

**DEPARTMENT OF ARTIFICAL INTELLIGENCE AND MACHINE LEARNING**



**AI19741- BIG DATA TECHNOLOGY LABORATORY**

**LAB MANUAL**

**FINAL YEAR**

**SEVENTH SEMESTER**

**2024- 2025**

**ODD SEMESTER**

**List of Experiments**

1. Installation of Hadoop (3)

2. File Management tasks in Hadoop.(3)

● Upload and download a file in HDFS

● Copy a file from source to destination

● Copy to file from /to local file system to HDFS

● Move file from source to destination

● Remove a file/directory in HDFS

3. Implement word count program using Map Reduce.(3)

4. Weather Report POC-Map Reduce Program to analyze time-temperature statistics and generate report with max/min temperature.(3)

5. Pig Latin scripts to sort, group, join, project, and filter your data.(6)

6. Hive Databases, Tables, Views, Functions and Indexes .(6)

7. Programs in Sqoop: Export data from Hadoop using Sqoop to import data to Hive.(6)

**RAJALAKSHMI ENGINEERING COLLEGE**

**DEPARTMENT OF ARTIFICAL INTELLIGENCE AND MACHINE LEARNING**

**AI19741- BIG DATA TECHNOLOGY LABORATORY**

**LAB PLAN**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Name of the Experiment** | **Hours Planned** |
| 1 | Installation of Hadoop | 4 |
| 2. | File Management tasks in Hadoop.  ● Upload and download a file in HDFS  ● Copy a file from source to destination  ● Copy to file from /to local file system to HDFS  ● Move file from source to destination  ● Remove a file/directory in HDFS | 2 |
| 3 | Implement word count program using Map Reduce. | 4 |
| 4 | Weather Report POC-Map Reduce Program to analyze time-temperature statistics and generate report with max/min temperature | 4 |
| 5.a | Pig Latin scripts to sort, group data. | 2 |
| b | Pig Latin scripts to project, and filter your data. | 4 |
| 6a | Hive Databases -> Tables, Views. | 2 |
| b | Hive Databases-> Functions and Indexes | 4 |
| 7 | Export data from Hadoop using Sqoop to import data to Hive. | 6 |
| Total Hours | | 32 |

**HARDWARE AND SOFTWARE REQUIREMENTS**

|  |  |
| --- | --- |
| Hardware Requirements | Core i3 and above, 8 GB RAM, minimum 10 GB harddisk |
| Software Requirements | Fedora 36 or other Linux variants/Windows 10, Java 8.0 JDK and JRE, Hadoop stable version |

**Course Outcomes (COs)**

**Course Name: Big Data Technology Laboratory Course Code: AI19741**

|  |  |
| --- | --- |
| Outcome 1 | Get familiar with the concepts of big data and Hadoop |
| Outcome 2 | Understand the process of accessing, storing and manipulating the huge data from different resources. |
| Outcome 3 | Learn the working principles of big data management using NoSQL |
| Outcome 4 | Learn and implement small programs in Pig, Hive and HBase |
| Outcome 5 | Get the concepts of Sqoop and Solr. |

# CO-PO –PSO matrices of course

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PO/PSO  CO | PO 1 | PO 2 | PO 3 | PO 4 | PO 5 | PO 6 | PO 7 | PO 8 | PO 9 | PO 10 | PO 11 | PO 12 | PSO 1 | PSO 2 | PSO 3 |
| AI19741.1 | - | 1 | - | 2 | 2 | - | - | - | - | - | - | 1 | 3 | 3 | 3 |
| AI19741.2 | - | 2 | 1 | 2 | 2 | - | - | - | 1 | 2 | - | 1 | 3 | 3 | 3 |
| AI19741.3 | 1 | 2 | 1 | 2 | 2 | - | - | - | 1 | 2 | - | 1 | 3 | 3 | 3 |
| AI19741.4 | 1 | 2 | 1 | 2 | 2 | - | - | - | 1 | 2 | - | 1 | 3 | 3 | 3 |
| AI19741.5 | 1 | 2 | 1 | 2 | 2 | - | - | - | 1 | 2 | - | 1 | 3 | 3 | 3 |
| Average | 1 | 1.8 | 1 | 2 | 2 | - | - | - | 1 | 2 | - | 1 | 3 | 3 | 3 |

