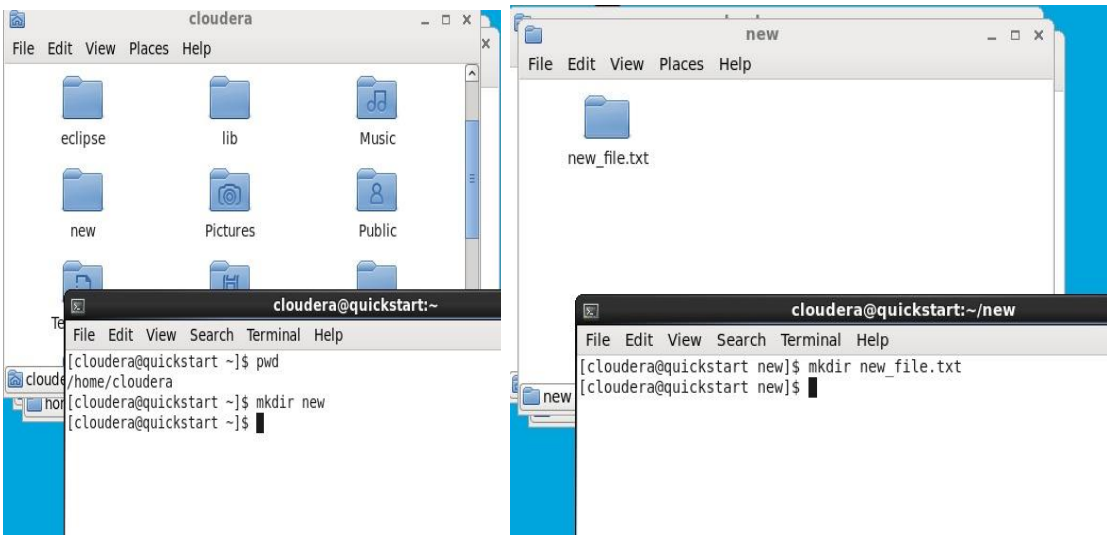
Syntax:

bin/hdfs dfs -mkdir <folder name>

creating home directory:

hdfs/bin -mkdir /user

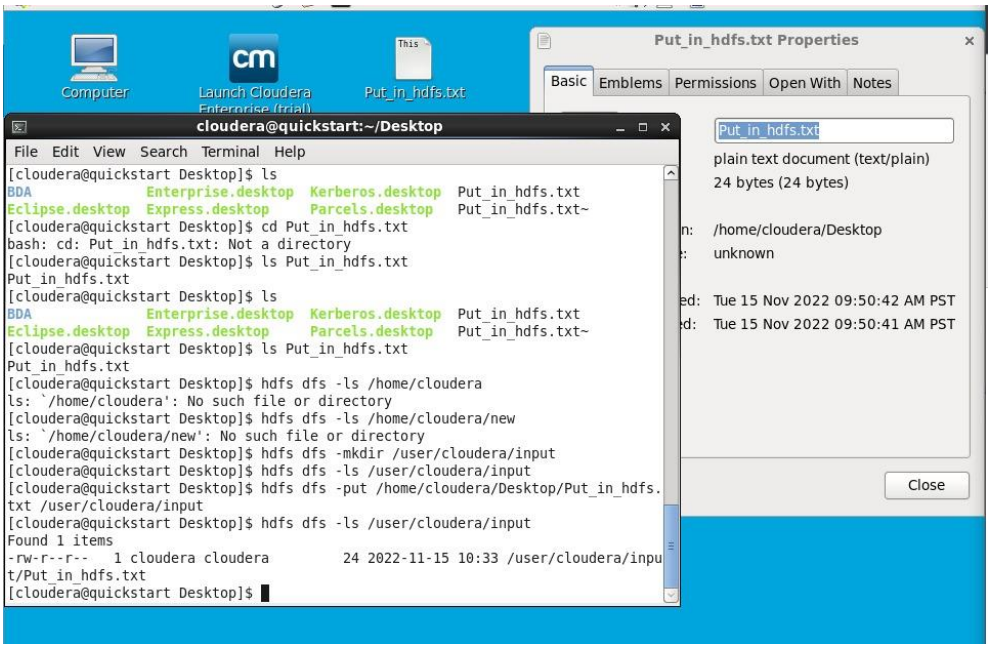
hdfs/bin -mkdir /user/username



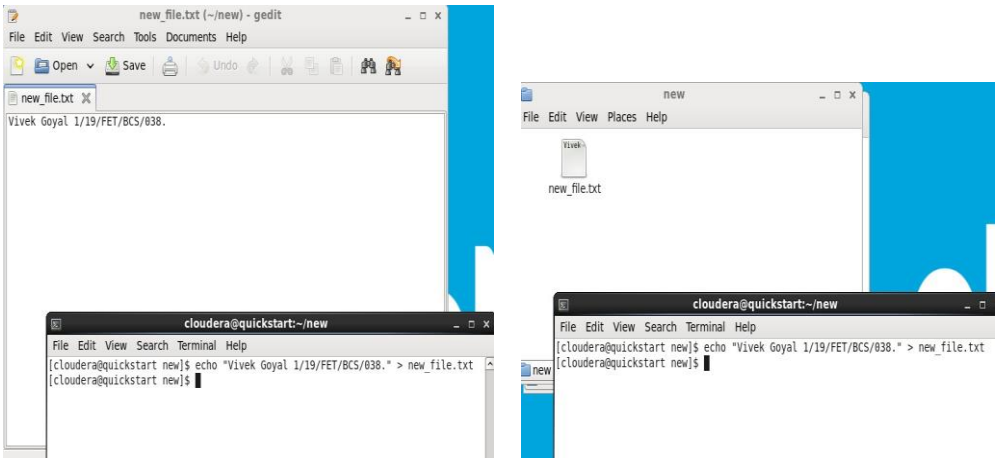
- **Put:** The Hadoop fs shell command put is similar to the copyFromLocal, which copies files or directory from the local filesystem to the destination in the Hadoop filesystem.

Syntax:

haoop fs -put <localsrc> <dest>



- **Echo:** the echo command can be used for displaying a line of string/text that is passed as the arguments.
  Syntax:
  echo [option] [string]



PRACTICAL –4

AIM: -WAP to count words in a file using a Map Reduce application

PROCEDURE: -Word Count MapReduce Program in Hadoop

The first MapReduce program most of the people write after installing Hadoop is invariably the word count MapReduce program.
The detailed steps for writing word count MapReduce program in Java, IDE used is Eclipse.

