Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	01

AIM: Case study on any organization using big data technologies.

WHAT IS BIG DATA & WHY IS IT IMPORTANT?

Big data analytics refers to the methods, tools, and applications used to collect, process, and derive insights from varied, high-volume, high-velocity data sets.

This ability to derive insights to inform better decision making is why big data is important. It's how a retailer might hone their targeted ad campaigns, or how a wholesaler might resolve bottlenecks in the supply chain. Big data analytics enables a more holistic, data-driven approach to decision-making, in turn promoting growth, efficiency, and innovation.

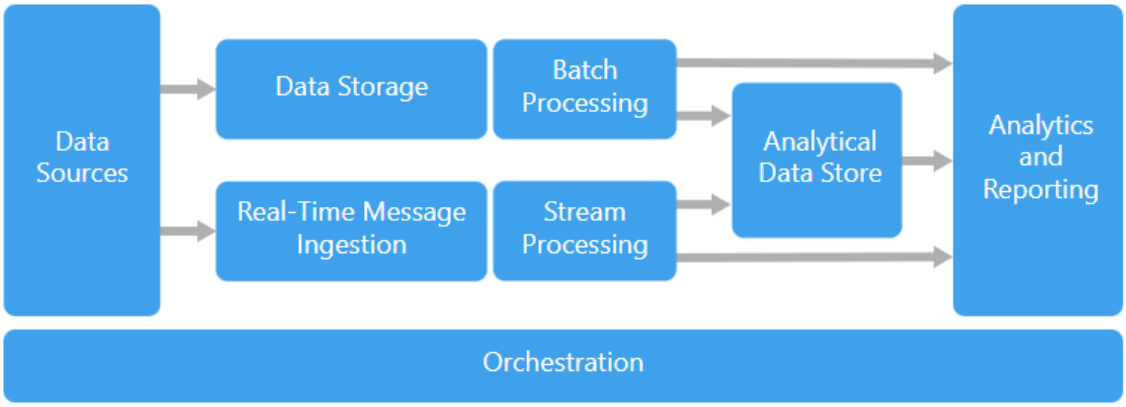
MICROSOFT

Introduction:

Microsoft makes extensive use of big data to improve its products and services. A well-known case study is Microsoft Azure, a cloud computing platform that uses various information technologies for data processing and analysis.

Big data applications:

- Microsoft Azure leverages many aspects of big data, including Apache Hadoop for distribution and processing, Apache Spark for big data processing, and Apache Spark for big data.
- Azure Data Lake Storage for scalable and secure data lakes.
- Azure also offers services like Azure Databricks and Azure HDInsight to simplify big data.
- A big data architecture is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.
- Components of a big data architecture:



- Microsoft leverages a variety of big data across its platforms and services, demonstrating how to process big data and providing insights from it.

- Some of the important information technologies used by Microsoft are:

1. Azure Data Lake Storage (ADLS): ADLS is designed to store big data in a distributed and secure manner. It provides a perfect solution for storing big data by supporting structured and unstructured data.

2. Azure HDInsight: This is a cloud-based big data analytics service that makes it easy to process big data using popular open source services such as Apache Hadoop, Spark, Hive, and more. HDInsight allows this process to be implemented on demand, providing a scalable and cost-effective solution.

3. Apache Hadoop: Microsoft integrates open-source Hadoop into its big data to store and process big data. Hadoop's Distributed File System (HDFS) & MapReduce programming model allow Microsoft to store & process data efficiently across clusters of computers.

4. Apache Spark: Apache Spark is a fast and versatile computing solution for big data processing. Microsoft is integrating Spark into real-time data analytics, machine learning, and image processing services to provide faster and more flexible data processing.

5. Azure Databricks: This is an analytics platform based on Apache Spark that is available as part of the Azure cloud service. Databricks simplifies the Spark cluster creation process and provides a unified platform for data scientists and architects to work together on big data analytics projects.

6. Azure Synapse Analytics (formerly SQL Data Warehouse): A cloud-based business data warehouse service that can process large amounts of data and provide close analysis. It integrates with various big data technologies to enable data storage and analysis of different data.

7. Azure Streaming Analytics: Microsoft leverages Azure Streaming Analytics for real-time data processing. The service enables analysis of data streams from multiple sources, providing rapid insights and actionable information.

Together, these technologies enable Microsoft to deliver a powerful ecosystem for big data that supports the storage, processing and analysis of diverse and large data sets. Listed as scalable and efficient.

- Advantages:
  1. Scalability: Large data sets allow Microsoft to scale its processes based on data needs.
  2. Analytics: Advanced analytics capabilities help Microsoft derive insights from large amounts of data to help make informed decisions.
  3. Real-time processing: Microsoft can help respond to user needs faster by processing and analysing data in real-time through technologies such as Apache Spark.

- Disadvantages:
  1. Complexity: Using and managing large data sets can be complex and require special skills and resources.
  2. Security issues: Processing large amounts of data can cause security issues and leaks can have serious consequences.

Conclusion:

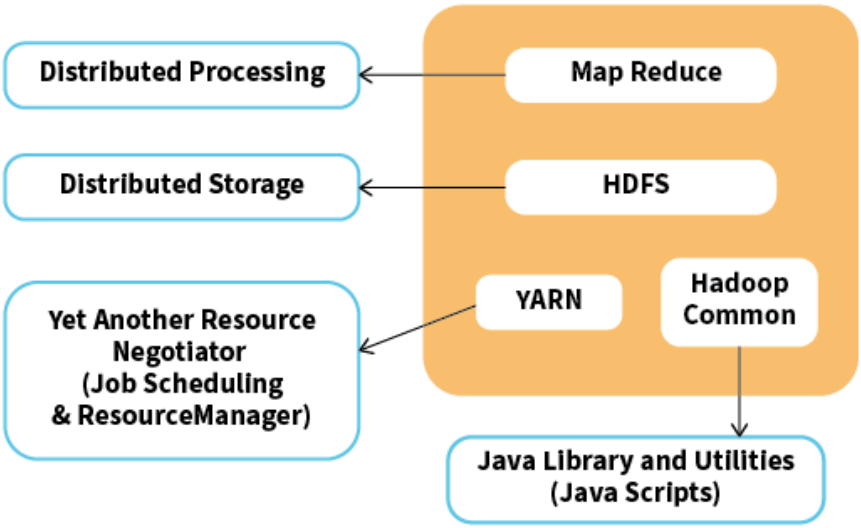
In conclusion, Microsoft's use of big data exemplified in the Azure ecosystem demonstrates the company's commitment to using data to create updated content. Although this technology has the potential to provide powerful and robust analysis, issues such as complexity and security need to be carefully managed to achieve good results. Overall, Microsoft's big data strategy has helped bolster its leadership in technology.

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	02

AIM: Install Hadoop on a Single Node Cluster.

WHAT IS HADOOP & WHY IS IT IMPORTANT?

- Hadoop is an open-source software programming framework for storing a large amount of data and performing the computation. Its framework is based on Java programming with some native code in C and shell scripts.
- Hadoop is an open-source software framework that is used for storing and processing large amounts of data in a distributed computing environment. It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets.
- Hadoop has two main components:
    - HDFS (Hadoop Distributed File System): This is the storage component of Hadoop, which allows for the storage of large amounts of data across multiple machines. It is designed to work with commodity hardware, which makes it cost-effective.
    - YARN (Yet Another Resource Negotiator): This is the resource management component of Hadoop, which manages the allocation of resources (such as CPU and memory) for processing the data stored in HDFS.
    - Hadoop also includes several additional modules that provide additional functionality, such as Hive (a SQL-like query language), Pig (a high-level platform for creating MapReduce programs), and HBase (a non-relational, distributed database).
    - Hadoop is commonly used in big data scenarios such as data warehousing, business intelligence, and machine learning. It's also used for data processing, data analysis, and data mining. It enables the distributed processing of large data sets across clusters of computers using a simple programming model.
- Hadoop is important as one of the primary tools to store and process huge amounts of data quickly. It does this by using a distributed computing model which enables the fast processing of data that can be rapidly scaled by adding computing nodes.
- Hadoop Architecture
    - Hadoop stands as a robust platform for storing and processing vast amounts of data. It serves as a key solution for storing and analysing data from diverse sources, including databases, web servers, and file systems.
    - Built on the MapReduce programming algorithm, Hadoop architecture comprises four key components, each playing a crucial role in managing and processing extensive datasets.
        - HDFS (Hadoop Distributed File System)
        - MapReduce
        - YARN (Yet Another Resource Negotiator)
        - Common Utilities or Hadoop Common



INSTALLATION:

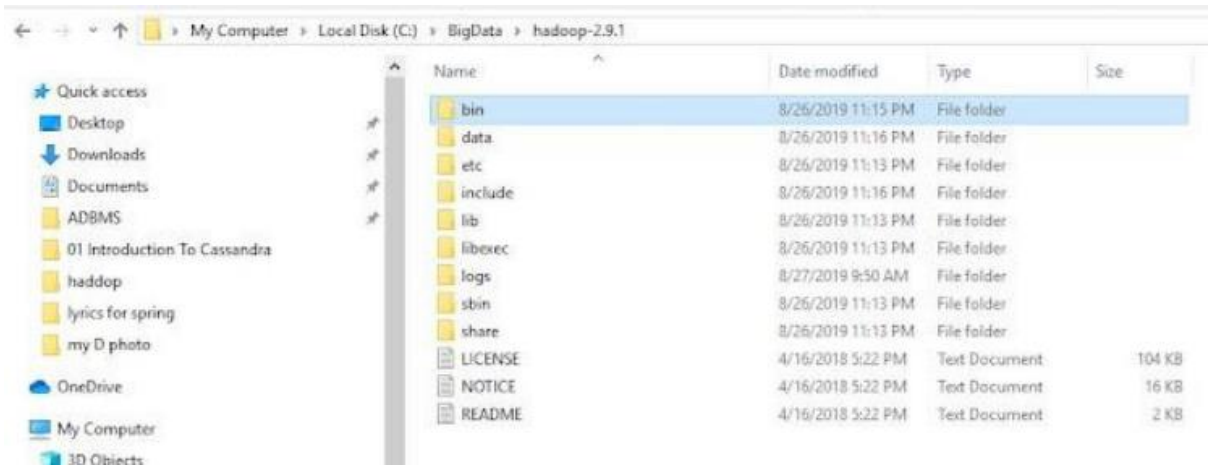
Install Hadoop 2.9.1 on Windows 10

First download the Hadoop 2.9.1 from the below link.

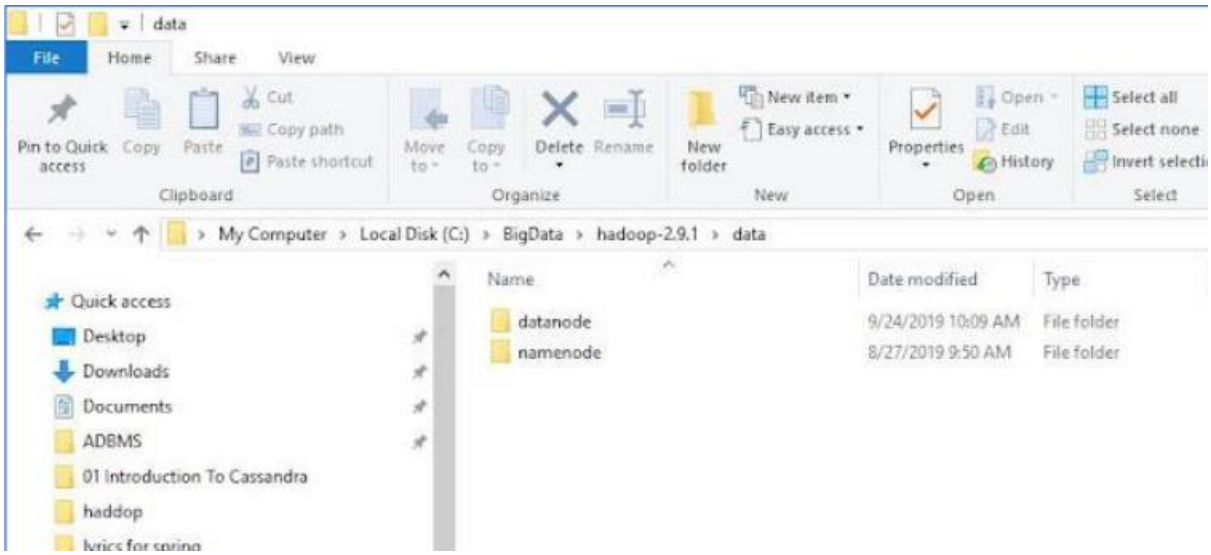
https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.9.1/hadoop-2.9.1.tar.gz

Create a folder path as below and copy the downloaded msi into this folder.

Path:- 'C:/BigData/hadoop-2.9.1'



Go to C:/BigData/3adoop-2.9.1 and create a folder ‘data’. Inside the ‘data’ folder create two folders ‘datanode’ and ‘namenode’.