Note: Enter correlation levels 1, 2 or 3as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

If there is no correlation, put “-“

# **Ex No: 1** **Installation of Hadoop Framework,**

# **AIM:**

Installation of Hadoop Framework, it‘s components and study the HADOOP ecosystem

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

**Hadoop Architecture:**

The Apache Hadoop framework includes following four modules:

**Hadoop Common**:

Contains Java libraries and utilities needed by other Hadoop modules. These libraries give file system and OS level abstraction and comprise of the essential Java files and scripts that are required to start Hadoop.

**Hadoop Distributed File System (HDFS):** A distributed file-system that provides highthroughput access to application data on the community machines thus providing very high aggregate bandwidth across the cluster.

**Hadoop YARN**: A resource-management framework responsible for job scheduling and cluster resource management.

**Hadoop MapReduce**: This is a YARN- based programming model for parallel processing of large data sets.

**Hadoop Installation procedure:**

**Step 1:** Download and install Java

https://www.oracle.com/java/technologies/javase-downloads.html

**Step 2:** Download Hadoop

https://hadoop.apache.org/releases.html

**Step 3:** Set Environment Variables

**Step 4:** Setup Hadoop

ou must configure Hadoop in this phase by modifying several configuration files. Navigate to the “etc/hadoop” folder in the Hadoop folder. You must make changes to three files:

core-site.xml

<configuration>  
 <property>  
 <name>fs.default.name</name>  
 <value>hdfs://localhost:9000</value>  
 </property>  
</configuration>

hdfs-site.xml

<configuration>  
 <property>  
 <name>dfs.replication</name>  
 <value>1</value>  
 </property>  
 <property>  
 <name>dfs.namenode.name.dir</name>  
 <value>file:/hadoop-3.3.1/data/namenode</value>  
 </property>  
 <property>  
 <name>dfs.datanode.data.dir</name>  
 <value>file:/hadoop-3.3.1/data/datanode</value>  
 </property>  
</configuration>

mapred-site.xml

<configuration>  
 <property>  
 <name>mapred.job.tracker</name>  
 <value>localhost:54311</value>  
 </property>  
</configuration>

## Step 5: Format Hadoop NameNode

hadoop namenode –format

## Step 6: Start Hadoop

start-all.cmd

## Step 7: Verify Hadoop Installation

<http://localhost:50070/>.

# **Ex No: 2** **File Management tasks in Hadoop.**

# **AIM:**

To perform various file operation in HDFS

PRACTICAL-6:

AIM: File Management tasks in Hadoop

1. Create a directory in HDFS at

given path(s).

Usage:

hadoop fs -mkdir <paths>

Example:

hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2

2. List the contents of a

directory.

Usage :

hadoop fs -ls <args>

Example:

hadoop fs -ls /user/saurzcode

3. Upload and download a file in

HDFS.

Upload:

hadoop fs -put:

Copy single src file, or multiple src files from local file system

to the Hadoop data file system

Usage:

hadoop fs -put <localsrc> ... <HDFS\_dest\_Path>

Example:

hadoop fs -put /home/saurzcode/Samplefile.txt /user/

saurzcode/dir3/

Download:

hadoop fs -get:

Copies/Downloads files to the local file system

Usage:

hadoop fs -get <hdfs\_src> <localdst>

Example:

hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/

4. See contents of a file

Same as unix cat command:

Usage:

hadoop fs -cat <path[filename]>

Example:

hadoop fs -cat /user/saurzcode/dir1/abc.txt

5. Copy a file from source to

destination

This command allows multiple sources as well in which case

the destination must be a directory.