Creating and copying file

1. If you already have a file in HDFS which you want to use as input then you can skip this step.
2. First thing is to create a file which will be used as input and copy it to HDFS.
3. Let's say you have a file wordcount.txt with the following content.
4. Hello wordcount MapReduce Hadoop program.
5. This is my first MapReduce program.
6. You want to copy this file to /user/process directory with in HDFS. If that path doesn't exist, then you need to create those directories first.
   hdfsdfs -mkdir -p /user/process
7. Then copy the file wordcount.txt to this directory.
   hdfsdfs -put /netjs/MapReduce/wordcount.txt /user/process

Word count example MapReduce code in Java

1. Write your wordcount MapReduce code. WordCount example reads text files and counts the frequency of the words. Each mapper takes a line of the input file as input and breaks it into words. It then emits a key/value pair of the word (In the form of (word, 1)) and each reducer sums the counts for each word and emits a single key/value with the word and sum.
2. In the word count MapReduce code, there is a Mapper class (MyMapper) with map function and a Reducer class (MyReducer) with a reduce function.
3. You will also need to add at least the following Hadoop jars so that your code can compile. You will find these jars inside the /share/hadoop directory of your Hadoop installation. With in /share/hadoop path look in hdfs, mapreduce and common directories for required jars.

hadoop-common-2.9.0.jar
hadoop-hdfs-2.9.0.jar
hadoop-hdfs-client-2.9.0.jar

hadoop-mapreduce-client-core-2.9.0.jar
hadoop-mapreduce-client-common-2.9.0.jar
hadoop-mapreduce-client-jobclient-2.9.0.jar
hadoop-mapreduce-client-hs-2.9.0.jar
hadoop-mapreduce-client-app-2.9.0.jar
commons-io-2.4.jar

Creating jar of your wordcount MapReduce code

Once you are able to compile your code you need to create jar file. In the eclipse IDE righ click on your Java program and select Export – Java – jar file.

Running the code

You can use the following command to run the program in your hadoop installation directory.

bin/hadoop jar /netjs/MapReduce/wordcount.jar org.netjs.WordCount /user/process /user/out

/netjs/MapReduce/wordcount.jar is the path to your jar file.

org.netjs.WordCount is the fully qualified path to your Java program class.

/user/process – path to input directory.

/user/out – path to output directory.

One your word count MapReduce program is successfully executed you can verify the output file.

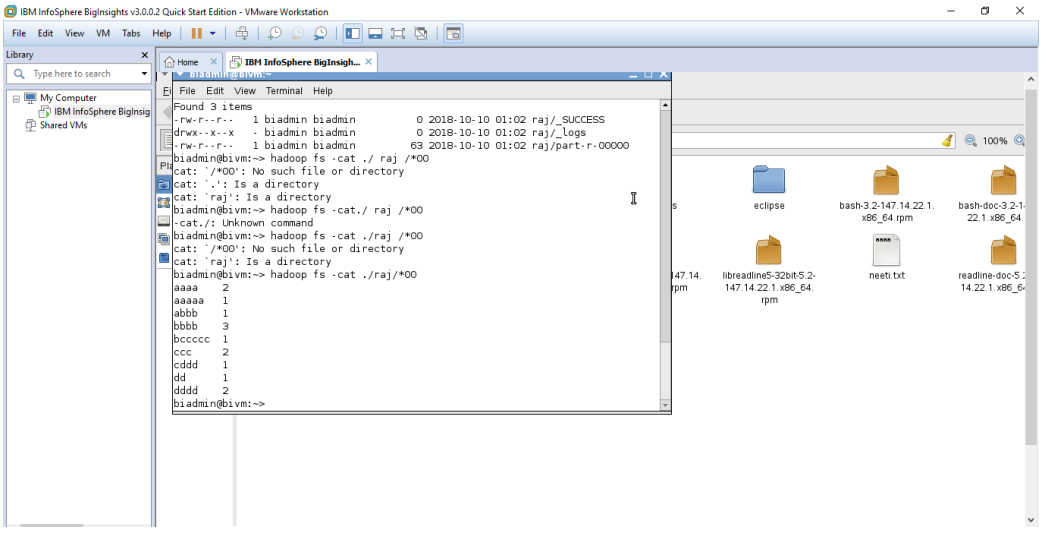
hdfsdfs -ls /user/out

Found 2 items

-rw-r--r-- 1 netjs supergroup 0 2018-02-27 13:37 /user/out/_SUCCESS

-rw-r--r-- 1 netjs supergroup 77 2018-02-27 13:37 /user/out/part-r-00000

As you can see Hadoop framework creates output files using part-r-xxxx format. Since only one reducer is used here so there is only one output file part-r-00000.



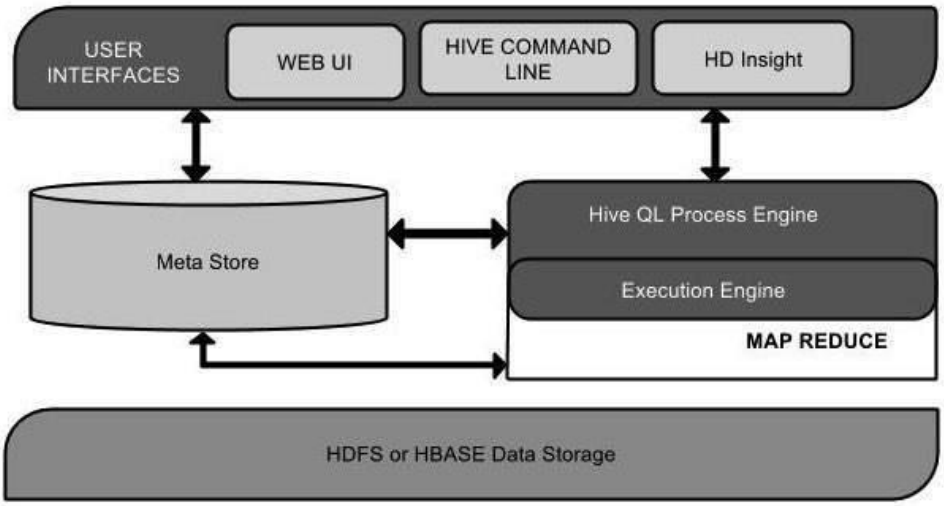
PRACTICAL –5

AIM: -Introduction to the Hive and Execution of Commands

PROCCEDURE: - Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data and makes querying and analysing easy. Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Architecture of Hive

The following component diagram depicts the architecture of Hive:



Some basic commands of Hive are:

Create Database Statement:

Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables.

