Sri Shridevi Charitable Trust (R.)
SHRIDEVI INSTITUTE OF ENGINEERING AND TECHNOLOGY
Approved by: AICTE, New Delhi
Recognised by: Govt. of Karnataka
Affiliated to: Visvesvaraya Technological University, Belagavi
22 YEARS Academic Excellence
NAAC

SHRIDEVI EDUCATION

LAB MANUAL

(As per CBCS Scheme 2022)

BIG DATA ANALYTICS

Sub Code: **BAD601**

Department of AI&DS

Prepared By:

Prof. Alfiya Javeed
Assistant Professor,
(Dept of AI&DS)

INDEX

1. **Install Hadoop and Implement the following file management tasks in Hadoop:**

   Adding files and directories

   Retrieving files

   Deleting files and directories.

   Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS using one of the above command line utilities.

Step 1: Install Hadoop

1. **Prerequisites**:
   - Java Development Kit (JDK) 8 or later must be installed. Use `java -version` to check the installed version.
   - Python installed on your system.
2. **Download Hadoop**:
   - Download the Hadoop binary distribution from the Apache Hadoop website.
3. **Extract and Configure Hadoop**:
   - Extract the downloaded Hadoop tarball.
   - Edit the `core-site.xml` and `hdfs-site.xml` files in the `etc/hadoop` directory to configure Hadoop's file system and replication settings.
4. **Start Hadoop**:
   - Format the Name Node:
     ```
     hdfs namenode -format
     ```
   - Start Hadoop services:
     ```
     start-dfs.sh
     ```

Step 1: Install Hadoop

1. **Update the system** (for Ubuntu/Linux):
   sudo apt update && sudo apt upgrade –y
2. **Install Java** (Hadoop requires Java):
   sudo apt install openjdk-8-jdk -y
   java –version
3. **Download Hadoop** (example: Hadoop 3.3.1):
   wget https://downloads.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
4. **Extract and Move Hadoop**:

   tar -xvzf hadoop-3.3.1.tar.gz

   sudo mv hadoop-3.3.1 /usr/local/hadoop
5. **Configure Environment Variables**: Edit ~/.bashrc and add:

export HADOOP_HOME=/usr/local/hadoop

export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

Step 2: Install hdfs Python Library

1. Add Files and Directories

- Create a directory in HDFS:

  hdfs dfs -mkdir /user

  hdfs dfs -mkdir /user/hadoop

Upload a file

hdfs dfs -put /home/user/sample.txt /user/hadoop/

2. Retrieve Files

List files:

hdfs dfs -ls /user/hadoop/

Download a file from HDFS:

hdfs dfs -get /user/hadoop/sample.txt /home/user/

View file contents:

hdfs dfs -cat /user/hadoop/sample.txt

Delete Files and Directories

Delete a file:

hdfs dfs -rm /user/hadoop/sample.txt

Delete a directory:

hdfs dfs -rm -r /user/hadoop/

Install the hdfs library to interact with Hadoop Distributed File System (HDFS) in Python.

- pip install hdfs

```
from hdfs import InsecureClient

# HDFS Configuration

HDFS_URL = 'http://localhost:50070'  # Change this to your Hadoop Name Node URL

HDFS_USER = 'hadoop' # Replace with your Hadoop user

# Initialize HDFS Client

client = InsecureClient(HDFS_URL, user=HDFS_USER)

# 1. Adding Files and Directories

def add_file_to_hdfs(local_file_path, hdfs_file_path):

    try:

        client.upload(hdfs_file_path, local_file_path)

        print(f"File '{local_file_path}' successfully uploaded to '{hdfs_file_path}' in  HDFS.")

    except Exception as e:

        print(f"Error uploading file: {e}")

def create_hdfs_directory(hdfs_directory_path):

    try:

        client.makedirs(hdfs_directory_path)

        print(f"Directory '{hdfs_directory_path}' successfully created in HDFS.")

    except Exception as e:

        print(f"Error creating directory: {e}")

# 2. Retrieving Files

def retrieve_file_from_hdfs(hdfs_file_path, local_file_path):

    try:

   client.download(hdfs_file_path, local_file_path)
```

```
    print(f"File '{hdfs_file_path}' successfully downloaded to '{local_file_path}'.")
  except Exception as e:
    print(f"Error downloading file: {e}")
# 3. Deleting Files and Directories
def delete_hdfs_file_or_directory(hdfs_path):
  try:
  client.delete(hdfs_path, recursive=True)
  print(f"Path '{hdfs_path}' successfully deleted from HDFS.")
 except Exception as e:
    print(f "Error deleting path: {e}")
# Example Usage
if __name__ == "__main__":
```