Usage:

hadoop fs -cp <source> <dest>

Example:

hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/

dir2

6. Copy a file from/To Local file

system to HDFS

copyFromLocal

Usage:

hadoop fs -copyFromLocal <localsrc> URI

1. Create a directory in HDFS at

given path(s).

Usage:

hadoop fs -mkdir <paths>

Example:

hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir

### Step 1: Adding Files and Directories to HDFS

Before running Hadoop programs on data stored in HDFS, the data needs to be added to HDFS. Let's start by creating a directory and adding a file to it.

1. **Create a directory in HDFS:**

hadoop fs -mkdir /user/myfile

This command creates a new directory named myfile in the /user directory in HDFS.

1. **Add a file to HDFS:**

hadoop fs -put a.txt

This command uploads the file a.txt from the local filesystem to the root directory of HDFS.

1. **Add the file to the newly created directory:**

hadoop fs -put a.txt /user/myfile

This command uploads the file a.txt from the local filesystem directly into the /user/myfile directory in HDFS.

### Step 2: Retrieving Files from HDFS

To copy files from HDFS back to the local filesystem, use the get command. Here’s how to retrieve a.txt:

hadoop fs -cat a.txt

This command displays the contents of the file a.txt directly to the console. To actually copy the file to the local filesystem, you would use:

hadoop fs -get a.txt /local/path

Replace /local/path with the desired path on your local filesystem.

### Step 3: Deleting Files from HDFS

To delete a file from HDFS, use the rm command. Here’s how to delete a.txt:

hadoop fs -rm a.txt

This command removes the file a.txt from HDFS.

## Output

The successful execution of the above commands will result in the following:

* Creation of the /user/myfile directory in HDFS.
* Addition of a.txt to HDFS and then to /user/myfile.
* Retrieval of a.txt from HDFS to the local filesystem.
* Deletion of a.txt from HDFS.

# **Ex No: 3 Implement word count program using Map Reduce.**

# **AIM:**

To implementing distinct word count problem using Map-Reduce

The function of the mapper is as follows:

• Create a IntWritable variable 'one' with value as 1

• Convert the input line in Text type to a String

• Use a tokenizer to split the line into words

• Iterate through each word and a form key value pairs as Assign each work from the tokenizer (of String type) to a Text 'word'

• Form key value pairs for each word as < word,one > and push it to the output collector

The function of Sort and Group:

After this, "aggregation" and "Shuffling and Sorting" done by framework.

Then Reducers task these final pair to produce output.

The function of the reducer is as follows

• Initialize a variable 'sum' as 0

• Iterate through all the values with respect to a key and sum up all of them

• Push to the output collector the Key and the obtained sum as value

For Example:

For the given sample input1 data file (input1.txt : Hello World Bye World) mapper emits:

<Hello,1>

<World,1>

<Bye,1>

<World,1>

The second input2 data file (input2.txt : Hello Hadoop Goodbye Hadoop) mapper emits:

<Hello,1>

<Hadoop,1>

<Goodbye,1>

<Hadoop,1>

WordCount also specifies a combiner. Hence, the output of each map is passed through the local combiner (which is same as the Reducer as per the job configuration) for local aggregation, after being sorted on the keys.

The output of the first map:

<Hello,1>

<Bye,1>

<World,2>

The output of the second map:

<Hello,1>

<Hadoop,2>

<Goodbye,1>

The Reducer implementation via the reduce method just sums up the values, which are the occurence counts for each key (i.e. words in this example).

Thus the output of the job is:

<Goodbye,1>

<Bye,1>

<Hello,2>

<Hadoop,2>

<World,2>

# **Ex No: 4 Map Reduce Program for Weather Report.**

# **AIM:**

To write a Map Reduce Program to analyze time-temperature statistics and generate report with max/min temperature Weather Report POC.