Syntax: CREATE DATABASE|SCHEMA<database name>

```
1 CREATE DATABASE Employee_info;
2 use Employee_info;
```

Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax: CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]

[COMMENT table_comment]

[ROW FORMAT row_format]

[STORED AS file_format]

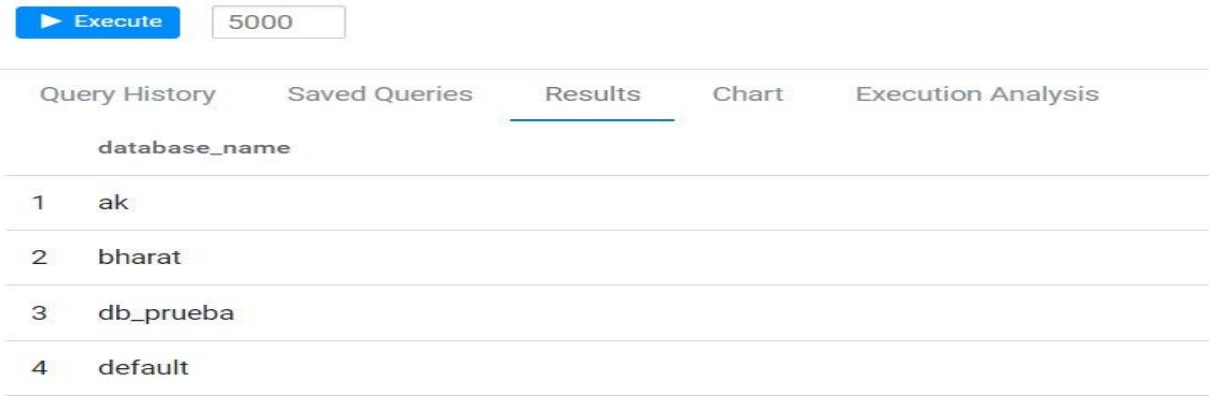
```
1 CREATE TABLE IF NOT EXISTS Employee(
2 name STRING,
3 Salary FLOAT)
4 ROW FORMAT DELIMITED
5 FIELDS TERMINATED BY ',';
```

Show Databases

Let's verify the creation of these databases in Hive CLI with show databases command. It will list down the databases in hive.

Syntax: SHOW (DATABASES|SCHEMAS) [LIKE identifier_with_wildcards];

```
1 show databases;
```



PRACTICAL-7

AIM: -HBase shell command

PROCEDURE: -HBase Shell

HBase contains a shell using which you can communicate with HBase. HBase uses the Hadoop File System to store its data. It will have a master server and region servers. The data storage will be in the form of regions (tables). These regions will be split up and stored in region servers.

The master server manages these region servers, and all these tasks take place on HDFS. Given below are some of the commands supported by HBase Shell.

Creating a Table using HBase Shell

- You can create a table using the create command, here you must specify the table name and the Column Family name. The syntax to create a table in HBase shell is shown on left page.
- Listing a Table using HBase Shell

  List is the command that is used to list all the tables in HBase. Given on the left page is the syntax of the list command.
- Disabling a Table using HBase Shell
- To delete a table or change its settings, you need to first disable the table using the disabled command. You can re-enable it using the enable command.

Given below is the syntax to disable a table:

Enabling a Table using HBase Shell

- Syntax to enable a table: describe
- This command returns the description of the table. Its syntax is as follows: alter
- Alter is the command used to make changes to an existing table. Using this command, you can change the maximum number of cells of a column family, set and delete table scope operators, and delete a column family from a table.
- Changing the Maximum Number of Cells of a Column Family
- Given below is the syntax to change the maximum number of cells of a column family.

Deleting a Specific Cell in a Table:

Using the delete command, you can delete a specific cell in a table.