Then Set Hadoop Environment Variables

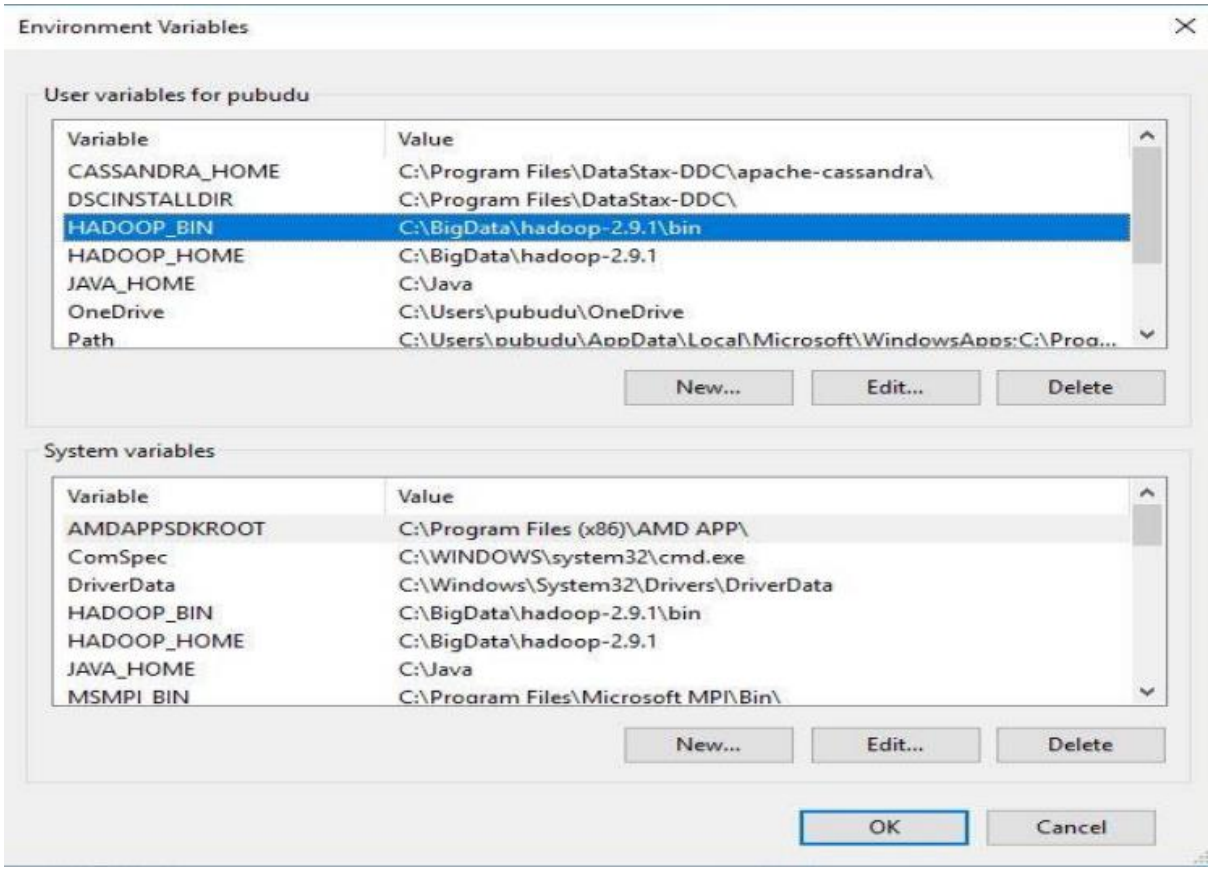
HADOOP_HOME=”C:\BigData\hadoop-2.9.1”

HADOOP_BIN=”C:\BigData\hadoop-2.9.1\bin”

JAVA_HOME=<JDK installation location>”

To set these variables, go to My Computer or This PC. Right click --> Properties --> Advanced

System settings --> Environment variables. Click New to create a new environment variables

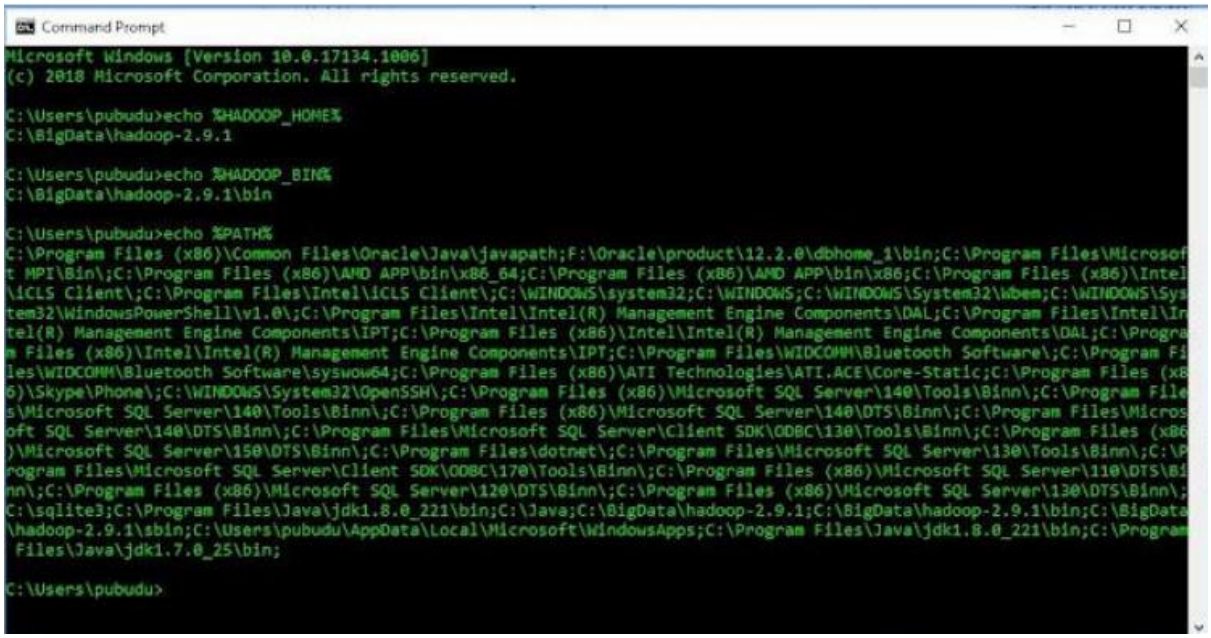


To validate the above setting, open new cmd and check the output.

echo %HADOOP_HOME%

echo %HADOOP_BIN%

echo %PATH%



To configure the Hadoop on windows we have to edit below mention files in the extracted location.

1. hadoop-env.cmd

2. core-site.xml

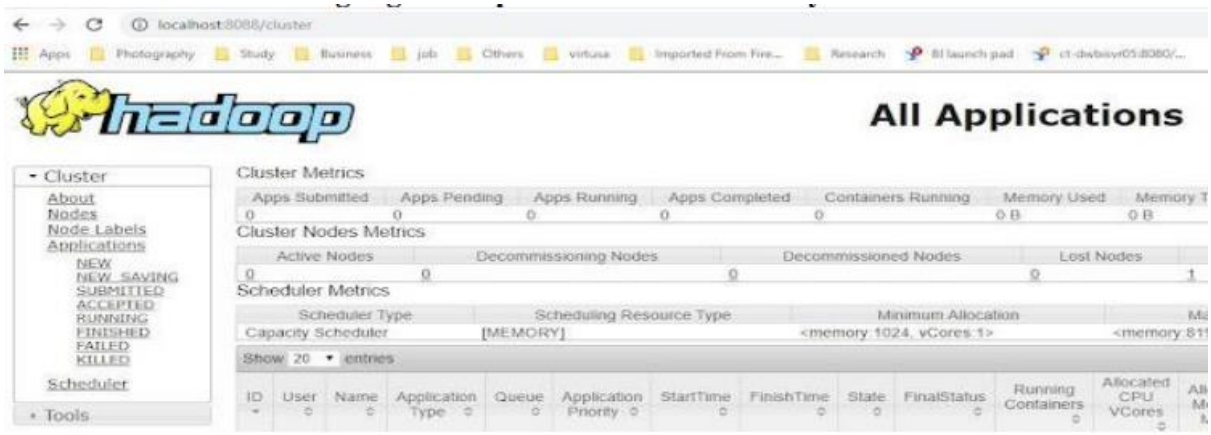
3. hdfs-site.xml

4. mapred-site.xml

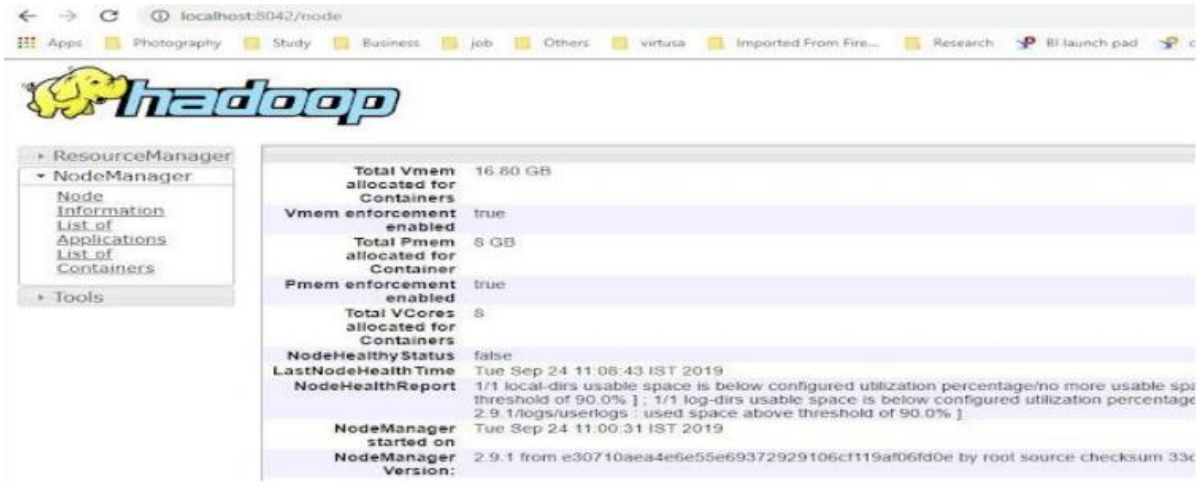
5. yarn-site.xml

Now you can access all the Hadoop components via web urls.

To access Resource Manager go to http://localhost:8088 from your web browser.

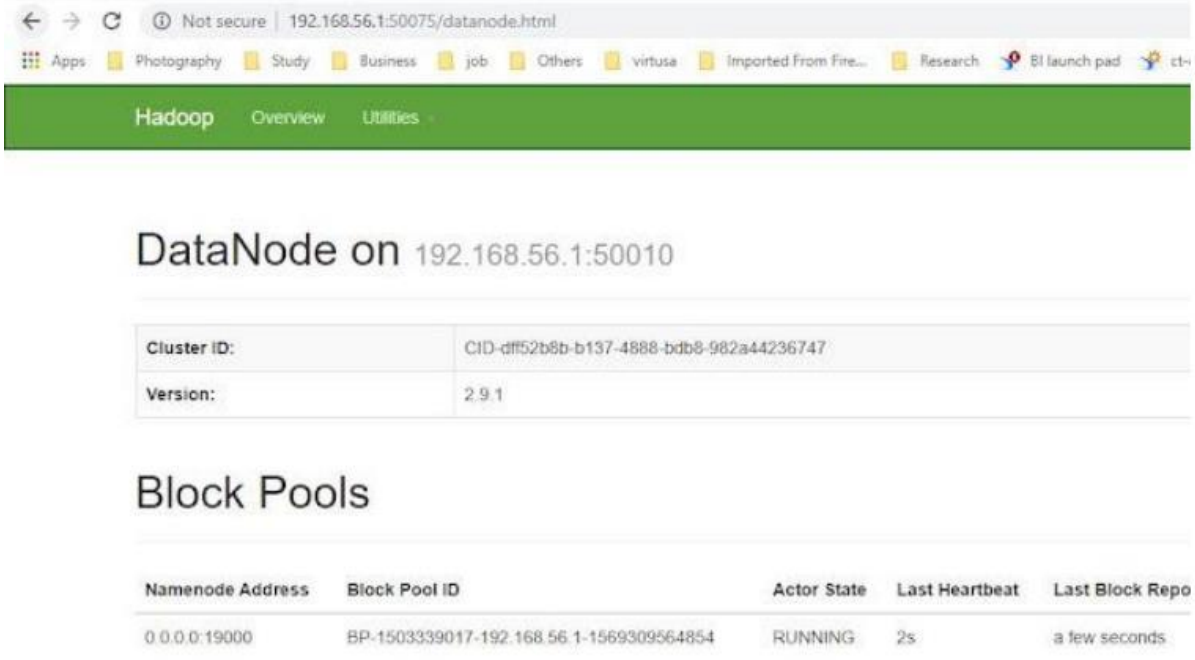


To access Node Manager go to http://localhost:8042 from your web browser.



To access Name Node go to http://localhost:50070 from your web browser

To access Data Node go to http://localhost:50075 from your web browser.



CONCLUSION: Hence, we successfully installed Hadoop

DATE:

Name: Prerna Sunil Jadhav
Sap ID: 60004220127
Batch : C22
Course: Big Data Infrastructure laboratory.
Course Code: DJ19CEEL6011

EXPERIMENT 03

AIM: Execute different HDFS commands.

THEORY:

→ The Hadoop Distributed file system (HDFS) is a distributed file system designed to run on commodity hardware.
→ It has many similarities with existing distributed file system. However, the differences from other distributed file systems are significant.
→ HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware.
→ HDFS provide high throughput access to application data and is suitable for the application that have large datasets.
→ HDFS relaxes a few POSIX requirements to enable streaming access to file system data
→ HDFS was originally built as infrastructure for the Apache Nutch web search engine project.

DATE:

COMMANDS:

1) 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.
⇒ hdfs dfs -ls /

2) mkdir:
To create a directory. In Hadoop dfs there is no home directory by default.
⇒ hdfs dfs -mkdir /user

3) touchz:
It creates an empty file.
⇒ hdfs dfs -touchz /user/myfile.txt

4) cat:
To print the file content
⇒ hdfs dfs -cat /user/sample.txt

5) cp:
This command is used to copy files within hdfs.
⇒ hdfs dfs -cp /user /user_copied.

6) mv:
This command is used to move files within hdfs
⇒ hdfs dfs -mv /user/myfile.txt /user_copied

7) rmr:

This command deletes a file from HDFS recursively. It is very useful command when you want to delete non-empty directory

=> hdfs dfs -rmr /user-copied

8) du:

It will give the given size of each file in the directory

=> hdfs dfs -du /user

9) dus:

This command will give the total size of the directory / file

=> hdfs dfs -dus /user

10) stat:

It will give the last modified time of the directory or path.

=> hdfs dfs -stat /user

11) setrep:

This command is used to change the replication factor of a file / directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs core-site.xml).

=> hdfs dfs -setrep -R 4 /user.

DATE:

CONCLUSION:

Thus we have successfully executed different HDFS commands.

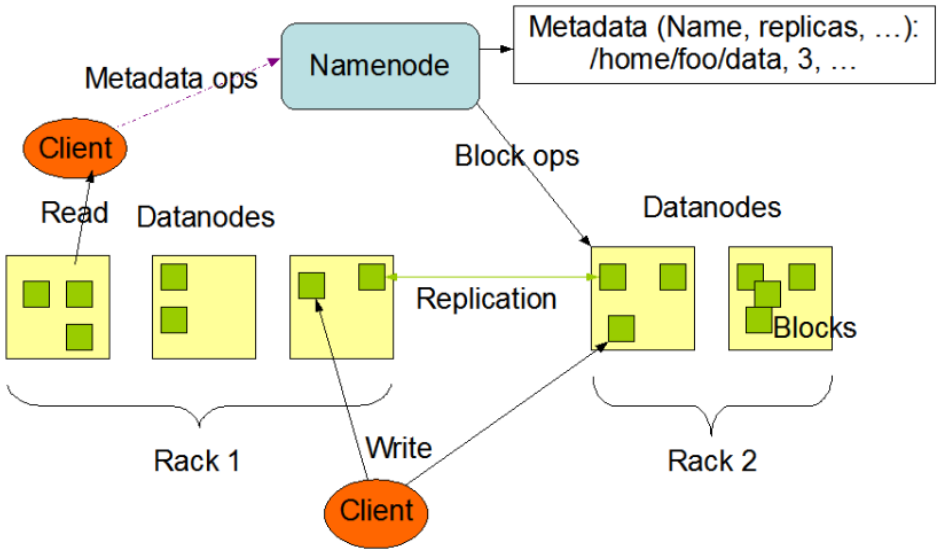
Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	03

AIM: Execute different HDFS Commands.