Program:

// importing Libraries

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.conf.Configuration;

public class MyMaxMin {

// Mapper

/\*MaxTemperatureMapper class is static

\* and extends Mapper abstract class

\* having four Hadoop generics type

\* LongWritable, Text, Text, Text.

\*/

public static class MaxTemperatureMapper extends

Mapper<LongWritable, Text, Text, Text> {

/\*\*

\* @method map

\* This method takes the input as a text data type.

\* Now leaving the first five tokens, it takes

\* 6th token is taken as temp\_max and

\* 7th token is taken as temp\_min. Now

\* temp\_max > 30 and temp\_min < 15 are

\* passed to the reducer.

\*/

// the data in our data set with

// this value is inconsistent data

public static final int MISSING = 9999;

@Override

public void map(LongWritable arg0, Text Value, Context context)

throws IOException, InterruptedException {

// Convert the single row(Record) to

// String and store it in String

// variable name line

String line = Value.toString();

// Check for the empty line

if (!(line.length() == 0)) {

// from character 6 to 14 we have

// the date in our dataset

String date = line.substring(6, 14);

// similarly we have taken the maximum

// temperature from 39 to 45 characters

float temp\_Max = Float.parseFloat(line.substring(39, 45).trim());

// similarly we have taken the minimum

// temperature from 47 to 53 characters

float temp\_Min = Float.parseFloat(line.substring(47, 53).trim());

// if maximum temperature is

// greater than 30, it is a hot day

if (temp\_Max > 30.0) {

// Hot day

context.write(new Text("The Day is Hot Day :" + date),

new Text(String.valueOf(temp\_Max)));

}

// if the minimum temperature is

// less than 15, it is a cold day

if (temp\_Min < 15) {

// Cold day

context.write(new Text("The Day is Cold Day :" + date),

new Text(String.valueOf(temp\_Min)));

}

}

}

}

// Reducer

/\*MaxTemperatureReducer class is static

and extends Reducer abstract class

having four Hadoop generics type

Text, Text, Text, Text.

\*/

public static class MaxTemperatureReducer extends

Reducer<Text, Text, Text, Text> {

/\*\*

\* @method reduce

\* This method takes the input as key and

\* list of values pair from the mapper,

\* it does aggregation based on keys and

\* produces the final context.

\*/

public void reduce(Text Key, Iterator<Text> Values, Context context)

throws IOException, InterruptedException {

// putting all the values in

// temperature variable of type String

String temperature = Values.next().toString();

context.write(Key, new Text(temperature));

}

}

/\*\*

\* @method main

\* This method is used for setting

\* all the configuration properties.

\* It acts as a driver for map-reduce

\* code.

\*/

public static void main(String[] args) throws Exception {

// reads the default configuration of the

// cluster from the configuration XML files

Configuration conf = new Configuration();

// Initializing the job with the

// default configuration of the cluster

Job job = new Job(conf, "weather example");

// Assigning the driver class name

job.setJarByClass(MyMaxMin.class);

// Key type coming out of mapper

job.setMapOutputKeyClass(Text.class);

// value type coming out of mapper

job.setMapOutputValueClass(Text.class);

// Defining the mapper class name

job.setMapperClass(MaxTemperatureMapper.class);

// Defining the reducer class name

job.setReducerClass(MaxTemperatureReducer.class);

// Defining input Format class which is

// responsible to parse the dataset

// into a key value pair

job.setInputFormatClass(TextInputFormat.class);

// Defining output Format class which is

// responsible to parse the dataset

// into a key value pair

job.setOutputFormatClass(TextOutputFormat.class);

// setting the second argument

// as a path in a path variable

Path OutputPath = new Path(args[1]);

// Configuring the input path

// from the filesystem into the job

FileInputFormat.addInputPath(job, new Path(args[0]));

// Configuring the output path from

// the filesystem into the job

FileOutputFormat.setOutputPath(job, new Path(args[1]));

// deleting the context path automatically

// from hdfs so that we don't have

// to delete it explicitly

OutputPath.getFileSystem(conf).delete(OutputPath);

// exiting the job only if the

// flag value becomes false

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

# **Ex No: 5.a** Pig Latin scripts to sort, group

# **AIM:**

To write a script for sorting and grouping of data.