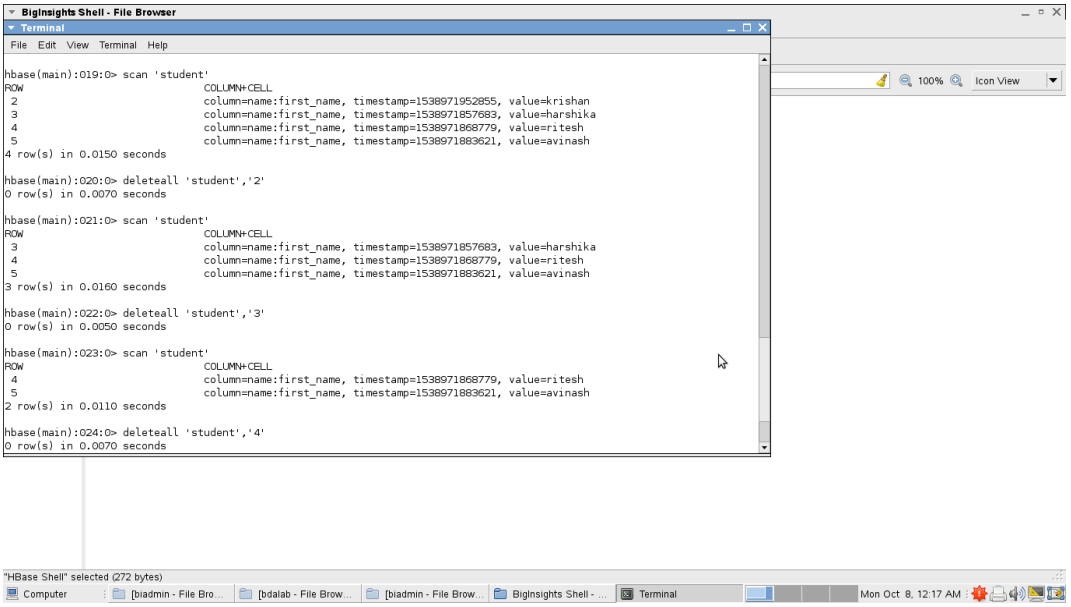
Inserting Data using HBase Shell

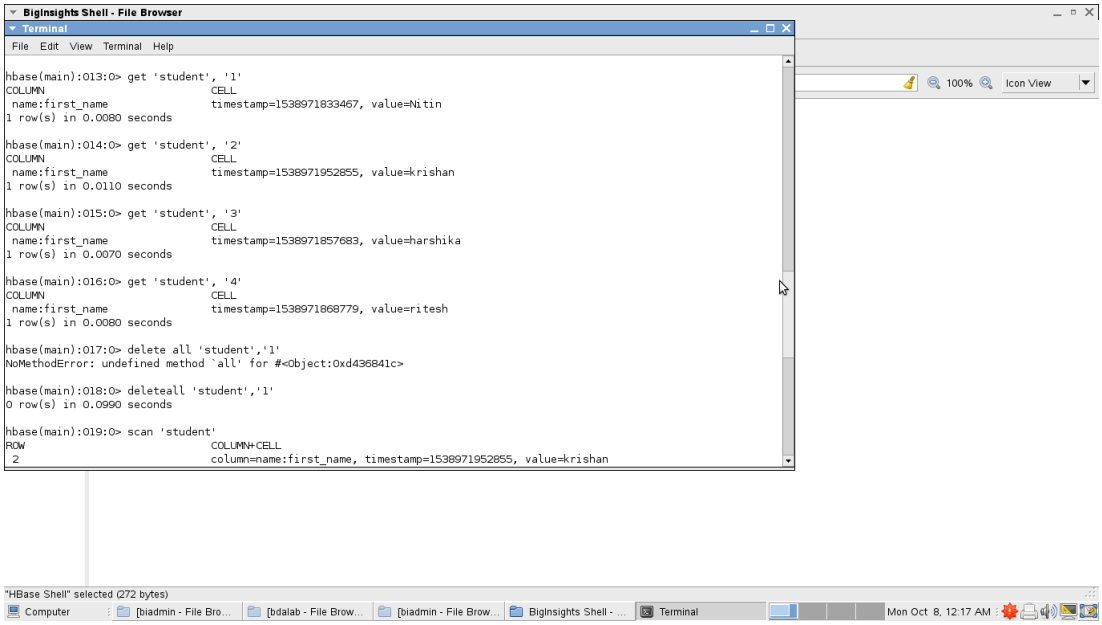
This chapter demonstrates how to create data in an HBase table. To create data in an HBase table, the following commands and methods are used:

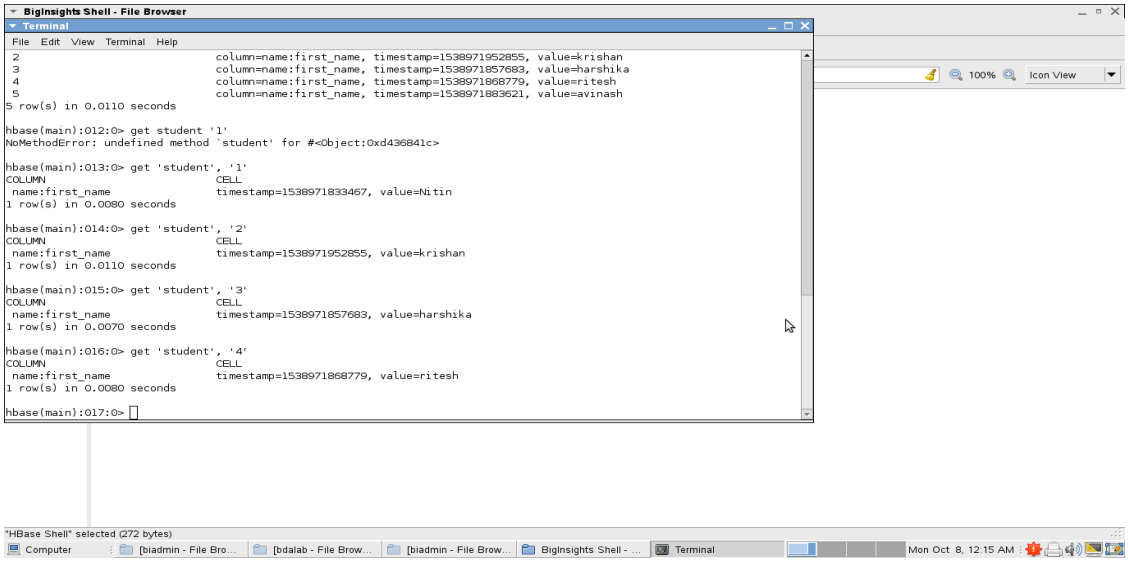
- put command,
- add() method of Put class, and
- put() method of HTable class.