2) Develop a MapReduce program to implement Matrix Multiplication using python

```
from mrjob.job import MRJob
from mrjob.step import MRStep
class MatrixMultiplication(MRJob)
  def configure_args(self):
    super(MatrixMultiplication, self).configure_args()
    self.add_passthru_arg('--m', type=int, help="Number of rows in Matrix A")
   self.add_passthru_arg('--n', type=int, help="Number of columns in Matrix A / rows in Matrix B")
    self.add_passthru_arg('--p', type=int, help="Number of columns in Matrix B")
      def mapper(self, _, line):
          # Input format: MatrixID, i, j, value
          matrix_id, i, j, value = line.split(',')
          i, j, value = int(i), int(j), float(value)
          if matrix_id == 'A':  # Matrix A
```

```
            for k in range(self.options.p):
                # Emit key as (i, k) and value as ('A', j, value)
                yield (i, k), ('A', j, value)
        elif matrix_id == 'B':  # Matrix B
            for i in range(self.options.m):
                # Emit key as (i, k) and value as ('B', j, value)
                yield (i, j), ('B', i, value)
    def reducer(self, key, values):
        # Key: (i, k)
        # Values: list of ('A', j, value) or ('B', j, value)
        A_values = {}
        B_values = {}
        for value in values:
            if value[0] == 'A':
                A_values[value[1]] = value[2]
            elif value[0] == 'B':
                B_values[value[1]] = value[2]
        # Multiply and sum over shared dimension
        result = sum(A_values.get(j, 0) * B_values.get(j, 0) for j in
range(self.options.n))
        yield key, result
if __name__ == '__main__':
    MatrixMultiplication.run()
```

Input Format:

Each line of input represents one element of matrix AAA or BBB, formatted as:

```
MatrixID, i, j, value
```

- `MatrixID` is either `A` (for Matrix AAA) or `B` (for Matrix BBB).
- i,ji, ji,j are the row and column indices (starting from 0).
- `value` is the element at position (i,j)(i, j)(i,j).

Example Input:

Matrix AAA (2x3):

```
A,0,0,1
A,0,1,2
A,0,2,3
A,1,0,4
A,1,1,5
A,1,2,6
```

Matrix BBB (3x2):

```
B,0,0,7
B,0,1,8
B,1,0,9
B,1,1,10
B,2,0,11
B,2,1,12
```

Command to Run:

python matrix_multiplication.py input.txt --m 2 --n 3 --p 2

Output:

Resultant Matrix CCC (2x2):

(0, 0) 58.0
(0, 1) 64.0
(1, 0) 139.0
(1, 1) 154.0

This implementation uses the MapReduce paradigm to distribute the computation of matrix multiplication across multiple nodes effectively.

3) Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.

Problem Statement:

The program processes a weather dataset containing information such as temperature, humidity, and wind speed for different timestamps in a day. It computes daily statistics (e.g., average temperature, maximum wind speed) and generates weather condition messages based on thresholds.

```
from mrjob.job import MRJob
from mrjob.step import MRStep
class WeatherDataAnalysis(MRJob):
    def mapper(self, _, line):
        """
        Map step: Process weather data and emit key-value pairs.
        Input line format: date,time,temperature,humidity,wind_speed
        Example: 2025-01-28,12:00,22.5,65,10.2
        """
        try:
            date, time, temperature, humidity, wind_speed = line.split(',')
            temperature = float(temperature)
            humidity = float(humidity)
            wind_speed = float(wind_speed)
            # Emit the date as the key and weather parameters as the value
            yield date, (temperature, humidity, wind_speed)
        except ValueError:
            # Skip lines with invalid data
            pass
    def reducer(self, date, values):
        """
        Reduce step: Compute daily statistics and generate weather messages.
        """
        total_temp = 0
        total_humidity = 0
        max_wind_speed = float('-inf')
        count = 0
        for temp, humidity, wind_speed in values:
            total_temp += temp
            total_humidity += humidity
            max_wind_speed = max(max_wind_speed, wind_speed)
```

```
            count += 1
        # Compute averages
        avg_temp = total_temp / count
        avg_humidity = total_humidity / count
        # Determine weather condition messages
        if avg_temp > 35:
            temp_message = "Hot day"
        elif avg_temp < 15:
            temp_message = "Cold day"
        else:
            temp_message = "Moderate temperature"
        if avg_humidity > 80:
            humidity_message = "High humidity"
        elif avg_humidity < 30:
            humidity_message = "Low humidity"
        else:
            humidity_message = "Comfortable humidity"
        if max_wind_speed > 20:
            wind_message = "Windy day"
        else:
            wind_message = "Calm day"
        # Combine the messages
        weather_summary = f"{temp_message}, {humidity_message}, {wind_message}"
        yield date, weather_summary
if __name__ == '__main__':
    WeatherDataAnalysis.run()
```