Student data:

Assume we have a file **student\_data.txt** in HDFS with the following content.

001,Rajiv,Reddy,21,9848022337,Hyderabad

002,siddarth,Battacharya,22,9848022338,Kolkata

003,Rajesh,Khanna,22,9848022339,Delhi

004,Preethi,Agarwal,21,9848022330,Pune

005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar

006,Archana,Mishra,23,9848022335,Chennai

007,Komal,Nayak,24,9848022334,trivendram

008,Bharathi,Nambiayar,24,9848022333,Chennai

Step 1:

Load and store the student data in HDFS .

grunt> student = LOAD 'hdfs://localhost:9000/pig\_data/student\_data.txt'

USING PigStorage(',')

as ( id:int, firstname:chararray, lastname:chararray, phone:chararray,

city:chararray );

The **ORDER BY** operator is used to display the contents of a relation in a sorted order based on one or more fields.

grunt> Relation\_name2 = ORDER Relatin\_name1 BY (ASC|DESC);

Verify the relation **order\_by\_data** using the **DUMP** operator as shown below.

grunt> Dump order\_by\_data;

### Output

It will produce the following output, displaying the contents of the relation **order\_by\_data**.

(8,Bharathi,Nambiayar,24,9848022333,Chennai)

(7,Komal,Nayak,24,9848022334,trivendram)

(6,Archana,Mishra,23,9848022335,Chennai)

(5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar)

(3,Rajesh,Khanna,22,9848022339,Delhi)

(2,siddarth,Battacharya,22,9848022338,Kolkata)

(4,Preethi,Agarwal,21,9848022330,Pune)

(1,Rajiv,Reddy,21,9848022337,Hyderabad)

The **GROUP** operator is used to group the data in one or more relations. It collects the data having the same key.

Given below is the syntax of the **group** operator.

Now, let us group the records/tuples in the relation by age as shown below.

grunt> group\_data = GROUP student\_details by age;

Verify the relation **group\_data** using the **DUMP** operator as shown below.

grunt> Dump group\_data;

Output:

(21,{(4,Preethi,Agarwal,21,9848022330,Pune),(1,Rajiv,Reddy,21,9848022337,Hydera bad)})

(22,{(3,Rajesh,Khanna,22,9848022339,Delhi),(2,siddarth,Battacharya,22,984802233 8,Kolkata)})

(23,{(6,Archana,Mishra,23,9848022335,Chennai),(5,Trupthi,Mohanthy,23,9848022336 ,Bhuwaneshwar)})

(24,{(8,Bharathi,Nambiayar,24,9848022333,Chennai),(7,Komal,Nayak,24,9848022334, trivendram)})

# **Ex No: 5.b** Pig Latin scripts to project, and filter your data.

# **AIM:**

To write a script to performing project and filtering.

The **FILTER** operator is used to select the required tuples from a relation based on a condition.

Given below is the syntax of the **FILTER** operator.

grunt> Relation2\_name = FILTER Relation1\_name BY (condition);

**student\_details.txt**

001,Rajiv,Reddy,21,9848022337,Hyderabad

002,siddarth,Battacharya,22,9848022338,Kolkata

003,Rajesh,Khanna,22,9848022339,Delhi

004,Preethi,Agarwal,21,9848022330,Pune

005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar

006,Archana,Mishra,23,9848022335,Chennai

007,Komal,Nayak,24,9848022334,trivendram

008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Pig with the relation name **student\_details** as shown below.

grunt> student\_details = LOAD 'hdfs://localhost:9000/pig\_data/student\_details.txt' USING PigStorage(',')

as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

Let us now use the Filter operator to get the details of the students who belong to the city Chennai.

filter\_data = FILTER student\_details BY city == 'Chennai';

### Verification

Verify the relation **filter\_data** using the **DUMP** operator as shown below.

grunt> Dump filter\_data;

### Output

It will produce the following output, displaying the contents of the relation **filter\_data** as follows.