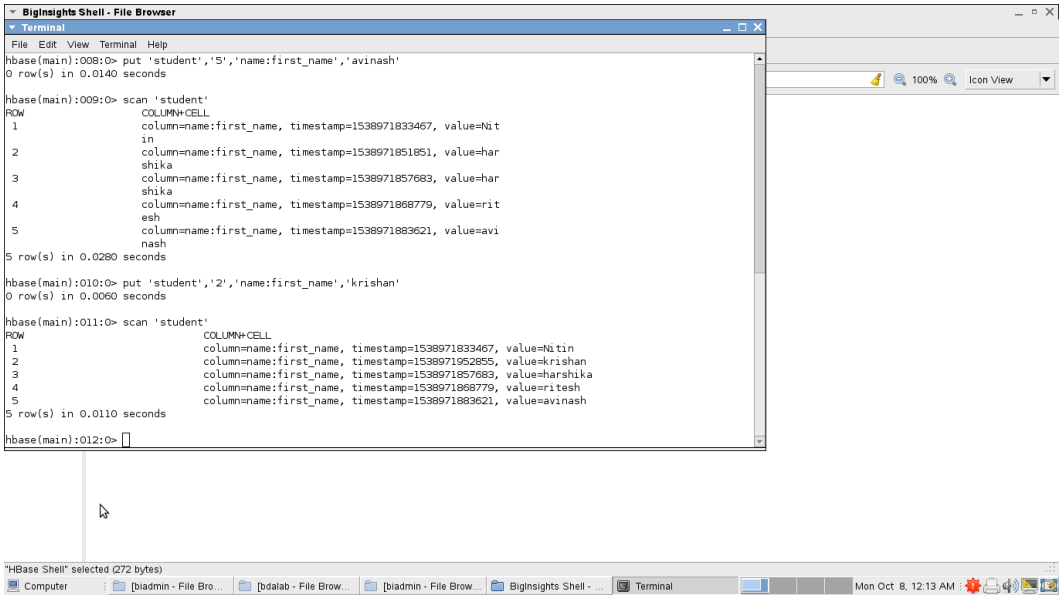
Inserting the First Row

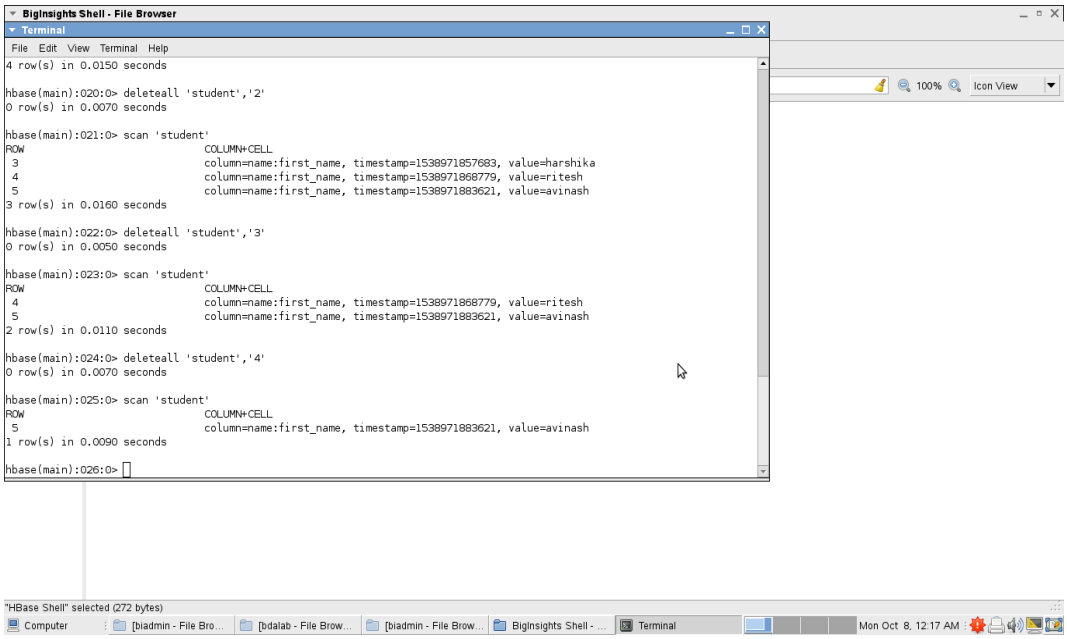
Let us insert the first-row values into the emp table as shown on the left page.






BigInsights Shell - File Browser
Terminal
File Edit View Terminal Help
2 column=name:first_name, timestamp=1538971952855, value=krishan
3 column=name:first_name, timestamp=1538971857683, value=harshika
4 column=name:first_name, timestamp=1538971868779, value=ritesh
5 column=name:first_name, timestamp=1538971883621, value=avinash
5 row(s) in 0.0110 seconds
hbase(main):012:0> get student '1'
NoMethodError: undefined method `student' for #<Object:0xd436841c>
hbase(main):013:0> get 'student', '1'
COLUMN CELL
name:first_name timestamp=1538971833467, value=Nitin
1 row(s) in 0.0080 seconds
hbase(main):014:0> get 'student', '2'
COLUMN CELL
name:first_name timestamp=1538971952855, value=krishan
1 row(s) in 0.0110 seconds
hbase(main):015:0> get 'student', '3'
COLUMN CELL
name:first_name timestamp=1538971857683, value=harshika
1 row(s) in 0.0070 seconds
hbase(main):016:0> get 'student', '4'
COLUMN CELL
name:first_name timestamp=1538971868779, value=ritesh
1 row(s) in 0.0080 seconds
hbase(main):017:0>
100%
Icon View
"HBase Shell" selected (272 bytes)
Computer
[biadmin - File Bro...
[bdalab - File Brow...
[biadmin - File Brow...
BigInsights Shell - ...
Terminal
Mon Oct 8, 12:15 AM


BigInsights Shell - File Browser
Terminal
File Edit View Terminal Help
hbase(main):008:0> put 'student','5','name:first_name','avinash'
0 row(s) in 0.0140 seconds
hbase(main):009:0> scan 'student'
ROW COLUMN+CELL
1 column=name:first_name, timestamp=1538971833467, value=Nit
in
2 column=name:first_name, timestamp=1538971851851, value=har
shika
3 column=name:first_name, timestamp=1538971857683, value=har
shika
4 column=name:first_name, timestamp=1538971868779, value=rit
esh
5 column=name:first_name, timestamp=1538971883621, value=avi
nash
5 row(s) in 0.0280 seconds
hbase(main):010:0> put 'student','2','name:first_name','krishan'
0 row(s) in 0.0060 seconds
hbase(main):011:0> scan 'student'
ROW COLUMN+CELL
1 column=name:first_name, timestamp=1538971833467, value=Nitin
2 column=name:first_name, timestamp=1538971952855, value=krishan
3 column=name:first_name, timestamp=1538971857683, value=harshika
4 column=name:first_name, timestamp=1538971868779, value=ritesh
5 column=name:first_name, timestamp=1538971883621, value=avinash
5 row(s) in 0.0110 seconds
hbase(main):012:0>
100%
Icon View
"HBase Shell" selected (272 bytes)
Computer
[biadmin - File Bro...
[bdalab - File Brow...
[biadmin - File Brow...
BigInsights Shell - ...
Terminal
Mon Oct 8, 12:13 AM


BigInsights Shell - File Browser
Terminal
File Edit View Terminal Help
4 row(s) in 0.0150 seconds
hbase(main):020:0> deleteall 'student','2'
0 row(s) in 0.0070 seconds
hbase(main):021:0> scan 'student'
ROW COLUMN+CELL
3 column=name:first_name, timestamp=1538971857683, value=harshika
4 column=name:first_name, timestamp=1538971868779, value=ritesh
5 column=name:first_name, timestamp=1538971883621, value=avinash
3 row(s) in 0.0160 seconds
hbase(main):022:0> deleteall 'student','3'
0 row(s) in 0.0050 seconds
hbase(main):023:0> scan 'student'
ROW COLUMN+CELL
4 column=name:first_name, timestamp=1538971868779, value=ritesh
5 column=name:first_name, timestamp=1538971883621, value=avinash
2 row(s) in 0.0110 seconds
hbase(main):024:0> deleteall 'student','4'
0 row(s) in 0.0070 seconds
hbase(main):025:0> scan 'student'
ROW COLUMN+CELL
5 column=name:first_name, timestamp=1538971883621, value=avinash
1 row(s) in 0.0090 seconds
hbase(main):026:0>
100%
Icon View
"HBase Shell" selected (272 bytes)
Computer
[biadmin - File Bro...
[bdalab - File Brow...
[biadmin - File Brow...
BigInsights Shell - ...
Terminal
Mon Oct 8, 12:17 AM