Input Format:

Each line of input represents weather data for a specific timestamp, formatted as:

date, time, temperature, humidity, wind_speed.

- `Date`: Date in the format `YYYY-MM-DD`.
- `Time`: Time in `HH:MM` format.

- `Temperature`: Temperature in degrees Celsius.
- `Humidity`: Humidity as a percentage.
- `Wind_speed`: Wind speed in km/h.

Example Input:

2025-01-28,08:00,22.5,65,10.2

2025-01-28,12:00,30.0,70,15.0

2025-01-28,18:00,25.0,80,18.0

2025-01-28,23:00,20.0,85,5.0

2025-01-29,08:00,10.0,90,25.0

2025-01-29,12:00,15.0,95,30.0

2025-01-29,18:00,12.0,85,20.0

Command to Run:

python weather_data_analysis.py weather_data.txt

Output:

The program outputs daily weather summaries based on the input data.

"2025-01-28" "Moderate temperature, Comfortable humidity, Calm day"

"2025-01-29" "Cold day, High humidity, Windy day"

Key Features:

1. **Daily Aggregation**: Processes data for each day separately and computes average temperature, average humidity, and maximum wind speed.
2. **Threshold-Based Messages**:
   - Hot day: Average temperature > 35°C.
   - Cold day: Average temperature < 15°C.
   - Moderate temperature: Otherwise.
   - High humidity: Average humidity > 80%.
   - Low humidity: Average humidity < 30%.
   - Windy day: Maximum wind speed > 20 km/h.
   - Calm day: Otherwise.

This program efficiently processes weather data to generate insightful daily summaries using the MapReduce paradigm.

4) Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.

Problem Statement:

Given the Movie Lens dataset, where each line contains information about a movie and its associated tags, the program finds all tags associated with each movie.

Program:

```
from mrjob.job import MRJob
class MovieTags(MRJob):
    def mapper(self, _, line):
        """
        Map step: Emit movie ID as the key and tag as the value.
        Input line format: movieId,tag,timestamp
        Example: 1,funny,1139045764
        """
        try:
            # Split the line into components
            movie_id, tag, timestamp = line.split(',')
            # Emit movie ID and tag
            yield movie_id, tag
        except ValueError:
            # Skip invalid lines
            pass
    def reducer(self, movie_id, tags):
        """
        Reduce step: Aggregate all tags for a movie.
        """
        # Collect unique tags for the movie
        unique_tags = set(tags)
        yield movie_id, list(unique_tags)
if __name__ == '__main__':
    MovieTags.run()
```

Input Format:

Each line in the input file represents one tag associated with a movie, formatted as:

movieId, tag, timestamp.

- `movieId`: Unique identifier for the movie.
- `tag`: Tag associated with the movie.
- `timestamp`: Unix timestamp indicating when the tag was added.

Example Input:

1,funny,1139045764

1,romantic,1139045765

2,action-packed,1139045766

1,funny,1139045767

2,intense,1139045768

3,thrilling,1139045769

Example Output:

The output lists each movie along with its unique associated tags.

"1" ["funny", "romantic"]

"2" ["action-packed", "intense"]

"3" ["thrilling"]

Explanation:

1. **Mapper**:
   - Reads each line and emits the movie ID (movieId) as the key and the tag as the value.
2. **Reducer**:
   - Aggregates all tags for each movie ID.
   - Uses a set to ensure that each tag is listed only once for a movie.
3. **Final Output**:
   - Each movie ID is paired with a list of unique tags.

5) Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB

1 . Setup MongoDB in Python

Install `pymongo` if you haven't:

pip install pymongo

Then, connect to MongoDB:

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")

db = client["mydatabase"]

collection = db["mycollection"]
```

2. Count Documents

Counts the number of documents in a collection.

```
db.collection.countDocuments({});
```

Example: Count total users

```
db.users.countDocuments({});
```

You can also count with a filter:

```
count_filtered = collection.count_documents({"status": "active"})

print(f"Active Users: {count_filtered}")
```

3. Sort Documents

Sorts documents in ascending (`1`) or descending (`-1`) order.

```
db.collection.find().sort({ field: 1 }); // Ascending order

sorted_docs = collection.find().sort("age", 1)  # Sort by age (ascending)

for doc in sorted_docs:

    print(doc)

db.collection.find().sort({ field: -1 }); // Descending order

sorted_docs = collection.find().sort("age", -1)  # Sort by age (descending)
```

4. Limit Function

Limit the number of results returned:

```
limited_docs = collection.find().limit(5)  # Get only 5 documents
for doc in limited_docs:
  print(doc)
```

5. Skip Function

Skip the first `n` records:

```
skipped_docs = collection.find().skip(5)  # Skip first 5 records
for doc in skipped_docs:
 print(doc)
```

6. Aggregation

Aggregation allows complex queries like grouping, filtering, and calculations.

Example: Count users by status

```
pipeline = [
   {"$group": {"_id": "$status", "count": {"$sum": 1}}}
]
results = collection.aggregate(pipeline)
for result in results:
  print(result)
```

Example: Get the average age of users

```
pipeline = [
   {"$group": {"_id": None, "average_age": {"$avg": "$age"}}}
]
result = list(collection.aggregate(pipeline))
print(f"Average Age: {result[0]['average_age']}")
```

6) Develop Pig Latin scripts to sort, group, join, project, and filter the data.

Sample Data (data.txt)

We assume the dataset is a tab-separated file with the following columns:

```
id  name    age   department  salary
1   Alice   25    HR          5000
2   Bob     30    IT          7000
3   Charlie 35    IT          6000
4   David   28    HR          8000
5   Eve     40    Finance     7500
```

1 Load Data in Pig

```
pig
data = LOAD 'data.txt' USING PigStorage('\t')
      AS (id:int, name:chararray, age:int, department: char array, salary:int);
```

2 Sort Data

Sort employees by salary in descending order.

```
sorted_data = ORDER data BY salary DESC;
DUMP sorted_data;
```

Output (Sorted by Salary Descending)

```
4   David   28    HR          8000
5   Eve     40    Finance     7500
2   Bob     30    IT          7000
3   Charlie 35    IT          6000
1   Alice   25    HR          5000
```

3 Group Data

Group employees by department.

```
grouped_data = GROUP data BY department;
DUMP grouped_data;
```

Output (Grouped by Department)

```
(HR, {(1, Alice, 25, HR, 5000), (4, David, 28, HR, 8000)})
(IT, {(2, Bob, 30, IT, 7000), (3, Charlie, 35, IT, 6000)})
(Finance, {(5, Eve, 40, Finance, 7500)})
```

4 Join Data

Suppose we have another file departments.txt with department details:

mathematica

```
HR      Human Resources
IT      Information Technology
Finance Financial Services
```

Load the department details:

```
dept_data = LOAD 'departments.txt' USING PigStorage('\t') AS (dept_code:chararray,
dept_name:chararray);
```

Perform the join operation:

```
joined_data = JOIN data BY department, dept_data BY dept_code;
DUMP joined_data;
```

Output (Joined with Department Names)

```
1   Alice   25   HR   5000    HR    Human Resources
2   Bob     30   IT   7000    IT    Information Technology
3   Charlie 35   IT   6000    IT    Information Technology
4   David   28   HR   8000    HR    Human Resources
5   Eve     40   Finance 7500  Finance  Financial Services
```

5. Project (Select Specific Columns)

Select only name and salary fields.