WHAT IS HDFS?

The Hadoop Distributed File System (HDFS) is a distributed file system 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. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project.

HDFS Architecture:

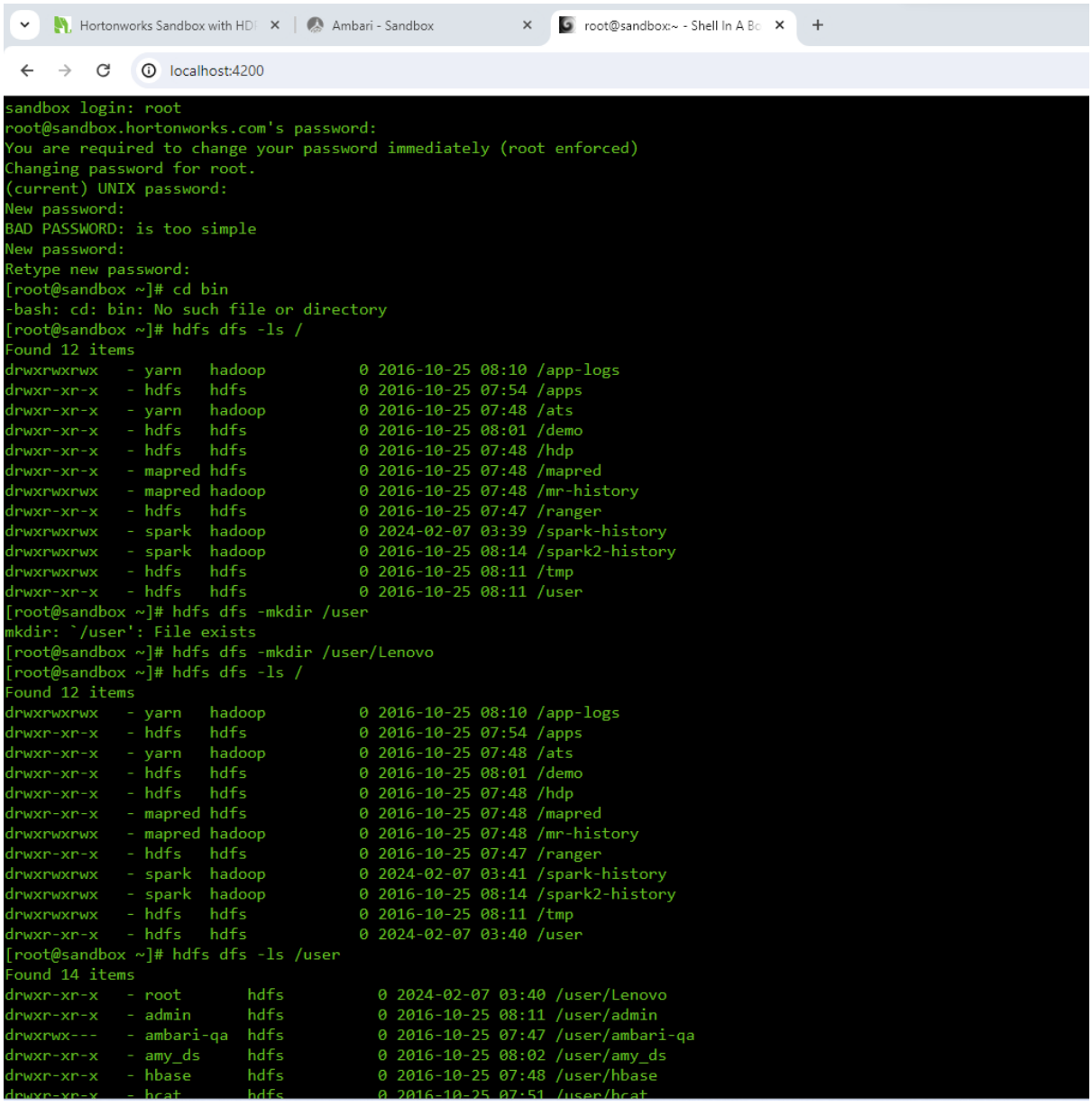


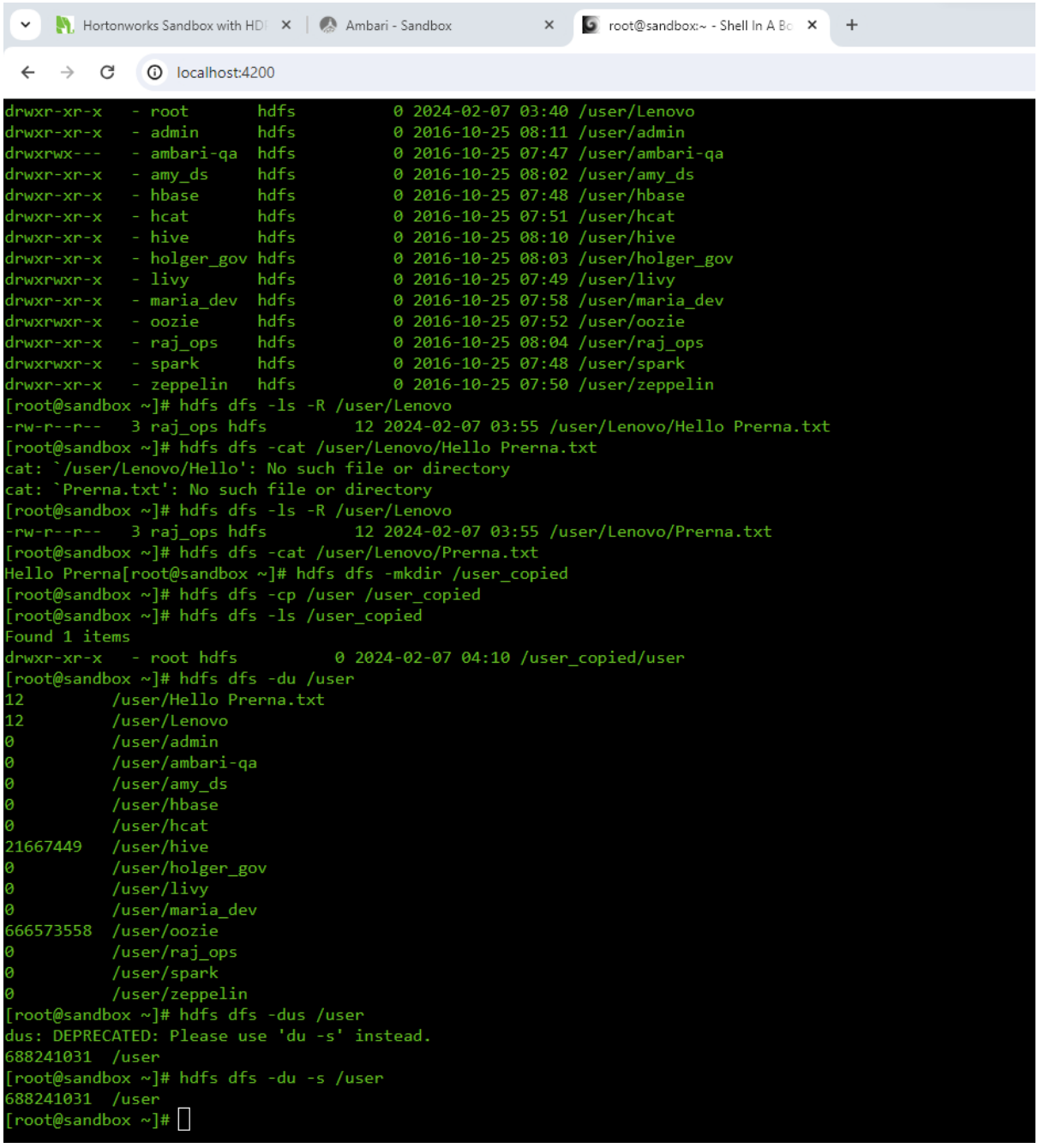
Commands:

- ***hadoop version***: The Hadoop fs shell command version prints the Hadoop version.
- ***hadoop fs –mkdir /path/directory_name:*** This command creates the directory in HDFS if it does not already exist. Use ***hadoop fs mkdir -p /path/directoryname***, so not to fail even if directory exists.
  Note: If the directory already exists in HDFS, then we will get an error message that file already exists.
- ***hadoop fs -ls /path:*** 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.

- ***hadoop fs –cat /path_to_file_in_hdfs:*** The cat command reads the file in HDFS and displays the content of the file on console or stdout.
- ***hadoop fs -mv <src> <dest>:*** The HDFS mv command moves the files or directories from the source to a destination within HDFS.
- ***hadoop fs -cp <src> <dest>:*** The cp command copies a file from one directory to another directory within the HDFS.

Code & Output:





Advantages:

- Distributed data storage, High fault tolerance, Blocks reduce seek time.
- The data is highly available as the same block is present at multiple data-nodes.
- Even if multiple data-nodes are down we can still do our work, thus making it highly reliable.

Limitations: Though HDFS provide many features there are some areas where it doesn't work well.

- Low latency data access
- Small file problem

DATE:

Name: Prerna Sunil Jadhav
Sap ID: 60004220127
Batch: C22
Course: Big Data Infrastructure Laboratory
Course Code: DJ19CEEL6011

EXPERIMENT 04

AIM: Execute HIVE commands to load, insert, retrieve, update or delete data in tables

THEORY:

→ Hive is a data warehouse infrastructure tool to process the structured datat in Hadoop.
→ It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.
→ Initially Hive was developed by facebook, later the Apache Software Foundation took it up and developed it further as an open sourse under the name Apache Hive.
→ It is used by different companies. For Example, Amazon uses it in Amazon Elastic MapReduce
→ features of Hive:
- It stores the schema in database & processed data into HDFS
- It is designed for OLAP
- It is familiar, fast, scalable & extensible.

DATE:

COMMANDS: [root@sandbox ~]# hive

1) show databases;
→ shows databases already existing in Hive.

2) create database student;
→ creates a new database, here, student.

3) use student;
→ using the database for executing queries.

4) create table student (id int, name varchar(20),
branch varchar(20), mobile int)
PARTITIONED by (load_date date)
CLUSTERED by (id) INTO 3 Buckets
STORED as ORC TBLPROPERTIES ('transactional'
='true');
→ creats table in the DB.

5) insert into student partition (load_date=
'2023-03-09') values (101, 'Prema', 'IT', 12346);

6) select * from student;
→ To display contents of table.

7) select id, name from student order by id desc;
→ order by clause (ordering the result
in descending order; by default
ascending order).

DATE:

8) select branch, count (*) from student group by branch;
→ Grouping the result with respect to branch here.

9) update student set name = 'Diksha' where id = 101;
→ updates the value in the table

10) delete from student where name = 'Diksha';
→ Delete the entry from the table.

11) alter table student rename to stud;
→ renaming a table

12) alter table stud add columns (cgpa double);
→ add new column in the table

13) alter table stud change name sname string;
→ Changing the column name with alter statement, also changing the the datatype from varchar to string.

14) select sum(cgpa), max(cgpa), min(cgpa), avg(cgpa), count(cgpa) from stud;
→ performing aggregate functions (sum, max, min, avg, count) on the table.

DATE:

15) Relational operations in Hive

```
select * from stud where cgpa < 9.0;
select * from stud where cgpa >= 7.5;
```

16) Joins in Hive

→ Inner : select * from customers c join orders o on (c.id = o.customer_id);

→ full outer: select * from customer c full outer join order o on (c.id = o.cust_id);

17) Views in Hive

```
create view if not exists customers_vw
as select * from customers where
address = "Mumbai";
```

→ creating view

18) Dropping the view

```
drop view if exists customers_vw;
```

CONCLUSION:

We have successfully executed HIVE Queries using HQL in Hadoop.

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	04

AIM: Execute Hive commands to load, insert, update or delete data in tables.