PRACTICAL-6

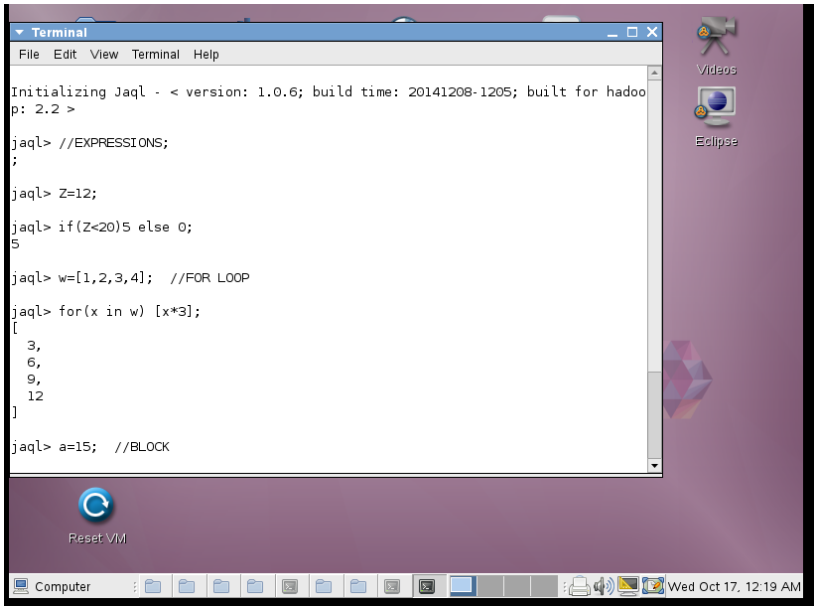
AIM: -To execute JAQL shell commands

PROCEDURE: -Jaql basics

- **Statement, Assignment and Comments**

Double and single quotes are treated the same. Semicolon terminates a statement.

```
jaql> "Hello world";
"Hello world"
jaql> a = 10*2;
jaql> a;
20
jaql> // This is a comment
jaql> /* and this is also
        a comment */
```



- **Data Types**

Jaql is a loosely typed functional language, with lazy evaluation. Type is usually inferred by how a value is provided. Many types have a function of the same name to force conversions of a value or variable (e.g. string(), double()).
null – null
boolean – true, false
string – “hi”
long – 10
double – 10.2, 10d, 10e-2
array – [1, 2, 3]
record – {a : 1, b : 2}
others as jaql extensions – decfloat, binary, date, schema, function, comparator, regex

```
jaql>num = 10;
jaql> array = [1, 2, 3, "hello", 4, {color: "red"}];
jaql> array;
[
  1,
  2,
  3,
  "hello"
  ...
```

- **Operators**

arithmetic (+, -, /, *)
boolean (and, or, not)
comparison (==, !=, <, >, in, isnull)

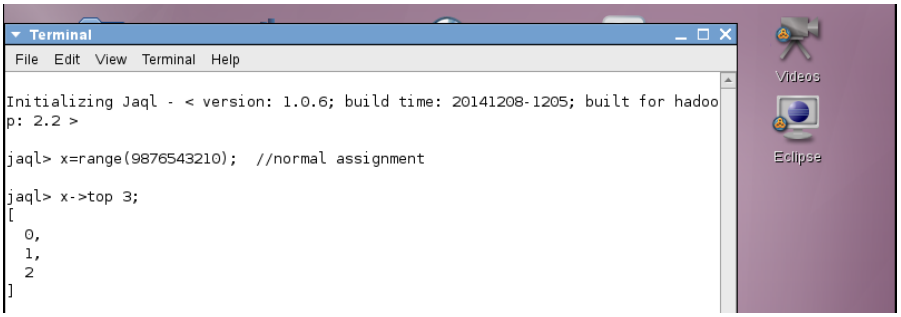
- **Arrays**

Arrays can be accessed with the [] operator.

```
jaql> a = [1, 2, 3];
jaql>a[1]; //retrieves start from zero
2
jaql> a[1:2]; //retrieve a range/subset
[2,3]
```

But you can’t change a value, you need to use function replaceElement().

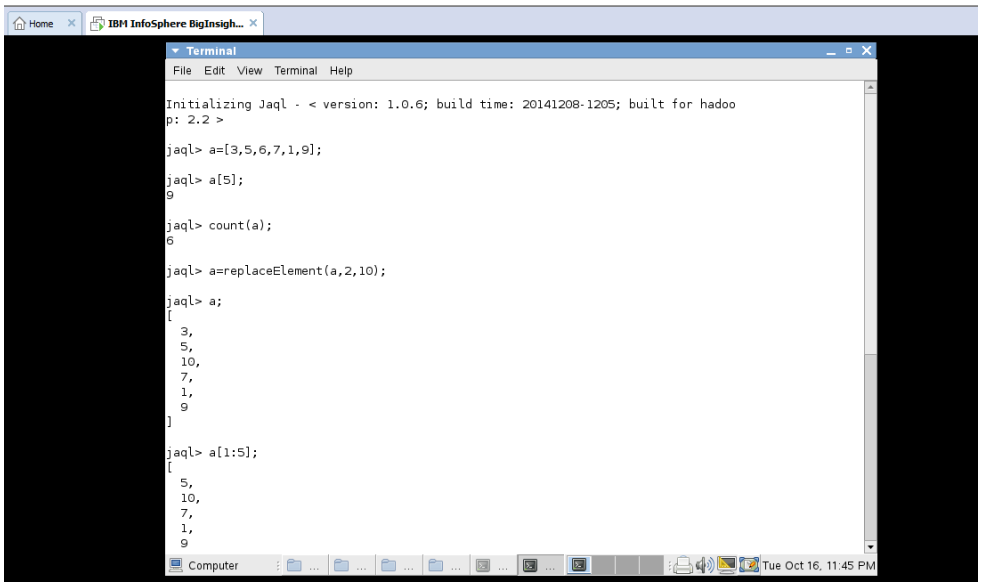
```
jaql> a = replaceElement(a, 1, 10);
jaql> a[1];
10
```