```
projected_data = FOREACH data GENERATE name, salary;
DUMP projected_data;
```

Output (Only Name & Salary)

```
(Alice, 5000)
(Bob, 7000)
(Charlie, 6000)
(David, 8000)
(Eve, 7500)
```

6. Filter Data

Filter employees with a salary greater than 6000.

```
filtered_data = FILTER data BY salary > 6000;
```

```
DUMP filtered_data;
```

Output (Filtered Employees with Salary > 6000)

```
2  Bob    30  IT   7000
4  David  28  HR   8000
5  Eve    40  Finance 7500
```

7) Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

Create a Database

```
CREATE DATABASE company_db;
```

Use a Database

```
USE company_db;
```

Alter a Database (Change Properties)

```
ALTER DATABASE company_db SET DBPROPERTIES ('owner'='admin', 'created_by'='hive_user');
```

Drop a Database

```
DROP DATABASE company_db CASCADE;
```

2. Working with Tables

Create a Table

```
CREATE TABLE employees (
 id INT,
 name STRING,
 age INT,
   salary DOUBLE
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Alter a Table

Add a Column

ALTER TABLE employees ADD COLUMNS (department STRING);

Rename a Column

ALTER TABLE employees CHANGE COLUMN age emp_age INT;

Rename Table

ALTER TABLE employees RENAME TO emp_data;

Drop a Table

DROP TABLE emp_data;

3. Working with Views

Create a View

CREATE VIEW employee_salaries AS
SELECT name, salary FROM employees WHERE salary > 60000;

Alter a View (Replace with New Query)

ALTER VIEW employee_salaries AS
SELECT name, salary FROM employees WHERE salary > 50000;

Drop a View

DROP VIEW employee_salaries;

4. Working with Functions

Create a User-Defined Function (UDF)

Hive allows custom functions using Java.

Register a UDF (Example: UPPERCASE)

CREATE FUNCTION to_upper AS 'org.apache.hadoop.hive.ql.udf.generic.GenericUDFUpper'
USING JAR 'hdfs:///user/hive/lib/hive-udfs.jar';

Drop a Function

DROP FUNCTION to_upper;

5. Working with Indexes

Create an Index

CREATE INDEX emp_index ON TABLE employees (name)
AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
WITH DEFERRED REBUILD;

Alter an Index (Rebuild it)

ALTER INDEX emp_index ON employees REBUILD;

Drop an Index

DROP INDEX emp_index ON employees;

8) Implement a word count program in Hadoop and Spark.

Hadoop MapReduce Word Count:

1. **Mapper Class:** This will split lines into words and emit each word with a count of 1.
2. **Reducer Class:** This will sum up the counts for each word.

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCount {
   public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{
      private final static IntWritable one = new IntWritable(1);
```

```
        private Text word = new Text();
        public void map(Object key, Text value, Context context) throws IOException,
    InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
               word.set(itr.nextToken());
               context.write(word, one);
            }
        } }
      public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
    IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
               sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
     }
      public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
    }
```

Compile and run with Hadoop:

hadoop jar WordCount.jar WordCount /input/path /output/path

Spark Word Count:

```
from pyspark import SparkContext
```

```
sc = SparkContext("local", "WordCount")
text_file = sc.textFile("/input/path")
counts = (text_file
        .flatMap(lambda line: line.split(" "))
        .map(lambda word: (word, 1))
        .reduceByKey(lambda a, b: a + b))
counts.saveAsTextFile("/output/path")
```

Run with Spark:

```
spark-submit wordcount.py
```

9) Use CDH (Cloudera Distribution for Hadoop) and HUE (Hadoop User Interface) to analyze data and generate reports for sample datasets

1. Setup: Ensure CDH & HUE Are Running

Before you begin, make sure your CDH cluster and HUE are up and running:

- Access **Cloudera Manager** at http://<your-cloudera-manager-ip>:7180
- Start necessary services:
  - HDFS
  - YARN
  - Hive, Impala (for SQL-based queries)
  - Hue (accessible at http://<hue-ip>:8888)

2. Uploading a Sample Dataset

You can use any structured dataset, such as a CSV file containing sales data. Example file: **sales_data.csv**