OUTPUT:

```
sandbox login: root
root@sandbox.hortonworks.com's password:
Last login: Wed Feb  7 03:26:53 2024 from 172.17.0.2
[root@sandbox ~]# hive

Logging initialized using configuration in file:/etc/hive/2.5.0.0-1245/0/hive-log4j.properties
hive> show databases;
OK
default
foodmart
xademo
Time taken: 2.348 seconds, Fetched: 3 row(s)
hive> create database prernaDB;
OK
Time taken: 2.276 seconds
hive> use prernaDB;
OK
Time taken: 0.274 seconds
hive> create table prerna(id int, name varchar(20), branch varchar(20), mobile int)
    > PARTITIONED BY (load_date date)
    > CLUSTERED BY (id) INTO 3 BUCKETS
    > STORED AS ORC TBLPROPERTIES ('TRANSACTIONAL'='true');
OK
Time taken: 1.084 seconds
hive> describe prerna;
OK
id                      int
name                    varchar(20)
branch                  varchar(20)
mobile                  int
load_date               date

# Partition Information
# col_name              data_type               comment

load_date               date
Time taken: 0.604 seconds, Fetched: 10 row(s)
hive> describe prernaDB DEFAULT;
FAILED: SemanticException [Error 10001]: Table not found prernaDB
hive> describe DATABASE prernaDB;
OK
prernadb                hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse/prernadb.db     roo
SER
Time taken: 0.308 seconds, Fetched: 1 row(s)
hive> insert into prerna partition (load_date = '01-01-2024') values (101,'Prerna','Computer',98765
);
FAILED: SemanticException [Error 10248]: Cannot add partition column load_date of type string as it
ot be converted to type date
hive> insert into prerna partition (load_date = 01-01-2024) values (101,'Prerna','Computer',9876543
```

```
hive> insert into prerna partition (load_date = '2024-02-14') values (101,'Prerna','Computer',98765
43210);
Query ID = root_20240214033300_12570c8c-8e3b-4b44-af8a-4168fbaf705d
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 9.16 s
--------------------------------------------------------------------------------
Loading data to table prernadb.prerna partition (load_date=2024-02-14)
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=1, numRows=0, totalSize=839, rawDa
taSize=0]
OK
Time taken: 23.035 seconds
hive> insert into prerna partition (load_date = '2024-02-14') values (102,'Akshata','EXTC',99988877
7666);
Query ID = root_20240214033428_a06736de-b8a0-47cb-8721-827ff194d39e
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 5.15 s
--------------------------------------------------------------------------------
Loading data to table prernadb.prerna partition (load_date=2024-02-14)
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=2, numRows=0, totalSize=1669, rawD
ataSize=0]
OK
Time taken: 8.092 seconds
hive> insert into prerna partition (load_date = '2024-02-14') values (103,'Diksha','Mechanical',816
39852);
Query ID = root_20240214033523_59c07c0e-94e6-441c-a9e7-bd246c9870e5
Total jobs = 1
Launching Job 1 out of 1
```

```
----------------------------------------------------------------------------------
       VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 3.50 s
----------------------------------------------------------------------------------
Loading data to table prernadb.prerna partition (load_date=2024-02-14)
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=3, numRows=0, totalSize=2545, rawD
ataSize=0]
OK
Time taken: 5.912 seconds
hive> select * from prerna;
OK
101     Prerna  Computer        NULL    2024-02-14
102     Akshata EXTC    NULL    2024-02-14
103     Diksha  Mechanical      81639852        2024-02-14
Time taken: 0.487 seconds, Fetched: 3 row(s)
hive> select id, name from prerna order by id desc;
Query ID = root_20240214033840_1c1aaee1-09fb-48f6-9c1a-585d9756a3b6
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

----------------------------------------------------------------------------------
       VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 28.00 s
----------------------------------------------------------------------------------
OK
103     Diksha
102     Akshata
101     Prerna
Time taken: 29.595 seconds, Fetched: 3 row(s)
hive> insert into prerna partition (load_date = '2024-02-14') values (104,'Yash','EXTC',816392);
Query ID = root_20240214034312_87ad898b-9224-4e40-b42a-a1a0ea30b868
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)
```

```
----------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 6.61 s
----------------------------------------------------------------------------------
Loading data to table prernadb.prerna partition (load_date=2024-02-14)
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=4, numRows=0, totalSize=3364, rawD
ataSize=0]
OK
Time taken: 9.346 seconds
hive> insert into prerna partition (load_date = '2024-02-14') values (105,'Krishna','Mechanical',85
852);
Query ID = root_20240214034353_07eb7d27-d08a-44d6-b007-25a25e9a6609
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

----------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 2.90 s
----------------------------------------------------------------------------------
Loading data to table prernadb.prerna partition (load_date=2024-02-14)
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=5, numRows=0, totalSize=4236, rawD
ataSize=0]
OK
Time taken: 5.408 seconds
hive> insert into prerna partition (load_date = '2024-02-14') values (106,'Kajal','Chemical',84555)
;
Query ID = root_20240214034438_bd92ae2e-dbbf-40f5-b064-69ea0dc9613c
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

----------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 2.25 s
----------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
Loading data to table prernadb.prerna partition (load_date=2024-02-14)
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=6, numRows=0, totalSize=5095, rawD
ataSize=0]
OK
Time taken: 4.007 seconds
hive> select * from prerna;
OK
101     Prerna  Computer        NULL    2024-02-14
102     Akshata EXTC    NULL    2024-02-14
103     Diksha  Mechanical      81639852        2024-02-14
104     Yash    EXTC    816392  2024-02-14
105     Krishna Mechanical      85852   2024-02-14
106     Kajal   Chemical        84555   2024-02-14
Time taken: 0.166 seconds, Fetched: 6 row(s)
hive> SELECT branch, count(*) FROM prerna GROUP BY branch;
Query ID = root_20240214034613_bd0b8fd4-fbfa-461a-842e-e02c1e834dca
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 12.66 s
--------------------------------------------------------------------------------
OK
Chemical        1
Computer        1
EXTC    2
Mechanical      2
Time taken: 14.684 seconds, Fetched: 4 row(s)
hive> update prerna set mobile=888552 where id=101;
Query ID = root_20240214035000_29c24cba-49cd-47b3-8a8e-f03d592c0bfd
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
```

```
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=7, numRows=0, totalSize=5965, rawD
ataSize=0]
OK
Time taken: 22.841 seconds
hive> update prerna set mobile=888552 where id=102;
Query ID = root_20240214035201_1260ea37-75c3-419e-8f0e-8957f8d430f4
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 11.20 s
----------------------------------------------------------------------------------------------
Loading data to table prernadb.prerna partition (load_date=null)
         Time taken for load dynamic partitions : 699
        Loading partition {load_date=2024-02-14}
         Time taken for adding to write entity : 0
Partition prernadb.prerna{load_date=2024-02-14} stats: [numFiles=8, numRows=0, totalSize=6836, rawD
ataSize=0]
OK
Time taken: 14.194 seconds
hive> select * from prerna;
OK
101     Prerna  Computer        888552  2024-02-14
102     Akshata EXTC    888552  2024-02-14
103     Diksha  Mechanical      81639852        2024-02-14
104     Yash    EXTC    816392  2024-02-14
105     Krishna Mechanical      85852   2024-02-14
106     Kajal   Chemical        84555   2024-02-14
Time taken: 0.187 seconds, Fetched: 6 row(s)
hive> delete from prerna where name='Prerna';
Query ID = root_20240214035417_f936f3cd-97b7-4415-9784-769bff40fe42
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

----------------------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
```

```
hive> select * from prerna;
OK
102     Akshata EXTC    888552  2024-02-14
103     Diksha  Mechanical      81639852        2024-02-14
104     Yash    EXTC    816392  2024-02-14
105     Krishna Mechanical      85852   2024-02-14
106     Kajal   Chemical        84555   2024-02-14
Time taken: 0.163 seconds, Fetched: 5 row(s)
hive> alter table prerna rename to prernaTable
    > ;
OK
Time taken: 0.574 seconds
hive> show tables;
OK
prernatable
values__tmp__table__1
values__tmp__table__2
values__tmp__table__3
values__tmp__table__4
values__tmp__table__5
values__tmp__table__6
values__tmp__table__7
values__tmp__table__8
values__tmp__table__9
Time taken: 0.16 seconds, Fetched: 10 row(s)
hive> alter table prernaTable add columns(cgpa double);
OK
Time taken: 0.294 seconds
hive> describe prernaTable;
OK
id                      int
name                    varchar(20)
branch                  varchar(20)
mobile                  int
cgpa                    double
load_date               date

# Partition Information
# col_name              data_type               comment

load_date               date
Time taken: 0.15 seconds, Fetched: 11 row(s)
hive> select * from prernaTable;
OK
102     Akshata EXTC    888552  NULL    2024-02-14
103     Diksha  Mechanical      81639852        NULL    2024-02-14
104     Yash    EXTC    816392  NULL    2024-02-14
105     Krishna Mechanical      85852   NULL    2024-02-14
106     Kajal   Chemical        84555   NULL    2024-02-14
```

```
hive> alter table prernaTable CHANGE name sname String;
OK
Time taken: 0.351 seconds
hive> update prernaTable set cgpa=9.3 where id=102;
Query ID = root_20240214035910_5623fb72-56df-46d1-81b8-40abe35bca95
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 18.41 s
--------------------------------------------------------------------------------
Loading data to table prernadb.prernatable partition (load_date=null)
         Time taken for load dynamic partitions : 249
        Loading partition {load_date=2024-02-14}
         Time taken for adding to write entity : 0
Partition prernadb.prernatable{load_date=2024-02-14} stats: [numFiles=10, numRows=0, totalSize=8327
, rawDataSize=0]
OK
Time taken: 21.172 seconds
hive> select * from prernaTable;
OK
102     Akshata EXTC    888552  9.3     2024-02-14
103     Diksha  Mechanical      81639852        NULL    2024-02-14
104     Yash    EXTC    816392  NULL    2024-02-14
105     Krishna Mechanical      85852   NULL    2024-02-14
106     Kajal   Chemical        84555   NULL    2024-02-14
Time taken: 0.603 seconds, Fetched: 5 row(s)
hive> select sum(cgpa) from prernaTable;
Query ID = root_20240214040100_4181bffb-a65b-46ab-be9a-38edd994780d
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
```

```
hive> alter table prernaTable CHANGE name sname String;
OK
Time taken: 0.351 seconds
hive> update prernaTable set cgpa=9.3 where id=102;
Query ID = root_20240214035910_5623fb72-56df-46d1-81b8-40abe35bca95
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 18.41 s
--------------------------------------------------------------------------------
Loading data to table prernadb.prernatable partition (load_date=null)
         Time taken for load dynamic partitions : 249
        Loading partition {load_date=2024-02-14}
         Time taken for adding to write entity : 0
Partition prernadb.prernatable{load_date=2024-02-14} stats: [numFiles=10, numRows=0, totalSize=8327
, rawDataSize=0]
OK
Time taken: 21.172 seconds
hive> select * from prernaTable;
OK
102     Akshata EXTC    888552  9.3     2024-02-14
103     Diksha  Mechanical      81639852        NULL    2024-02-14
104     Yash    EXTC    816392  NULL    2024-02-14
105     Krishna Mechanical      85852   NULL    2024-02-14
106     Kajal   Chemical        84555   NULL    2024-02-14
Time taken: 0.603 seconds, Fetched: 5 row(s)
hive> select sum(cgpa) from prernaTable;
Query ID = root_20240214040100_4181bffb-a65b-46ab-be9a-38edd994780d
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
```

```
hive> alter table prernaTable CHANGE name sname String;
OK
Time taken: 0.351 seconds
hive> update prernaTable set cgpa=9.3 where id=102;
Query ID = root_20240214035910_5623fb72-56df-46d1-81b8-40abe35bca95
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 18.41 s
--------------------------------------------------------------------------------
Loading data to table prernadb.prernatable partition (load_date=null)
         Time taken for load dynamic partitions : 249
        Loading partition {load_date=2024-02-14}
         Time taken for adding to write entity : 0
Partition prernadb.prernatable{load_date=2024-02-14} stats: [numFiles=10, numRows=0, totalSize=8327
, rawDataSize=0]
OK
Time taken: 21.172 seconds
hive> select * from prernaTable;
OK
102     Akshata EXTC    888552  9.3     2024-02-14
103     Diksha  Mechanical      81639852        NULL    2024-02-14
104     Yash    EXTC    816392  NULL    2024-02-14
105     Krishna Mechanical      85852   NULL    2024-02-14
106     Kajal   Chemical        84555   NULL    2024-02-14
Time taken: 0.603 seconds, Fetched: 5 row(s)
hive> select sum(cgpa) from prernaTable;
Query ID = root_20240214040100_4181bffb-a65b-46ab-be9a-38edd994780d
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
```

```
hive> update prernaTable set cgpa=8.4 where id=103;
Query ID = root_20240214040138_ab9bc15a-55bf-4aa6-a8e1-1324aa93e057
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 8.70 s
--------------------------------------------------------------------------------
Loading data to table prernadb.prernatable partition (load_date=null)
         Time taken for load dynamic partitions : 232
        Loading partition {load_date=2024-02-14}
         Time taken for adding to write entity : 0
Partition prernadb.prernatable{load_date=2024-02-14} stats: [numFiles=11, numRows=0, totalSize=9315
, rawDataSize=0]
OK
Time taken: 10.29 seconds
hive> update prernaTable set cgpa=9.65 where id=106;
Query ID = root_20240214040217_575fc595-0b96-4131-9562-fee04bd1541c
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 9.02 s
--------------------------------------------------------------------------------
Loading data to table prernadb.prernatable partition (load_date=null)
         Time taken for load dynamic partitions : 164
        Loading partition {load_date=2024-02-14}
         Time taken for adding to write entity : 0
Partition prernadb.prernatable{load_date=2024-02-14} stats: [numFiles=12, numRows=0, totalSize=1028
8, rawDataSize=0]
OK
Time taken: 11.294 seconds
```

```
hive> select sum(cgpa) from prernaTable;
Query ID = root_20240214040233_0a24b166-1f88-44d2-8eca-b7110296dc79
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

----------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       2       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 8.86 s
----------------------------------------------------------------------------------
OK
27.35
Time taken: 9.513 seconds, Fetched: 1 row(s)
hive> select max(cgpa),min(cgpa), avg(cgpa) from prernaTable;
Query ID = root_20240214040308_b65aae0a-1aae-4212-9630-885ea4e90d3e
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

----------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.78 s
----------------------------------------------------------------------------------
OK
9.65    8.4     9.116666666666667
Time taken: 7.665 seconds, Fetched: 1 row(s)
hive> select * from prernaTable where cgpa<9;
OK
103     Diksha  Mechanical      81639852        8.4     2024-02-14
Time taken: 0.4 seconds, Fetched: 1 row(s)
hive> select * from prernaTable where cgpa>9;
OK
102     Akshata EXTC    888552  9.3     2024-02-14
106     Kajal   Chemical        84555   9.65    2024-02-14
Time taken: 0.256 seconds, Fetched: 2 row(s)
```

```
hive> select * from prernaTable where cgpa>9;
OK
102     Akshata EXTC    888552  9.3     2024-02-14
106     Kajal   Chemical        84555   9.65    2024-02-14
Time taken: 0.256 seconds, Fetched: 2 row(s)
hive> create table customers(id int, name String, age int, address String, salary int)
    > partitioned by (load_date date)
    > clustered by (id) into 3 buckets
    > stored as orc tblproperties ('transactional'='true');
OK
Time taken: 0.418 seconds
```

```
hive> insert into customers partition (load_date = '2024-02-14') values (1,"Ross",25,"Mumbai",25000
);
Query ID = root_20240214041011_538486b4-221d-4a0c-9ac0-e5cc95c6b26e
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 4.01 s
--------------------------------------------------------------------------------
Loading data to table prernadb.customers partition (load_date=2024-02-14)
Partition prernadb.customers{load_date=2024-02-14} stats: [numFiles=1, numRows=0, totalSize=864, ra
wDataSize=0]
OK
Time taken: 8.401 seconds
hive> insert into customers partition (load_date = '2024-02-14') values (2,"Mike",27,"Bhopal",35000
);
Query ID = root_20240214041109_7c8069e3-274d-4682-b846-d66574964db2
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 2.95 s
--------------------------------------------------------------------------------
Loading data to table prernadb.customers partition (load_date=2024-02-14)
Partition prernadb.customers{load_date=2024-02-14} stats: [numFiles=2, numRows=0, totalSize=1735, r
awDataSize=0]
OK
Time taken: 4.776 seconds
hive> insert into customers partition (load_date = '2024-02-14') values(3,"Albin",24,"Pune",50000);

Query ID = root_20240214041218_76da6ad6-be77-4eed-8cc1-ad6d89100add
Total jobs = 1
Launching Job 1 out of 1
```

```
Loading data to table prernadb.customers partition (load_date=2024-02-14)
Partition prernadb.customers{load_date=2024-02-14} stats: [numFiles=3, numRows=0, totalSize=2599, r
awDataSize=0]
OK
Time taken: 5.092 seconds
hive>  select * from customers;
OK
1       Ross    25      Mumbai  25000   2024-02-14
2       Mike    27      Bhopal  35000   2024-02-14
3       Albin   24      Pune    50000   2024-02-14
Time taken: 0.207 seconds, Fetched: 3 row(s)
hive> create table orders(oid int, customer_id int, amount int);
OK
Time taken: 0.305 seconds
hive>  insert into orders values(101, 2, 20000);
Query ID = root_20240214041336_181bc368-0c3d-478a-8f48-6e5fc61b814f
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 3.13 s
--------------------------------------------------------------------------------
Loading data to table prernadb.orders
Table prernadb.orders stats: [numFiles=1, numRows=1, totalSize=12, rawDataSize=11]
OK
Time taken: 3.995 seconds
hive>  insert into orders values(102, 2, 25000);
Query ID = root_20240214041344_bbf57554-0785-4cf2-a53d-f63f272048fc
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 0.41 s
--------------------------------------------------------------------------------
```

```
Loading data to table prernadb.orders
Table prernadb.orders stats: [numFiles=2, numRows=2, totalSize=24, rawDataSize=22]
OK
Time taken: 1.153 seconds
hive> insert into orders values(104,4,10000);
Query ID = root_20240214041358_37fc63e6-e6ad-4791-9fc7-da25306e28c7
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 0.40 s
--------------------------------------------------------------------------------
Loading data to table prernadb.orders
Table prernadb.orders stats: [numFiles=3, numRows=3, totalSize=36, rawDataSize=33]
OK
Time taken: 1.545 seconds
hive> select * from orders;
OK
101     2       20000
102     2       25000
104     4       10000
Time taken: 0.149 seconds, Fetched: 3 row(s)
hive> select * from customers c join orders o on (c.id = o.customer_id);
Query ID = root_20240214041503_be050749-b1aa-463a-944c-436f82383c07
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Map 2 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 9.06 s
--------------------------------------------------------------------------------
OK
2       Mike    27      Bhopal  35000   2024-02-14      101     2       20000
2       Mike    27      Bhopal  35000   2024-02-14      102     2       25000
Time taken: 10.074 seconds, Fetched: 2 row(s)
```

```
hive> select * from customers c left outer join orders o on (c.id = o.customer_id);
Query ID = root_20240214041536_d89dfa9f-9462-4923-8d02-7d8e8b8ab3a4
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Map 2 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.58 s
--------------------------------------------------------------------------------
OK
1       Ross    25      Mumbai  25000   2024-02-14      NULL    NULL    NULL
3       Albin   24      Pune    50000   2024-02-14      NULL    NULL    NULL
2       Mike    27      Bhopal  35000   2024-02-14      101     2       20000
2       Mike    27      Bhopal  35000   2024-02-14      102     2       25000
Time taken: 7.505 seconds, Fetched: 4 row(s)
hive> select * from customers c right outer join orders o on (c.id = o.customer_id);
Query ID = root_20240214041618_8eda5f4f-db8b-4109-b53b-da7d25e646e4
Total jobs = 1
Launching Job 1 out of 1


Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       0       0
Map 2 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.68 s
--------------------------------------------------------------------------------
OK
2       Mike    27      Bhopal  35000   2024-02-14      101     2       20000
2       Mike    27      Bhopal  35000   2024-02-14      102     2       25000
NULL    NULL    NULL    NULL    NULL    NULL    104     4       10000
Time taken: 9.036 seconds, Fetched: 3 row(s)
hive> select * from customers c full outer join orders o on (c.id = o.customer_id);
Query ID = root_20240214041637_8c91b21b-62d9-47f2-89de-66cd979517f3
Total jobs = 1
Launching Job 1 out of 1
```

```
Status: Running (Executing on YARN cluster with App id application_1707880073946_0002)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      3          3        0        0       3       0
Map 3 ..........   SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 6.01 s
--------------------------------------------------------------------------------
OK
1       Ross    25      Mumbai  25000   2024-02-14      NULL    NULL    NULL
2       Mike    27      Bhopal  35000   2024-02-14      101     2       20000
2       Mike    27      Bhopal  35000   2024-02-14      102     2       25000
3       Albin   24      Pune    50000   2024-02-14      NULL    NULL    NULL
NULL    NULL    NULL    NULL    NULL    NULL    104     4       10000
Time taken: 6.902 seconds, Fetched: 5 row(s)
hive> create view if not exists customers_vw as select * from customers where address="Mumbai";
OK
Time taken: 0.605 seconds
hive> select * from customers_vw;
OK
1       Ross    25      Mumbai  25000   2024-02-14
Time taken: 0.271 seconds, Fetched: 1 row(s)
hive> alter view customers_vw as select * from customers;
OK
Time taken: 0.242 seconds
hive> select * from customers_vw;
OK
1       Ross    25      Mumbai  25000   2024-02-14
2       Mike    27      Bhopal  35000   2024-02-14
3       Albin   24      Pune    50000   2024-02-14
Time taken: 0.141 seconds, Fetched: 3 row(s)
hive>  drop view if exists customers_vw;
OK
Time taken: 0.505 seconds
hive> select * from customers_vw;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'customers_vw'
hive>  drop view if exists customers_vw;
```