(6,Archana,Mishra,23,9848022335,Chennai)

(8,Bharathi,Nambiayar,24,9848022333,Chennai)

# **Ex No: 6.a Hive Databases -> Tables, Views**

# AIM:

To write a script to Hive Databases -> Tables, Views,

## Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables. The **syntax** for this statement is as follows:

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**:

hive> CREATE DATABASE [IF NOT EXISTS] userdb;

**or**

hive> CREATE SCHEMA userdb;

The following query is used to verify a databases list:

hive> SHOW DATABASES;

default

userdb

### JDBC Program

The JDBC program to create a database is given below.

import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;

public class HiveCreateDb {

private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

public static void main(String[] args) throws SQLException {

// Register driver and create driver instance

Class.forName(driverName);

// get connection

Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");

Statement stmt = con.createStatement();

stmt.executeQuery("CREATE DATABASE userdb");

System.out.println(“Database userdb created successfully.”);

con.close();

}

}

Save the program in a file named HiveCreateDb.java. The following commands are used to compile and execute this program.

$ javac HiveCreateDb.java

$ java HiveCreateDb

### Output:

Database userdb created successfully.

## Creating a View

You can create a view at the time of executing a SELECT statement. The syntax is as follows:

CREATE VIEW [IF NOT EXISTS] view\_name [(column\_name [COMMENT column\_comment], ...) ]

[COMMENT table\_comment]

AS SELECT ...

## Example

Let us take an example for view. Assume employee table as given below, with the fields Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000. We store the result in a view named **emp\_30000.**

+------+--------------+-------------+-------------------+--------+

| ID | Name | Salary | Designation | Dept |

+------+--------------+-------------+-------------------+--------+

|1201 | Gopal | 45000 | Technical manager | TP |

|1202 | Manisha | 45000 | Proofreader | PR |

|1203 | Masthanvali | 40000 | Technical writer | TP |

|1204 | Krian | 40000 | Hr Admin | HR |

|1205 | Kranthi | 30000 | Op Admin | Admin |

+------+--------------+-------------+-------------------+--------+

The following query retrieves the employee details using the above scenario:

hive> CREATE VIEW emp\_30000 AS

SELECT \* FROM employee

WHERE salary>30000;

## Dropping a View

Use the following syntax to drop a view:

DROP VIEW view\_name

The following query drops a view named as emp\_30000:

hive> DROP VIEW emp\_30000;

# **Ex No: 6.b Hive Databases-> Functions and Indexes**

# AIM:

To write a script to Hive Databases -> **Functions and Indexes**

The following queries demonstrate some built-in functions:

### round() function

hive> SELECT round(2.6) from temp;

On successful execution of query, you get to see the following response:

3.0

### floor() function

hive> SELECT floor(2.6) from temp;

On successful execution of the query, you get to see the following response:

2.0

### ceil() function

hive> SELECT ceil(2.6) from temp;

On successful execution of the query, you get to see the following response:

3.0

## Aggregate Functions

Hive supports the following built-in **aggregate functions**. The usage of these functions is as same as the SQL aggregate functions.

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Signature** | **Description** |
| BIGINT | count(\*), count(expr), | count(\*) - Returns the total number of retrieved rows. |
| DOUBLE | sum(col), sum(DISTINCT col) | It returns the sum of the elements in the group or the sum of the distinct values of the column in the group. |
| DOUBLE | avg(col), avg(DISTINCT col) | It returns the average of the elements in the group or the average of the distinct values of the column in the group. |
| DOUBLE | min(col) | It returns the minimum value of the column in the group. |
| DOUBLE | max(col) | It returns the maximum value of the column in the group. |

## Creating an Index

An Index is nothing but a pointer on a particular column of a table. Creating an index means creating a pointer on a particular column of a table. Its syntax is as follows:

CREATE INDEX index\_name

ON TABLE base\_table\_name (col\_name, ...)

AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

[IDXPROPERTIES (property\_name=property\_value, ...)]

[IN TABLE index\_table\_name]

[PARTITIONED BY (col\_name, ...)]

[

[ ROW FORMAT ...] STORED AS ...

| STORED BY ...

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

## Example

Let us take an example for index. Use the same employee table that we have used earlier with the fields Id, Name, Salary, Designation, and Dept. Create an index named index\_salary on the salary column of the employee table.

The following query creates an index:

hive> CREATE INDEX inedx\_salary ON TABLE employee(salary)

AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';

It is a pointer to the salary column. If the column is modified, the changes are stored using an index value.

## Dropping an Index

The following syntax is used to drop an index:

DROP INDEX <index\_name> ON <table\_name>

The following query drops an index named index\_salary:

hive> DROP INDEX index\_salary ON employee;

# **Ex No: 7 Export data from Hadoop using Sqoop**

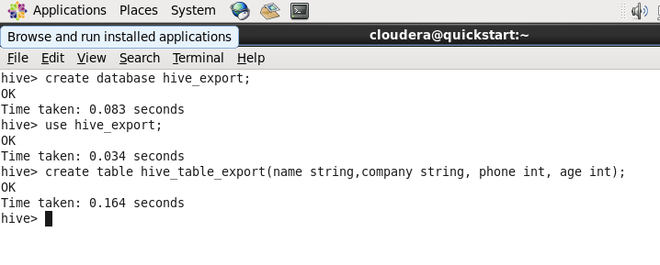
# AIM:

# To export data from Hadoop using Sqoop to import data to Hive**.**

To export data into MySQL from HDFS, perform the following steps:

**Step 1:** Create a database and table in the hive.

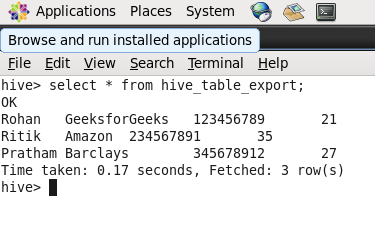
*create table hive\_table\_export(name string,company string, phone int, age int) row format delimited fields terminated by ‘,’;*



*Hive Database : hive\_export and Hive Table : hive\_table\_export*

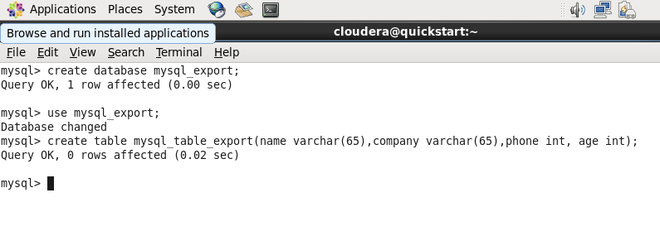
**Step 2:** Insert data into the hive table.

insert into hive\_table\_export values("Ritik","Amazon",234567891,35);



*Data in Hive table*

**Step 3:** Create a database and table in MySQL in which data should be exported.



*MySQL Database : mysql\_export and MySQL Table : mysql\_table\_export*

**Step 4:** Run the following command on Hadoop.

sqoop export --connect \

jdbc:mysql://127.0.0.1:3306/database\_name\_in\_mysql \

--table table\_name\_in\_mysql \

--username root --password cloudera \

--export-dir /user/hive/warehouse/hive\_database\_name.db/table\_name\_in\_hive \

--m 1 \

-- driver com.mysql.jdbc.Driver

--input-fields-terminated-by ','



*SQOOP command to export data*

In the above code following things should be noted.

* **127.0.0.1** is the localhost IP address.
* **3306** is the port number for MySQL.
* In the case of exporting data, the entire path to the table should be specified
* **m** is the number of mappers

**Step 5:** Check-in MySQL if data is exported successfully or not.