```
OrderID,Product,Category,Amount,Date
1,Mobile,Electronics,200,2024-01-10
2,Laptop,Electronics,800,2024-01-11
3,Shirt,Clothing,50,2024-01-12
4,TV,Electronics,500,2024-01-13
5,Shoes,Footwear,100,2024-01-14
```

Upload Data to HDFS:

1. Open **HUE** (http://<hue-ip>:8888).
2. Navigate to **File Browser → /user/hadoop**.

3. Click **Upload** and select sales_data.csv.

3. Creating a Table in Hive

HUE allows running Hive queries directly.

Steps to Create a Hive Table:

1. Open **HUE → Query Editors → Hive**.
2. Run the following query:

```
CREATE TABLE sales_data (
    OrderID INT,
    Product STRING,
    Category STRING,
    Amount INT,
    Date STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

3. Load the data into Hive:

```
LOAD DATA INPATH '/user/hadoop/sales_data.csv' INTO TABLE sales_data;
```

4. Verify the table:

```
SELECT * FROM sales_data LIMIT 10;
```

4. Querying & Analyzing Data

Now that the data is in Hive, you can analyze it.

Example Queries

1. Total Sales Per Category

```
SELECT Category, SUM(Amount) AS Total_Sales
FROM sales_data
GROUP BY Category;
```

2. Top 3 Selling Products

```
SELECT Product, SUM(Amount) AS Revenue
FROM sales_data
GROUP BY Product
ORDER BY Revenue DESC
```

```
LIMIT 3;
```

3. Daily Sales Report

```
SELECT Date, SUM(Amount) AS Total_Revenue
FROM sales_data
GROUP BY Date
ORDER BY Date;
```

5. Generating Reports & Visualizations

Using HUE Dashboards

1. **Go to Hue → Dashboards.**
2. Click **"New Dashboard"** and select **"Hive"** as the data source.
3. Create visualizations:
    - **Bar Chart** for **Total Sales Per Category**.
    - **Pie Chart** for **Top Selling Products**.
    - **Line Chart** for **Sales Trends Over Time**.

6. Exporting Reports

- HUE allows exporting query results:
    - Click **"Download"** and select **CSV, JSON, or Excel**.
- You can also schedule queries to run automatically and email reports.

VIVA QUESTIONS

1. **What is Big Data?**
   Big Data is a collection of large and complex semi-structured and unstructured data sets that have the potential to deliver actionable insights using traditional data management tools
2. **Mention the 5 Vs of big data.**
   The **5 Vs of big data** are:
   a. Volume b. Velocity c. Variety d. Veracity e. Value:
3. **What is MapReduce?**
   MapReduce is a programming model or pattern within the Hadoop framework that is used to access big data stored in the Hadoop File System (HDFS). It is a core component, integral to the functioning of the Hadoop framework.
4. **What is Hadoop used for?**
   Hadoop is an open source framework based on Java that manages the storage and processing of large amounts of data for applications. Hadoop uses distributed storage and parallel processing to handle big data and analytics jobs, breaking workloads down into smaller workloads that can be run at the same time.
5. **When to use MapReduce with Big Data.**
   MapReduce is suitable for iterative computing involving massive amounts of data that must be processed in parallel. It is also appropriate for large-scale graph analysis.
6. **Mention the core methods of Reducer.**
   There are three core methods of a reducer:
   setup() - Configure various parameters such as distributed cache, heap size, and input data.
   reduce() - A parameter called once per key with the relevant reduce task.
   cleaning() - Clears all temporary files and is only executed at the end of a reducer task.
7. **Mention the core components of Hadoop.**
   Hadoop is a framework for storing and managing large amounts of data that uses distributed storage and parallel processing. There are three components of Hadoop:
   Hadoop HDFS (Hadoop Distributed File System), Hadoop MapReduce, Hadoop YARN
8. **What is HDFS, and why is it important?**

Hadoop Distributed File System (HDFS) is a key part of big data systems, built to store and manage large amounts of data across multiple nodes. It works by dividing large datasets into smaller blocks and distributing them across a cluster of nodes.

9. What are the common tools for machine learning in big data?

Common tools include:

- **Spark MLlib**: For distributed data processing and model training.
- **H2O.ai**: For scalable machine learning and AI applications.
- **TensorFlow** and **PyTorch**: For deep learning with GPU/TPU support.
- **Scikit-learn:** For smaller datasets integrated into larger pipelines.