DATE:

Name: Prerna Sunil Jadhav
Sap ID: 60004220127
Batch: C22
Course: Big Data Infrastructure Laboratory.
Course code: DJ19CEEL6011

EXPERIMENT 05

AIM: To implement commands in HBase shell scripting.

THEORY:

→ HBase is a data model that is similar to Google's big table.
→ It is an open source, distributed database developed by Apache software foundation written in Java.
→ HBase is an essential part of our Hadoop Ecosystem.
→ HBase runs on top of HDFS (Hadoop Distributed File system).
→ It can store massive amounts of data from tetrabytes to petabytes.
→ It is column oriented & horizontally scalable
→ Features:
1) It provides easy to use Java API for client access
2) HBase provides consistent read and writes.
3) It doesn't enforce relationship within your data.

DATE:

COMMANDS: (hbase shell)

1) CREATE

The 'create' command is used to create table within HBase Shell.

=> create 'product', 'shoe', 'tshirt'

2) LIST

It is used to list all tables

=> list

3) PUT

Add or update only one column

=> put 'product', '1', 'shoe:size', '28'

4) SCAN

It applies to all rowkey & in turn all column within specified table

=> scan 'product' => scan 'product', {limit=>1}

5) GET

It is used to fetch data associated with a particular row key.

=> get 'product', '1'

6) DELETE TABLE

first we need to disable table then drop it.

=> disable 'product'

=> drop 'product'

CONCLUSION: Hence we implemented the above commands

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	05

AIM: Execute HBase Commands.

OUTPUT:

```
sandbox login: root
root@sandbox.hortonworks.com's password:
Last login: Wed Feb 14 03:11:34 2024 from 172.17.0.2
[root@sandbox ~]# hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.1.2.2.5.0.0-1245, r53538b8ab6749cbb6fdc0fe448b89aa82495fb3f, Fri Aug 26 01:32:27 UTC 2016

hbase(main):001:0> create 'product','shoe','tshirt'
0 row(s) in 3.5020 seconds

=> Hbase::Table - product
hbase(main):002:0> list
TABLE
ATLAS_ENTITY_AUDIT_EVENTS
atlas_titan
iemployee
product
4 row(s) in 0.0570 seconds

=> ["ATLAS_ENTITY_AUDIT_EVENTS", "atlas_titan", "iemployee", "product"]
```

```
hbase(main):003:0> put 'product', '1', 'shoe:title', 'Adidas Shoe'
0 row(s) in 0.3420 seconds

hbase(main):004:0> put 'product', '1', 'shoe:description', 'Running Shoes For Men'
0 row(s) in 0.1120 seconds

hbase(main):005:0> put 'product', '1', 'shoe:color', 'black'
0 row(s) in 0.0830 seconds

hbase(main):006:0> put 'product', '1', 'shoe:price', '40'
0 row(s) in 0.0270 seconds

hbase(main):007:0> put 'product', '1', 'shoe:currency', 'USD'
0 row(s) in 0.0220 seconds

hbase(main):008:0> put 'product', '1', 'shoe:size', '28'
0 row(s) in 0.0220 seconds

hbase(main):009:0> put 'product', '2', 'tshirt:title', 'Puma Tshirt'
0 row(s) in 0.0290 seconds

hbase(main):010:0> put 'product', '2', 'tshirt:description', 'Graphic Print Men Round Neck Pink T-Shir
0 row(s) in 0.0190 seconds

hbase(main):011:0> put 'product', '2', 'tshirt:color', 'pink'
0 row(s) in 0.0200 seconds

hbase(main):012:0> put 'product', '2', 'tshirt:price', '20'
0 row(s) in 0.0210 seconds

hbase(main):013:0> put 'product', '2', 'tshirt:currency', 'USD'
0 row(s) in 0.0220 seconds

hbase(main):014:0> put 'product', '2', 'tshirt:size', 'M'
0 row(s) in 0.0250 seconds
```

```
hbase(main):015:0> get 'product','1'
COLUMN                          CELL
 shoe:color                     timestamp=1708484818137, value=black
 shoe:currency                  timestamp=1708484849126, value=USD
 shoe:description               timestamp=1708484809517, value=Running Shoes For Men
 shoe:price                     timestamp=1708484827894, value=40
 shoe:size                      timestamp=1708484861763, value=28
 shoe:title                     timestamp=1708484801101, value=Adidas Shoe
6 row(s) in 0.1350 seconds

hbase(main):016:0> get 'product','2'
COLUMN                          CELL
 tshirt:color                   timestamp=1708484896191, value=pink
 tshirt:currency                timestamp=1708484929185, value=USD
 tshirt:description             timestamp=1708484886158, value=Graphic Print Men Round Neck Pink T-Shirt
 tshirt:price                   timestamp=1708484906505, value=20
 tshirt:size                    timestamp=1708484944594, value=M
 tshirt:title                   timestamp=1708484874200, value=Puma Tshirt
6 row(s) in 0.0180 seconds

hbase(main):017:0> put 'product','1','shoe:color','white'
0 row(s) in 0.0250 seconds

hbase(main):018:0> get 'product','1'
COLUMN                          CELL
 shoe:color                     timestamp=1708485037341, value=white
 shoe:currency                  timestamp=1708484849126, value=USD
 shoe:description               timestamp=1708484809517, value=Running Shoes For Men
 shoe:price                     timestamp=1708484827894, value=40
 shoe:size                      timestamp=1708484861763, value=28
 shoe:title                     timestamp=1708484801101, value=Adidas Shoe
6 row(s) in 0.0310 seconds
```

```
hbase(main):019:0> scan 'product'
ROW                         COLUMN+CELL
 1                          column=shoe:color, timestamp=1708485037341, value=white
 1                          column=shoe:currency, timestamp=1708484849126, value=USD
 1                          column=shoe:description, timestamp=1708484809517, value=Running Shoes For
                            n
 1                          column=shoe:price, timestamp=1708484827894, value=40
 1                          column=shoe:size, timestamp=1708484861763, value=28
 1                          column=shoe:title, timestamp=1708484801101, value=Adidas Shoe
 2                          column=tshirt:color, timestamp=1708484896191, value=pink
 2                          column=tshirt:currency, timestamp=1708484929185, value=USD
 2                          column=tshirt:description, timestamp=1708484886158, value=Graphic Print Me
                            Round Neck Pink T-Shirt
 2                          column=tshirt:price, timestamp=1708484906505, value=20
 2                          column=tshirt:size, timestamp=1708484944594, value=M
 2                          column=tshirt:title, timestamp=1708484874200, value=Puma Tshirt
2 row(s) in 0.0740 seconds

hbase(main):020:0> scan 'product',{LIMIT => 1}
ROW                         COLUMN+CELL
 1                          column=shoe:color, timestamp=1708485037341, value=white
 1                          column=shoe:currency, timestamp=1708484849126, value=USD
 1                          column=shoe:description, timestamp=1708484809517, value=Running Shoes For M
                            en
 1                          column=shoe:price, timestamp=1708484827894, value=40
 1                          column=shoe:size, timestamp=1708484861763, value=28
 1                          column=shoe:title, timestamp=1708484801101, value=Adidas Shoe
1 row(s) in 0.0430 seconds

hbase(main):021:0> disable 'product'
0 row(s) in 2.4440 seconds

hbase(main):022:0> drop 'product'
0 row(s) in 1.3160 seconds

hbase(main):023:0>
```

DATE:

Name: Prerna Sunil Jadhav
Sap ID: 60004220127
Batch: C2-2
Course: Big Data Infrastructure Laboratory.
Course Code: DJ19CEEL6011

EXPERIMENT 06

AIM: Implement Matrix Multiplication and word Frequency count using MapReduce

THEORY: MapReduce is a technique in which a huge program is sub-divided into small tasks and run parallely to make computation faster, save time, and mostly used in distributed system.
It has 2 important parts:

- MAPPER: It takes raw data i/p and organizes into key, value pairs.
  For Eg: In a dictionary, you search for the word 'Data' and its ~~o~~ associated meaning is "the facts and statistics collected together for reference or analysis". Here the key is "Data" and the value associated with "its facts and statistics collected together for reference or analysis".
- Reducer: It is responsible for processing data in parallel and produce final output.