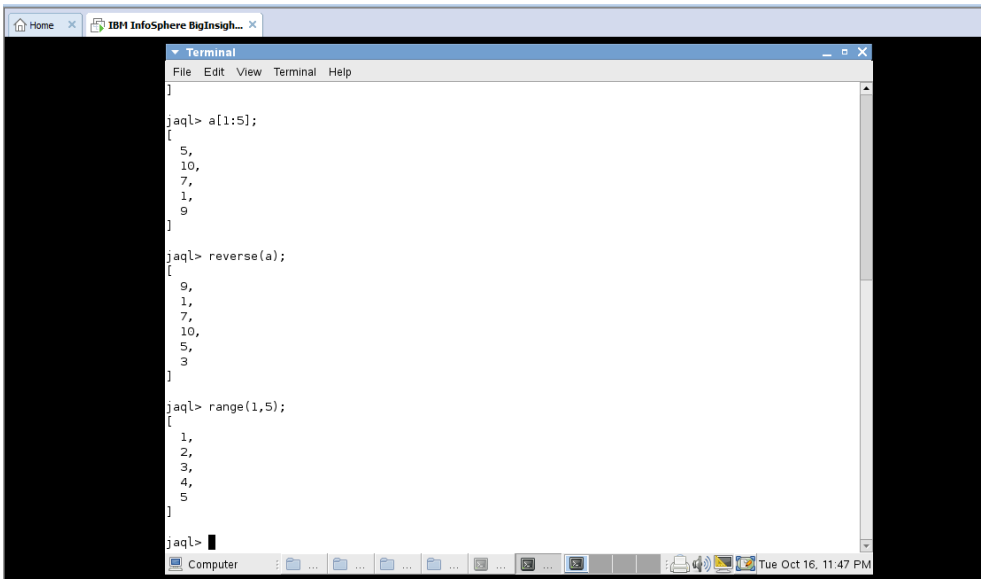



```
jaql> x:=range(99999999999999888888888887777777777777555555555555); //materializ
ed assignment
parse error: java.lang.NumberFormatException: For input string: "999999999999998
88888888888777777777777755555555555"
Ignoring input until semicolon ';' ...
encountered an exception during the evaluation of a statement
java.lang.NumberFormatException: For input string: "99999999999999888888888888877
7777777777555555555555"
```

- **Other array functions**

count() – count(['a', 'b', 'c']); returns 3
index() – index(['a', 'b', 'c'], 1); returns "b"
replaceElement() – replaceElement(['a', 'b', 'c'], 1, 'z'); returns ["a", "z", "c"]
slice() – index(['a', 'b', 'c', 'd'], 1, 2); returns ["b", "c"]
reverse() – reverse(['a', 'b', 'c']); returns ["c", "b", "a"]
range() – range(2, 5); returns [2, 3, 4, 5]
And many more – see BigInsights Information Center





- **Records**

Records are delineated by {} and contains a comma separated list of name:value pairs. Fields are then accessed by "." operator.

```
jaql> a = { name : "scott", age : 42, children : ["jake", "sam"] };
jaql> a.name;
"scott"
jaql>a.children[0];
"jake"
```

Again you cannot change an existing record, but you can produce a new one:

```
jaql> a = { a.name, age : 37, a.children };
```

- **The -> operator**

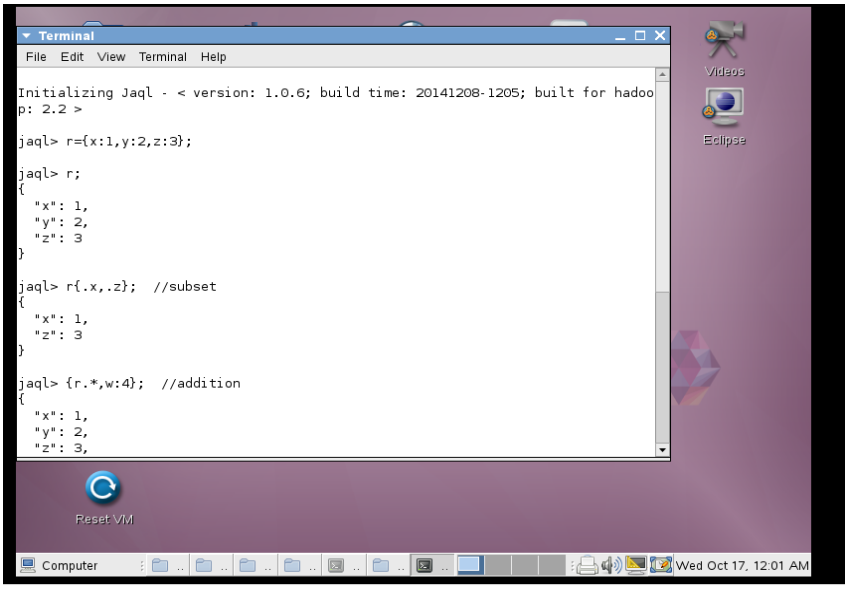
The -> operator "streams" an array through a function or core operator.