10. What are the components of the Hadoop ecosystem?

The Hadoop Ecosystem includes:

- **HDFS:** Distributed storage for large datasets.
- **YARN:** Resource management and task scheduling.
- **MapReduce:** Data processing framework.
- **Hive:** SQL-like querying for structured data.
- **Pig:** Scripting for semi-structured data.
- **HBase:** NoSQL database for real-time analytics.

11. What is Hive used for?

Hive is a data warehouse system that is used to query and analyze large datasets stored in the HDFS. Hive uses a query language called Hive QL, which is similar to SQL. As seen from the image below, the user first sends out the Hive queries.

12. What are mapper and reducer in Hadoop?

Mappers and Reducers are the Hadoop servers that run the Map and Reduce functions respectively. It doesn't matter if these are the same or different servers.

13. What is Spark in Hadoop?

In a typical Hadoop implementation, different execution engines are also deployed such as Spark, Tez, and Presto. Spark is an open source framework focused on interactive query, machine learning, and real-time workloads.

14. Why do we use Spark?

It has a thriving open-source community and is the most active Apache project at the moment. Spark provides a faster and more general data processing platform. Spark lets you run programs up to 100x faster in memory, or 10x faster on disk, than Hadoop.

15. What is YARN, and how does it enhance Hadoop?

YARN (Yet Another Resource Negotiator) is Hadoop's resource management layer, enabling multiple applications to run on a Hadoop cluster simultaneously. It decouples resource management from data processing, enabling scalability and cluster utilization.

16. What is Apache HIVE

Apache Hive is a data warehouse and an ETL tool which provides an SQL-like interface between the user and the Hadoop distributed file system (HDFS) which integrates Hadoop.

17. How does Spark store data?

For data storage, Apache Spark uses the HDFS file system. It is compatible with any Hadoop data source, including HDFS, HBase, Cassandra, and Amazon S3.

18. What is spark architecture?

The spark architecture is a framework-based open-source component that helps process massive amounts of semi-structured, unstructured, and structured data for easy analysis. The data may then be used in Apache Spark.

19. What are HDFS and YARN?

Hadoop Distributed File System (HDFS) or HDS is the primary storage component of Hadoop. It uses blocks to store many forms of data in a distributed environment, and it follows the master and slave topology.

20. What are YARN

Yarn: Yet Another Resource Negotiator, or YARN, is the program's execution system that improves MapReduce (MR). YARN is used for scheduling, queuing, and execution management systems and organizes the executions within the containers.

21. What is Apache PIG

Pig Represents Big Data as data flows. Pig is a high-level platform or tool which is used to process the large datasets. It provides a high-level of abstraction for processing over the MapReduce. It provides a high-level scripting language, known as *Pig Latin* which is used to develop the data analysis codes.

22. Types of Data Models in Apache Pig:

It consist of the 4 types of data models as follows:

- **Atom**: It is a atomic data value which is used to store as a string.
- **Tuple**: It is an ordered set of the fields.
- **Bag**: It is a collection of the tuples.
- **Map**: It is a set of key/value pairs.

23. What is a MongoDB used for

MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.

24. What are the 4 basic operations in MongoDB?

The basic methods of interacting with a MongoDB server are called CRUD operations. CRUD stands for Create, Read, Update, and Delete.

25. What query language does MongoDB use?

MySQL, like many relational databases, uses structured query language (SQL) for access. MongoDB uses the MongoDB Query Language (MQL), designed for easy use by developers.

26. What is a NoSQL database?

The term NoSQL, short for "not only SQL," refers to non-relational databases that store data in a non-tabular format, rather than in rule-based, relational tables like relational databases do.

27. Why is Data Warehouse used?

Data Warehouse tool is used to store structured data while Hadoop is used to store unstructured data both can be used to store large amount of data.

28. What is HQL Hive query language?

What is Hive Query Language (HQL)? HQL is a SQL-like scripting language for querying and managing large datasets stored in Hadoop.

29. What are some use cases of HQL?

HQL is often used for data mining, log processing, text analytics, and customer behavior analysis.

30. What is a user-defined function in UDF?

A user-defined function (UDF) lets you create a function by using a SQL expression or JavaScript code. A UDF accepts columns of input, performs actions on the input, and returns the result of those actions as a value.