DATE:

⇒ MATRIX MULTIPLICATION Using MAP REDUCE

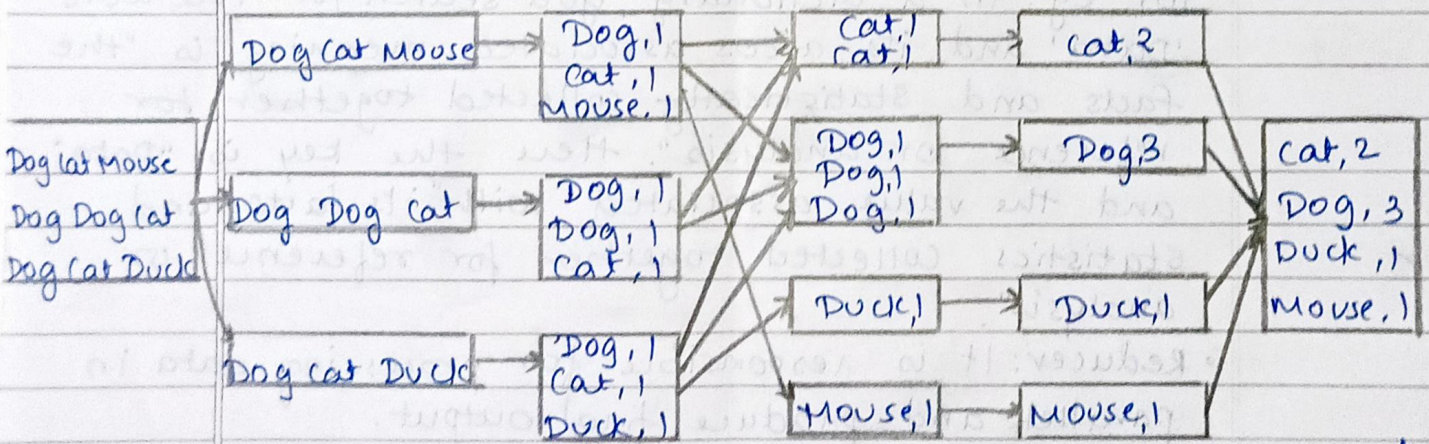
- The Map function:

```
for each element mij of M do
    produce (key,value) pairs as ((i,k),(M,j,mij)),
    for k=1,2,3,... up to the no. of columns of N.
for each element njk of N do
    produce (key,value) pairs as ((i,k),(N,j,njk)),
    for i=1,2,3,... up to the no. of rows of M.
return set of (key,value) pair that each key (i,k) has a
list with values (M,j,mij) & (N,j,njk) for all values of j.
```

- The Reduce function:

```
for each key (i,k) do
    sort values begin with M by j in listM.
    sort value begin with N by j in listN.
    multiply mij and njk for jth value of each list.
    sum up mij * njk
return (i,k), Σ (j=1) mij * njk
```

⇒ ~~MATRIX~~ COUNT FREQUENCY OF WORDS Using MAP REDUCE.



CONCLUSION: Thus, we studied and implement MapReduce

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	06

AIM: Implement Matrix Multiplication and Word Frequency Count using Map Reduce

MATRIX MULTIPLICATION USING MAP REDUCE.

CODE:

```
def matrix_multiply_mapper(matrix_a, matrix_b):
    result = []
    for i in range(len(matrix_a)):
        for j in range(len(matrix_b[0])):
            for k in range(len(matrix_b)):
                result.append(((i, j), (matrix_a[i][k] * matrix_b[k][j])))
    return result

def matrix_multiply_reducer(mapped_result):
    intermediate_result = {}
    for key, value in mapped_result:
        if key in intermediate_result:
            intermediate_result[key].append(value)
        else:
            intermediate_result[key] = [value]

    final_result = []
    for key, values in intermediate_result.items():
        total = sum(values)
        final_result.append((key, total))
    return final_result

def matrix_multiply(matrix_a, matrix_b):
    mapped_result = matrix_multiply_mapper(matrix_a, matrix_b)
    reduced_result = matrix_multiply_reducer(mapped_result)
    final_result = [[0 for _ in range(len(matrix_b[0]))] for _ in
range(len(matrix_a))]

    for key, value in reduced_result:
        i, j = key
        final_result[i][j] = value
```

```
    return final_result

# Example usage:
matrix_a = [[1, 2],
            [3, 4]]

matrix_b = [[5, 6],
            [7, 8]]

result = matrix_multiply(matrix_a, matrix_b)
print(result)
```

OUTPUT:

```
PS C:\Users\Jadhav\Documents\BTech\Docs\6th Sem\BDI\Code> & C:
/msys64/mingw64/bin/python.exe "c:/Users/Jadhav/Documents/BTec
h/Docs/6th Sem/BDI/Code/MatrixMulByMapReduce/Mapper.py"
[[19, 22], [43, 50]]
```

WORD FREQUENCY COUNT USING MAP REDUCE.

CODE:

```
import re

def word_count_mapper(document):
    words = re.findall(r'\w+', document.lower())
    word_count = {}
    for word in words:
        word_count[word] = word_count.get(word, 0) + 1
    return list(word_count.items())

def word_count_reducer(mapped_result):
    intermediate_result = {}
    for item in mapped_result:
        for key, value in item:
            if key in intermediate_result:
                intermediate_result[key] += value
            else:
                intermediate_result[key] = value
    return list(intermediate_result.items())

def word_frequency_count(documents):
```

```
    mapped_result = [word_count_mapper(doc) for doc in documents]
    reduced_result = word_count_reducer(mapped_result)

    final_result = {}
    for key, value in reduced_result:
        final_result[key] = value

    return final_result

# Example usage:
documents = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?"
]

result = word_frequency_count(documents)
print(result)
```

OUTPUT:

```
PS C:\Users\Jadhav\Documents\BTech\Docs\6th Sem\BDI\Code> & C:
/msys64/mingw64/bin/python.exe "c:/Users/Jadhav/Documents/BTec
h/Docs/6th Sem/BDI/Code/MatrixMulByMapReduce/Reducer.py"
{'this': 4, 'is': 4, 'the': 4, 'first': 2, 'document': 4, 'sec
ond': 1, 'and': 1, 'third': 1, 'one': 1}
```

Name: Prerna Sunil Jadhav
Sap ID: 60004220127
Batch: C22
Course: Big Data Infrastructure laboratory
Course code: DJ19CEEL6011

EXPERIMENT 07

AIM: Implement RDD using Python.

THEORY:

SPARK

→ Apache spark is an open-source, distributed processing system used for dig data workloads
→ It utilizes in-memory caching, and optimized query execution for fast analytics queries against data of any size.
→ Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory and 10 times faster when running on disk. This is possible by reducing the no. of read/write operations to disk.
→ Spark not only supports 'Map' and 'Reduce'. It provides development APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads like batch processing, interactive queries, real-time analytics, machine learning & graph processing

RDD (Resilient Distributed Dataset)

→ It is the fundamental data structure of Apache Spark

→ RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.

→ Decomposing the name RDD:

1) Resilient: It is fault-tolerant with the help of RDD lineage graph (DAG) and so able to recompute missing or damaged partitions due to node failure.

2) Distributed: Since Data resides on multiple nodes

3) Dataset: It represents records of the data you work with. The user can data the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.

```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 4.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packag
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=3174884
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

```
from pyspark import SparkContext, SparkConf
import math

# Create a SparkContext
conf = SparkConf().setAppName("SquareRootNumbers").setMaster("local")
sc = SparkContext(conf=conf)

# Create an RDD containing numbers from 1 to 20
numbers_rdd = sc.parallelize(range(1, 21))

# Apply the square root function to each element in the RDD
square_root_rdd = numbers_rdd.map(lambda x: math.sqrt(x))

# Collect and print square roots
square_roots = square_root_rdd.collect()
for square_root in square_roots:
    print(square_root)

# Stop the SparkContext
sc.stop()
```

```
1.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
3.1622776601683795
3.3166247903554
3.4641016151377544
3.605551275463989
3.7416573867739413
3.872983346207417
4.0
4.123105625617661
4.242640687119285
```

```
4.358898943540674
4.47213595499958
```

```
from pyspark import SparkContext, SparkConf

# Initialize Spark
conf = SparkConf().setAppName("ArmstrongNumbers").setMaster("local[*]")
sc = SparkContext(conf=conf)

# Define a function to check if a number is an Armstrong number
def is_armstrong(num):
    order = len(str(num))
    sum = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum += digit ** order
        temp //= 10
    return num == sum

# Create an RDD containing numbers from 100 to 9999
numbers_rdd = sc.parallelize(range(100, 10000))

# Filter the RDD to keep only Armstrong numbers
armstrong_rdd = numbers_rdd.filter(is_armstrong)

# Collect and print the Armstrong numbers
armstrong_numbers = armstrong_rdd.collect()
print("Armstrong numbers between 100 and 9999:", armstrong_numbers)

# Stop Spark
sc.stop()
```

```
Armstrong numbers between 100 and 9999: [153, 370, 371, 407, 1634, 8208, 9474]
```

```
 from pyspark import SparkContext, SparkConf

# Create a SparkContext
conf = SparkConf().setAppName("SquaredNumbers").setMaster("local")
```

```
from pyspark import SparkContext, SparkConf

# Initialize Spark
conf = SparkConf().setAppName("PerfectNumbers").setMaster("local[*]")
sc = SparkContext(conf=conf)

# Define a function to check if a number is a Perfect number
def is_perfect(num):
    sum_divisors = sum([i for i in range(1, num) if num % i == 0])
    return sum_divisors == num

# Create an RDD containing numbers from 1 to 100
numbers_rdd = sc.parallelize(range(1, 101))

# Filter the RDD to keep only Perfect numbers
perfect_rdd = numbers_rdd.filter(is_perfect)

# Collect and print the Perfect numbers
perfect_numbers = perfect_rdd.collect()
print("Perfect numbers between 1 and 100:", perfect_numbers)

# Stop Spark
sc.stop()
```

```
Perfect numbers between 1 and 100: [6, 28]
```

Name: Prerna Sunil Jadhav
Sap Id: 60004220127
Batch: C2-2
Course: Big Data Infrastructure Laboratory
Course code: DJ19CEEL6011

EXPERIMENT 08

AIM: To study and implement SparkQL using PySpark.

THEORY: SparkQL is a query language for RDF data. It is used to retrieve and manipulate stored data in RDF format. SparkQL stands for "SparkQL Protocol and RDF Query language". It was developed by world wide web consontium (W3C) and is a standard for querying RDF data. RDF stands for Resource Desciption framework. It is a standard for describing resources on web. RDF data is stored in triplets, which consists of a subject, a predicate and an object. SparkQL Query is used to retrieve data from a RDF dataset. It consists of a set of patterns that match against RDF data. The patterns are written in a syntax similar to SQL, but with some differences.

Pyspark provides an easy-to-use interface to SparkSQL, allowing users to perform complex data processing tasks with few lines of code with Pyspark. Users can create spark DataFrames, which is similar

to Pandas DataFrame and can be queried using Spark SQL.

CONCLUSION: By studying and implementing SparkQL using PySpark, we leverage the power of SQL queries to effectively analyze and process big data within Spark, enhancing data processing capabilities.

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	08

AIM: To study and Implement SparkQL using PySpark

IRIS DATASET:

CODE:

```
[ ]  from pyspark.sql import SparkSession

     spark = SparkSession.builder.appName("Online IRIS dataset").getOrCreate()

     url = "/content/sample_data/Iris.csv"

     df = spark.read.csv(url, header = False, inferSchema =  True)

     columns = ["id", "sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
     df = df.toDF(*columns)

     df.createOrReplaceTempView("iris_data")

     result1 = spark.sql("Select * from iris_data  WHERE class = 'Iris-setosa'")
     result2 = spark.sql("Select * from iris_data  WHERE sepal_length > 7.0")
     result3 = spark.sql("Select class, COUNT(*) from iris_data group by class ")

     result1.show()
     result2.show()
     result3.show()

     spark.stop()
```

OUTPUT:

```
+---+------------+-----------+------------+-----------+-----------+
| id|sepal_length|sepal_width|petal_length|petal_width|      class|
+---+------------+-----------+------------+-----------+-----------+
|  1|         5.1|        3.5|         1.4|        0.2|Iris-setosa|
|  2|         4.9|        3.0|         1.4|        0.2|Iris-setosa|
|  3|         4.7|        3.2|         1.3|        0.2|Iris-setosa|
|  4|         4.6|        3.1|         1.5|        0.2|Iris-setosa|
|  5|         5.0|        3.6|         1.4|        0.2|Iris-setosa|
|  6|         5.4|        3.9|         1.7|        0.4|Iris-setosa|
|  7|         4.6|        3.4|         1.4|        0.3|Iris-setosa|
|  8|         5.0|        3.4|         1.5|        0.2|Iris-setosa|
|  9|         4.4|        2.9|         1.4|        0.2|Iris-setosa|
| 10|         4.9|        3.1|         1.5|        0.1|Iris-setosa|
| 11|         5.4|        3.7|         1.5|        0.2|Iris-setosa|
| 12|         4.8|        3.4|         1.6|        0.2|Iris-setosa|
| 13|         4.8|        3.0|         1.4|        0.1|Iris-setosa|
| 14|         4.3|        3.0|         1.1|        0.1|Iris-setosa|
| 15|         5.8|        4.0|         1.2|        0.2|Iris-setosa|
| 16|         5.7|        4.4|         1.5|        0.4|Iris-setosa|
| 17|         5.4|        3.9|         1.3|        0.4|Iris-setosa|
| 18|         5.1|        3.5|         1.4|        0.3|Iris-setosa|
| 19|         5.7|        3.8|         1.7|        0.3|Iris-setosa|
| 20|         5.1|        3.8|         1.5|        0.3|Iris-setosa|
+---+------------+-----------+------------+-----------+-----------+
only showing top 20 rows
```

```
+---+------------+-----------+------------+-----------+--------------+
| id|sepal_length|sepal_width|petal_length|petal_width|         class|
+---+------------+-----------+------------+-----------+--------------+
|103|         7.1|        3.0|         5.9|        2.1|Iris-virginica|
|106|         7.6|        3.0|         6.6|        2.1|Iris-virginica|
|108|         7.3|        2.9|         6.3|        1.8|Iris-virginica|
|110|         7.2|        3.6|         6.1|        2.5|Iris-virginica|
|118|         7.7|        3.8|         6.7|        2.2|Iris-virginica|
|119|         7.7|        2.6|         6.9|        2.3|Iris-virginica|
|123|         7.7|        2.8|         6.7|        2.0|Iris-virginica|
|126|         7.2|        3.2|         6.0|        1.8|Iris-virginica|
|130|         7.2|        3.0|         5.8|        1.6|Iris-virginica|
|131|         7.4|        2.8|         6.1|        1.9|Iris-virginica|
|132|         7.9|        3.8|         6.4|        2.0|Iris-virginica|
|136|         7.7|        3.0|         6.1|        2.3|Iris-virginica|
+---+------------+-----------+------------+-----------+--------------+

+---------------+--------+
|          class|count(1)|
+---------------+--------+
|        Species|       1|
| Iris-virginica|      50|
|    Iris-setosa|      50|
|Iris-versicolor|      50|
+---------------+--------+
```

Titanic Dataset - Perform SQL Queries to find:

A. What is the number of passengers who survived the Titanic Disaster?

B. How many female passengers were on board the Titanic?

C. What is the average age of passengers in each passenger class?

CODE:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Online IRIS dataset").getOrCreate()

url = "/content/sample_data/Titanic-Dataset.csv"

df = spark.read.csv(url, header = False, inferSchema =  True)

columns = ["PassengerId","Survived","Pclass","Name","Sex","Age","SibSp","Parch","Ticket","Fare","Cabin","Embarked"]
df = df.toDF(*columns)

df.createOrReplaceTempView("titanic")

result1 = spark.sql("Select * from titanic  WHERE survived = 1")
result2 = spark.sql("Select  count(*) from titanic where sex = 'female'")
result3 = spark.sql("Select avg(Age) from titanic group by Pclass ")

result1.show()
result2.show()
result3.show()

spark.stop()
```