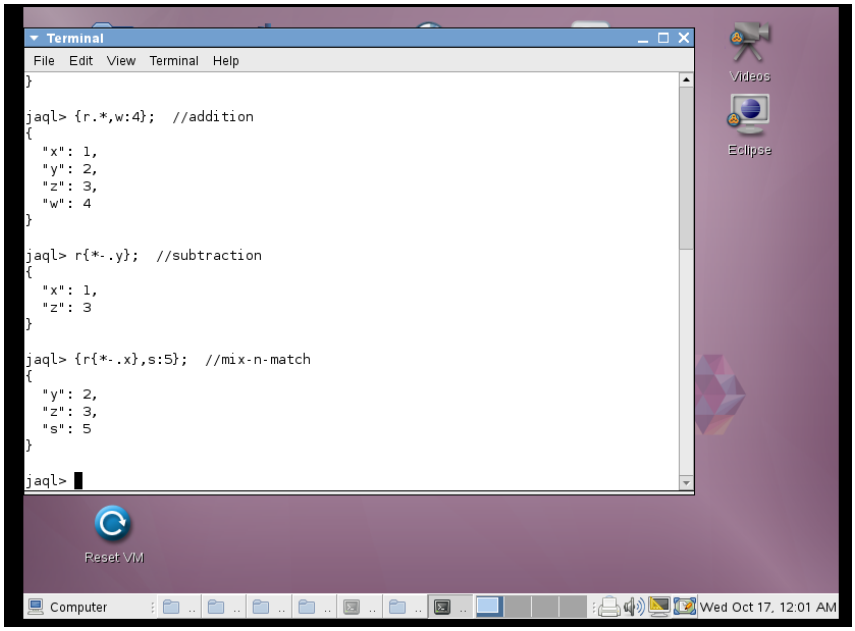
```
jaql>range(10) -> batch(5);
[
  [0, 1, 2, 3, 4],
  [5, 6, 7, 8, 9]
]
```

The array on the left is implicitly passed as the first argument to the function. This is identical to the above:

```
jaql> batch(range(10), 5);
```

Operator -> is just a "syntactic sugar" that can dramatically improve readability when multiple operations are involved on your data.





- **Input/Output with Jaql**

Input/Output operations are performed through I/O adapters. Adapters are a description of how to access and process a data source. I/O adapter is then passed into I/O function (e.g. read() or write()). Following is an example of writing array into a (comma-)delimited csv file where we used delimited file I/O adapter:

```
jaql> [1, 2, 3, 4, 5] -> write(del("test.csv"));
jaql> read(del("test.csv"));
[1, 2, 3, 4, 5]
```

Local files or HDFS are then accessed by specyfing the full path as URI:

```
jaql> read(del("file:///home/user/test.csv")); // for local file system
jaql> read(del("hdfs://localhost:9000/user/test.csv")); // for hdfs file system
```

To read a JSON data from a URL, you can use jaglGet() function:
jaqlGet("file:///tmp/test.txt");

```
jaql>
jaql> //FUNCTIONS;
;

jaql> add=fn(x,y)x+y;

jaql> add(1,2);      //addition in functions
3

jaql> a=fn(op,ar) for (val in ar) [op(val)];

jaql> a(fn(x) x * x, [1,2,3]);
[
  1,
  4,
  9
]

jaql>
```

Data manipulation (Core operators).
Core operators manipulate streams (arrays) of data, much in the way SQL clauses interact with data.

- **Filter**
  $ represents the current array value being evaluated.
  jaql> read(del("file:///path/to/people.txt")) -> filter $.name == "Fred";
  [
  { fname: "Fred", lname: "Johnson", age: 20 }
  ]
  Other example could be:
  jaql> data = [1, 2, 3, 4, 5, 6, 7, 8, 9];
  jaql> data -> filter 3 <= $ <= 6;
  [3, 4, 5, 6]
  Alternatively, the each clause can be used to provide a name different than $:
  jaql> data = [1, 2, 3, 4, 5, 6, 7, 8, 9];
  jakq> data -> filter each num (3 <= num<= 6);
  [3, 4, 5, 6]

```
jaql> data -> expand(slice($,0,0));
[
  1,
  4,
  7
]

jaql> data=[1,2,3,4,5,6,7,8,9];

jaql> data -> filter 3 <= $ <= 6;
[
  3,
  4,
  5,
  6
]
```

```
jaql> data -> filter each h(3 <= h <= 6);
[
  3,
  4,
  5,
  6
]
```

- **Transform**
  The transform operator allows you to manipulate the values in an array. An expression is applied to each element in the array:
  jaql> recs = [{a: 1, b: 4}, {a: 2, b: 5}, {a: -1, b: 4}];
  jaql> recs -> transform $.a + $.b;
  [5, 7, 4]
  jaql> recs -> transform { sum: $.a + $.b };
  [{ sum: 5 }, { sum: 7 }, { sum: 3 }]

```
jaql>
jaql> b= [{a:1,b:4},{a:-1,b:5}];

jaql> b -> transform $.a + $.b;
[
  5,
  4
]
```

```
jaql>
jaql> b -> transform {sum: $.a + $.b};
[
  {
    "sum": 5
  },
  {
    "sum": 4
  }
]
```

- **Sort**
  jaql> read(del("file:///path/to/people.txt")) -> sort by [$.ageasc];

```
jaql> //SORT;
;

jaql> x=[{name:"abc",age:12},{name:"xyz",age:15}];

jaql> x -> sort by[$.name];
[
  {
    "name": "abc",
    "age": 12
  },
  {
    "name": "xyz",
    "age": 15
  }
]
```

```
jaql> x -> sort by[$.name desc,$.age asc];
[
  {
    "name": "xyz",
    "age": 15
  },
  {
    "name": "abc",
    "age": 12
  }
]
```

Other data manipulation operators

Other data manipulation operators are expanded, group, join, top.

- **Join**

```
jaql> //JOIN;
;

jaql> users = [{n:"ab",pass:"abc123",id:1},{n:"cd",pass:"cde456",id:2}];

jaql> pages = [{uid:1,url:"www.google.com"},{uid:1,url:"www.yahoo.com"}];

jaql> join users,pages where users.id == pages.uid into {users.n,pages.*};
[
  {
    "n": "ab",
    "uid": 1,
    "url": "www.google.com"
  },
  {
    "n": "ab",
    "uid": 1,
    "url": "www.yahoo.com"
  }
]
```

- **Top**

```
jaql> //TOP;
;

jaql> data = [1,2,3,4,5,6];

jaql> data -> top 2;
[
  1,
  2
]

jaql> data -> top 3 by [$ desc];
[
  6,
  5,
  4
]

jaql>
```

- **Group**

```
jaql> //GROUP;
;

jaql> emp = [{id:1,dept:1,income:12000},{id:2,dept:1,income:13000},{id:3,dept:2,
income:15000}];

jaql> depts = [{did:1,name:"development"},{did:2,name:"marketing"},{did:3,name:"
sales"}];

jaql> emp -> group into count($);
[
  3
]

jaql> emp -> group by dept_id = $.dept into {dept_id,total_income : sum($[*].inc
ome)};
[
  {
    "dept_id": 2,
    "total_income": 15000
  },
  {
    "dept_id": 1,
    "total_income": 25000
  }
]
```