OUTPUT:

```
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+--------+-----+--------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ticket|    Fare|Cabin|Embarked|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+--------+-----+--------+
|          2|       1|     1|Cumings, Mrs. Joh...|female|  38|    1|    0|        PC 17599| 71.2833|  C85|       C|
|          3|       1|     3|Heikkinen, Miss. ...|female|  26|    0|    0|STON/O2. 3101282|   7.925| NULL|       S|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|  35|    1|    0|          113803|    53.1| C123|       S|
|          9|       1|     3|Johnson, Mrs. Osc...|female|  27|    0|    2|          347742| 11.1333| NULL|       S|
|         10|       1|     2|Nasser, Mrs. Nich...|female|  14|    1|    0|          237736| 30.0708| NULL|       C|
|         11|       1|     3|Sandstrom, Miss. ...|female|   4|    1|    1|         PP 9549|    16.7|   G6|       S|
|         12|       1|     1|Bonnell, Miss. El...|female|  58|    0|    0|          113783|   26.55| C103|       S|
|         16|       1|     2|Hewlett, Mrs. (Ma...|female|  55|    0|    0|          248706|      16| NULL|       S|
|         18|       1|     2|Williams, Mr. Cha...|  male|NULL|    0|    0|          244373|      13| NULL|       S|
|         20|       1|     3|Masselmani, Mrs. ...|female|NULL|    0|    0|            2649|   7.225| NULL|       C|
|         22|       1|     2|Beesley, Mr. Lawr...|  male|  34|    0|    0|          248698|      13|  D56|       S|
|         23|       1|     3|"McGowan, Miss. A...|female|  15|    0|    0|          330923|  8.0292| NULL|       Q|
|         24|       1|     1|Sloper, Mr. Willi...|  male|  28|    0|    0|          113788|    35.5|   A6|       S|
|         26|       1|     3|Asplund, Mrs. Car...|female|  38|    1|    5|          347077| 31.3875| NULL|       S|
|         29|       1|     3|"O'Dwyer, Miss. E...|female|NULL|    0|    0|          330959|  7.8792| NULL|       Q|
|         32|       1|     1|Spencer, Mrs. Wil...|female|NULL|    1|    0|        PC 17569|146.5208|  B78|       C|
|         33|       1|     3|Glynn, Miss. Mary...|female|NULL|    0|    0|          335677|    7.75| NULL|       Q|
|         37|       1|     3|    Mamee, Mr. Hanna|  male|NULL|    0|    0|            2677|  7.2292| NULL|       C|
|         40|       1|     3|Nicola-Yarred, Mi...|female|  14|    1|    0|            2651| 11.2417| NULL|       C|
|         44|       1|     2|Laroche, Miss. Si...|female|   3|    1|    2|   SC/Paris 2123| 41.5792| NULL|       C|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+--------+-----+--------+
only showing top 20 rows

+--------+
|count(1)|
+--------+
|     314|
+--------+

+------------------+
|          avg(Age)|
+------------------+
|              NULL|
| 25.14061971830986|
|38.233440860215055|
| 29.87763005780347|
+------------------+
```

Wine Quality Dataset Example

A. How many wines are considered high quality (quality score of 7 or higher)

B. What is the average alcohol content of the wines in the dataset

CODE:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Online IRIS dataset").getOrCreate()

url = "/content/sample_data/winequality-red.csv"

df = spark.read.csv(url, header = False, inferSchema =  True)

columns = ["fixed acidity","volatile acidity","citric acid","residual sugar","chlorides","free sulfur dioxide","total sulfur dioxide","density","pH","sulphates","alcohol","quality"]
df = df.toDF(*columns)

df.createOrReplaceTempView("wine")

result1 = spark.sql("Select count(*) from wine  WHERE quality >= 7.0")
result3 = spark.sql("Select avg(alcohol) from wine ")

result1.show()
result3.show()

spark.stop()
```

OUTPUT:

```
+--------+
|count(1)|
+--------+
|     217|
+--------+

+------------------+
|      avg(alcohol)|
+------------------+
|10.422983114446502|
+------------------+
```

California Housing Dataset Example:

A. **How many houses have a median value above $5000,000 in California**
B. **What is the average age of the houses in the dataset**

CODE:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Online IRIS dataset").getOrCreate()

url = "/content/sample_data/california_housing_train.csv"

df = spark.read.csv(url, header = False, inferSchema =  True)

columns = ["longitude", "latitude", "housing_median_age", "total_rooms", "total_bedrooms", "population", "households", "median_income", "median_house_value"]
df = df.toDF(*columns)

df.createOrReplaceTempView("california")

result1 = spark.sql("Select count(*) from california WHERE median_house_value > 50000")
result3 = spark.sql("Select avg(housing_median_age) from california ")

result1.show()
result3.show()

spark.stop()
```

OUTPUT:

```
+--------+
|count(1)|
+--------+
|   16820|
+--------+

+----------------------+
|avg(housing_median_age)|
+----------------------+
|      28.58935294117647|
+----------------------+
```

02/04/20

Name: Prerna Sunil Jadhav
Sap Id: 60004220127
Batch : C2-2
Course: Big Data Infrastructure Laboratory
course code: DJ19CEEL6011

EXPERIMENT 09

AIM: Perform CRUD operations using MongoDB.

THEORY: MongoDB is an open source document-oriented database. The term 'NOSQL' means Non-relational. Mongo DB isnt based on the table like relational database structure but provides an different mechanism for storage and relation of data. This format of storage is called BSON.
CRUD operations include create, read, update and delete documents.
Create operation: Add new document to a collection.

```
db.user.insert one (
    {
        name: "prerna",
        age: 20,
        status: "pending"
    }
)
```

Read operations: Retrieve document from a collection.

```
db.user.find(
    {
        age: {$gt:18}},
    {
        name:1,
        address:1
    }).limit(5)
```

Update operation: Modify existing documents in a collection.

```
db.user.updateMany(
    {
        age:{$lp}},
        {$set : {status:"reject"}}
)
```

Delete operation: Remove documents in a collection.

```
db.users.deleteMany(
    {status:"reject"}
)
```

CONCLUSION: Thus, we ~~are~~ performed and studied MongoDB and its operations.

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	09

AIM: Perform CRUD operation on MongoDB

script.js — Prerna Jadhav-60004220127 — NEW — MONGODB — RUN

```
db.createCollection("Employee123");
```

STDIN: Input for the program (Optional)

Output:

```
{ "ok" : 1 }
```

script.js — Prerna Jadhav-60004220127 — NEW — MONGODB — RUN

```
db.Employee123.insert(
  {
    Eid:101,
    Ename:"Akshay",
    Eaddress:"Mumbai",
    Emobile:1234567890
  }
);
```

STDIN: Input for the program (Optional)

Output:

```
WriteResult({ "nInserted" : 1 })
```

script.js — Prerna Jadhav-60004220127 — NEW — MONGODB — RUN

```
db.Employee123.insertOne(
  {
    Eid:102,
    Ename:"Arvind",
    Eaddress:"Mumbai",
    Emobile:9870654321
  }
);
```

STDIN: Input for the program (Optional)

Output:

```
{
        "acknowledged" : true,
        "insertedId" : ObjectId("660d74188089574
}
```

script.js — Prerna Jadhav-60004220127 — NEW — MONGODB — RUN

```
db.Employee123.insertMany(
  [
    {
      Eid: 103,
      Ename:"Sahil",
      Eaddress: "Pune",
      Emobile: 8765093214
    },
    {
      Eid: 104,
      Ename: "Pranay",
      Eaddress: "Bangalore",
      Emobile: 74310982365
    }
  ]
);
```

STDIN: Input for the program (Optional)

Output:

```
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("660d747739101f80b5dcd1
                ObjectId("660d747739101f80b5dcd1
        ]
}
```

```
> db.Employee123.find();
<   {
      _id: ObjectId("66038dcac78b602ec9a44675"),
      Eid: 101,
      Ename: 'Akshay',
      Eaddress: 'Mumbai',
      Emobile: 1234567890
    }
    {
      _id: ObjectId("66038dd8c78b602ec9a44676"),
      Eid: 102,
      Ename: 'Arvind',
      Eaddress: 'Mumbai',
      Emobile: 9870654321
    }
    {
      _id: ObjectId("66038e3dc78b602ec9a44677"),
      Eid: 103,
      Ename: 'Sahil',
      Eaddress: 'Pune',
      Emobile: 8765093214
    }
    {
      _id: ObjectId("66038e3dc78b602ec9a44678"),
      Eid: 104,
      Ename: 'Pranay',
      Eaddress: 'Bangalore',
      Emobile: 74310982365
    }
```

```
> db.Employee123.find({ Eaddress: "Pune" });
<   {
      _id: ObjectId("66038e3dc78b602ec9a44677"),
      Eid: 103,
      Ename: 'Sahil',
      Eaddress: 'Pune',
      Emobile: 8765093214
    }
```

```
> db.Employee123.updateMany(
      { Eaddress: "Mumbai" },
      { $set: { Esalary: 250000 } }
  );
<   {
      acknowledged: true,
      insertedId: null,
      matchedCount: 2,
      modifiedCount: 2,
      upsertedCount: 0
    }
```

```
> db.Employee123.find({ Esalary: { $gt: 150000 } });
<   {
      _id: ObjectId("66038dcac78b602ec9a44675"),
      Eid: 101,
      Ename: 'Akshay',
      Eaddress: 'Mumbai',
      Emobile: 1234567890,
      Esalary: 250000
    }
    {
      _id: ObjectId("66038dd8c78b602ec9a44676"),
      Eid: 102,
      Ename: 'Arvind',
      Eaddress: 'Mumbai',
      Emobile: 9870654321,
      Esalary: 250000
    }
```

Name: Prerna Sunil Jadhav
Sap Id: 60004220127
Batch: C2-2
Course: Big Data Infrastructure laboratory
Course Code: DJ19CEEL6011

EXPERIMENT 10

AIM: Perform sentiment analysis using Kafka.

THEORY: Data reasoning is the process of transmitting a continuous data (also known a streams) typically fed into stream processing software to derive valuable insights.
Apache Kafka is an open source, distributed streaming platform that enables the development of real time, event driven application.

Components of Apache Kafka

- Producers
- Brokers
- Topics & Partition
- Replicas
- consumers
- followers & leaders

- **Producers:** Producers in Kafka publish messages to one or more topics.

- **Brokers:** A Kafka cluster comprises one or more servers that are known as brokers. Broker works as a container that can hold multiple topics

- Topic: A stream of messages that are a part of specific category or feed name is referred to as a kafka topic.

- Partitions: Topics in kafka are divided into a configurable no. of parts, which are known as partitions.

- Replicas: Replicas are like backups for partition in kafka.

- Leaders & followers: Easy Every partition will have one server that plays the role of a leader for that partition. Leader will perform read and write operations. Followers will replicate the data of the leader.

CONCLUSION: Thus, we performed sentiment analysis using kafka.

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B. Tech (Computer Engineering)
Course:	Big Data Infrastructure Laboratory
Course Code:	DJ19CEEL6011
Experiment No.:	10

AIM: Perform Sentiment Analysis using Kafka.

Step 1 : As the Log Data is unstructured, we parse and create a structure from each line, which will in turn become each row while analysis.

```
import re
from pyspark.sql import Row
# This is the regex which is specific to Apache Access Logs parsing, which can be modified according to different Log formats as per the need
# Example Apache log line:
#   127.0.0.1 - - [21/Jul/2014:9:55:27 -0800] "GET /home.html HTTP/1.1" 200 2048
#   1:IP  2:client 3:user 4:date time           5:method 6:req 7:proto   8:respcode 9:size
APACHE_ACCESS_LOG_PATTERN = '^(\S+) (\S+) (\S+) \[([\w:/]+\s[+\-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3}) (\d+)'

# The below function is modelled specific to Apache Access Logs Model, which can be modified as per needs to different Logs format
# Returns a dictionary containing the parts of the Apache Access Log.
def parse_apache_log_line(logline):
    match = re.search(APACHE_ACCESS_LOG_PATTERN, logline)
    if match is None:
        raise Error("Invalid logline: %s" % logline)
    return Row(
        ip_address    = match.group(1),
        client_identd = match.group(2),
        user_id       = match.group(3),
        date = (match.group(4)[:-6]).split(":", 1)[0],
        time = (match.group(4)[:-6]).split(":", 1)[1],
        method        = match.group(5),
        endpoint      = match.group(6),
        protocol      = match.group(7),
        response_code = int(match.group(8)),
        content_size  = int(match.group(9))
    )
```

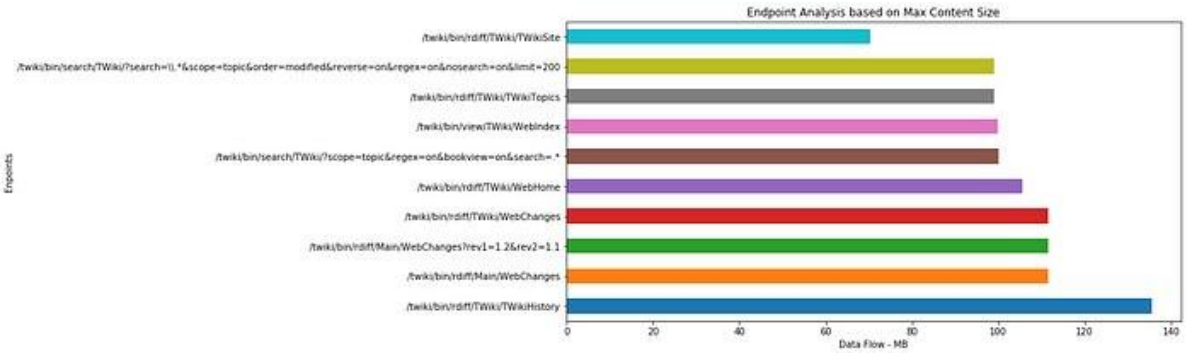
Step 2: Create Spark Context, SQL Context, DataFrame (is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database)

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

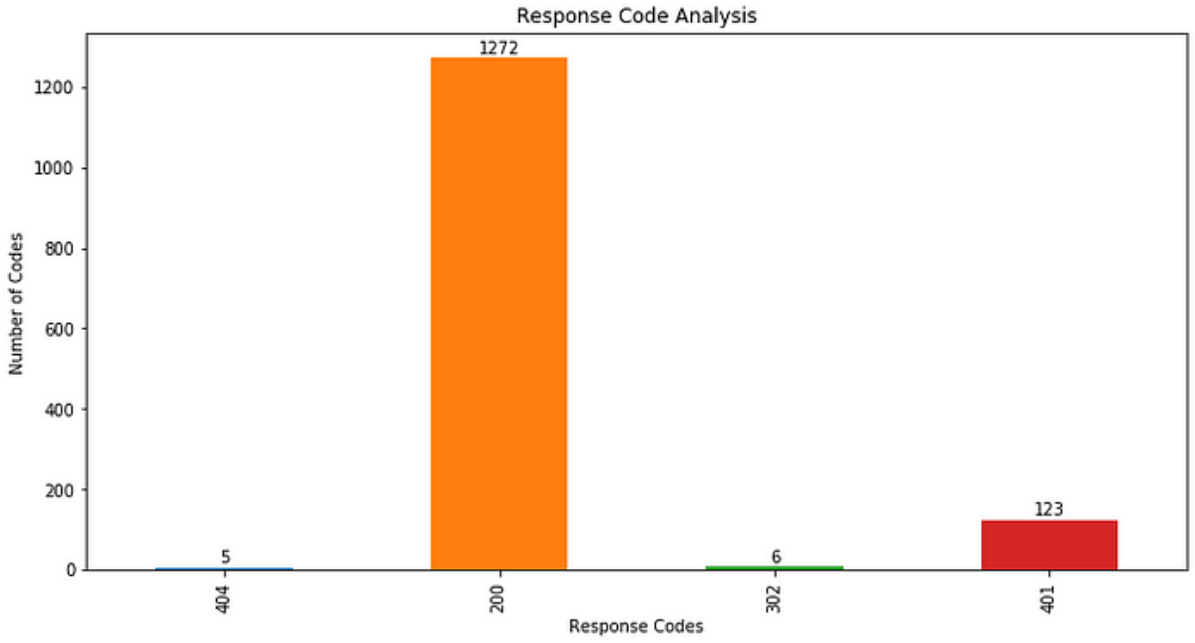
```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
import apache_access_log # This is the first file name , in which we created Data Structure of Log
import sys

# Set up The Spark App
conf = SparkConf().setAppName("Log Analyzer")
# Create Spark Context
sc = SparkContext(conf=conf)
#Create SQL Context
sqlContext = SQLContext(sc)

#Input File Path
logFile = 'Give Your Input File Path Here'

# .cache() - Persists the RDD in memory, which will be re-used again
access_logs = (sc.textFile(logFile)
               .map(apache_access_log.parse_apache_log_line)
               .cache())

schema_access_logs = sqlContext.createDataFrame(access_logs)
#Creates a table on which SQL like queries can be fired for analysis
schema_access_logs.registerTempTable("logs")
```

Step 3 : Analyze Top 10 Endpoints which Transfer Maximum Content in MB

```
#Top 10 Endpoints which Transfer Maximum Content
#.rdd.map() - Will convert the resulted rows from SQL query into a map
# .collect() - actually executes the DAG to get the overall results
topEndpointsMaxSize = (sqlContext
               .sql("SELECT endpoint,content_size/1024 FROM logs ORDER BY content_size DESC LIMIT 10")
               .rdd.map(lambda row: (row[0], row[1]))
               .collect())
# Plot Analysis Code
bar_plot_list_of_tuples_horizontal(topEndpointsMaxSize,'Data Flow - MB','Enpoints','Endpoint Analysis
    based on Max Content Size')
```

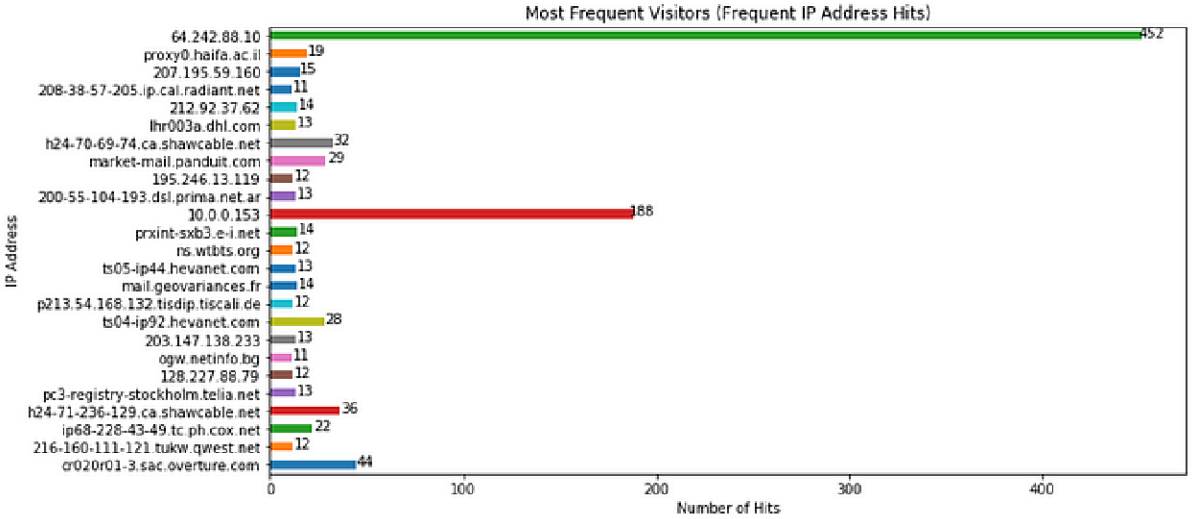


```
# Response Code Analysis
responseCodeToCount = (sqlContext
                       .sql("SELECT response_code, COUNT(*) AS theCount FROM logs GROUP BY
                           response_code")
                       .rdd.map(lambda row: (row[0], row[1]))
                       .collect())
bar_plot_list_of_tuples(responseCodeToCount,'Response Codes','Number of Codes','Response Code Analysis'
    )

# Code to Plot the results
def bar_plot_list_of_tuples(input_list,x_label,y_label,plot_title):
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    plt.figure(figsize=(12, 6))
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(plot_title)
    ax = pd.Series(y_labels).plot(kind='bar')
    ax.set_xticklabels(x_labels)
    rects = ax.patches
    for rect, label in zip(rects, y_labels):
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='center', va='bottom')
```

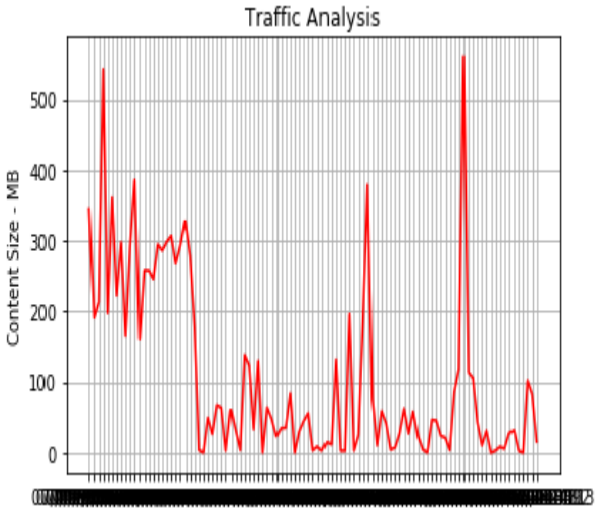


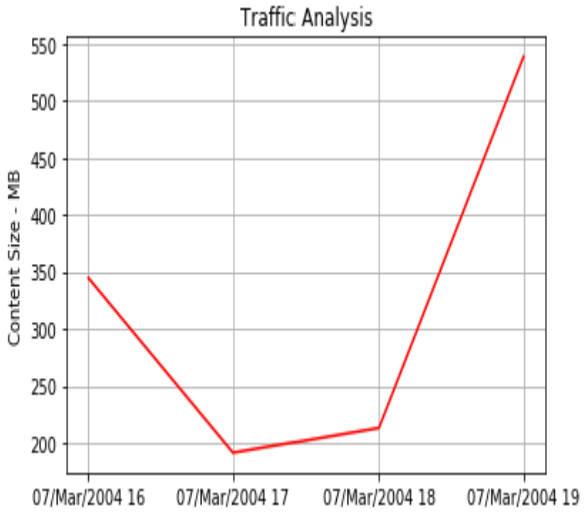

```
# Most Frequent Visitors (Most Frequent IP Address visits).
frequentIpAddressesHits = (sqlContext
                .sql("SELECT ip_address, COUNT(*) AS total FROM logs GROUP BY ip_address HAVING total >
                    10")
                .rdd.map(lambda row: (row[0], row[1]))
                .collect())
bar_plot_list_of_tuples_horizontal(frequentIpAddressesHits,'Number of Hits','IP Address','Most Frequent
    Visitors (Frequent IP Address Hits)')
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

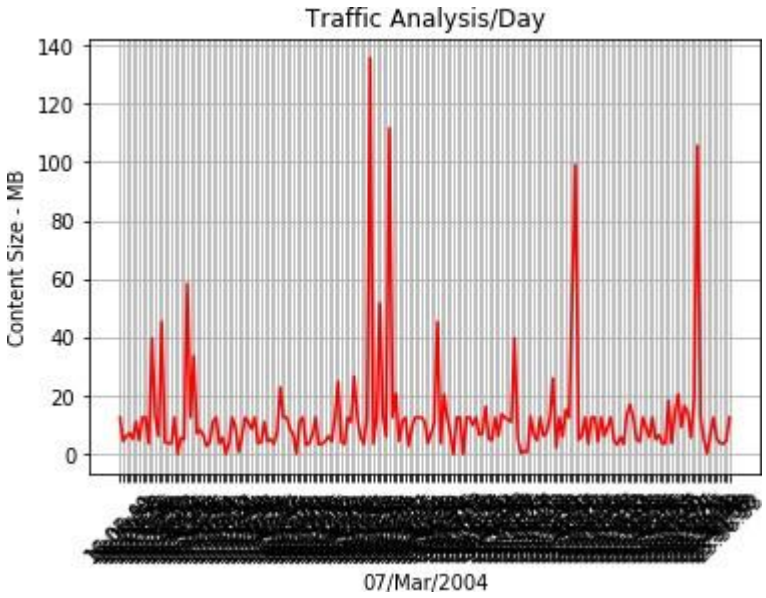



```
1  # Traffic Analysis for past One Week
2  trafficWithTime = (sqlContext
3                     .sql("SELECT date, content_size/1024 FROM logs")
4                     .rdd.map(lambda row: (row[0], row[1]))
5                     .collect())
6  time_series_plot(trafficWithTime)
7
```

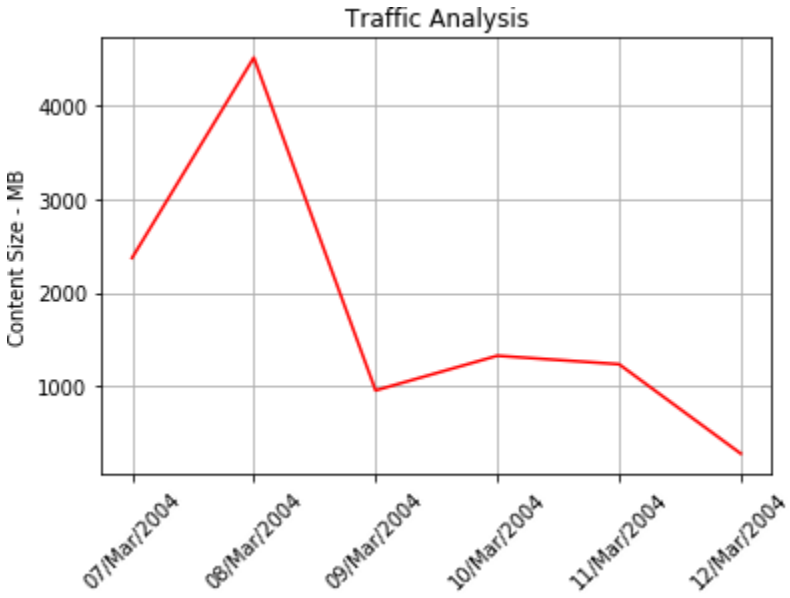




```
# Overall Traffic Analysis for a Day
Day = '07/Mar/2004'
trafficperDay = (sqlContext
                 .sql("SELECT time,content_size/1024 FROM logs where date='07/Mar/2004'")
                 .rdd.map(lambda row: (row[0], row[1]))
                 .collect())
time_series_plot(trafficperDay,Day,'Content Size - MB','Traffic Analysis/Day')
```



Outliers can be clearly detected by analysis the spikes and which end points were been hit at time by what IP Addresses.



Here, we can see an unusual spike on 8th March, which can be analyzed further for identifying discrepancy.

Code for Plot Analysis:

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

```
def time_series_plot(input_list,x_label,y_lablel,plot_title):
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    dict_plot = OrderedDict()
    for x,y in zip(x_labels,y_labels):
       # cur_val = x.split(":", 1)[0]
        cur_val = x.split(" ")[0]
        #print(cur_val)
        dict_plot[cur_val] = dict_plot.get(cur_val, 0) + y
    input_list = list(dict_plot.items())
    x_labels = [val[0] for val in input_list]
    y_labels = [val[1] for val in input_list]
    plt.plot_date(x=x_labels, y=y_labels, fmt="r-")
    plt.xticks(rotation=45)
    plt.title(plot_title)
    plt.xlabel(x_label)
    plt.ylabel(y_lablel)
    plt.grid(True)
    plt.show()
```

```
20
21      def bar_plot_list_of_tuples_horizontal(input_list,x_label,y_label,plot_title):
22      y_labels = [val[0] for val in input_list]
23      x_labels = [val[1] for val in input_list]
24      plt.figure(figsize=(12, 6))
25      plt.xlabel(x_label)
26      plt.ylabel(y_label)
27      plt.title(plot_title)
28      ax = pd.Series(x_labels).plot(kind='barh')
29      ax.set_yticklabels(y_labels)
30      for i, v in enumerate(x_labels):
31          ax.text(int(v) + 0.5, i - 0.25, str(v),ha='center', va='bottom')
```

```
32
33      # Frequent End Points
34  topEndpoints = (sqlContext
35                  .sql("SELECT endpoint, COUNT(*) AS total FROM logs GROUP BY endpoint ORDER BY total
                        DESC LIMIT 10")
36                  .rdd.map(lambda row: (row[0], row[1]))
37                  .collect())
38  bar_plot_list_of_tuples_horizontal(topEndpoints,'Number of Times Accessed','End Points','Most
        Frequent Endpoints')
```