Sri Eshwar College of Engineering
Coimbatore | Tamilnadu
An Autonomous Institution
Affiliated to Anna University, Chennai

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

U23AM592–LARGE SCALE DATA ENGINEERING LABORATORY

LAB RECORD

ACADEMIC YEAR 2025-2026

FIFTH SEMESTER

TABLE OF CONTENTS

U23AM592–LARGE SCALE DATA ENGINEERING **L T P J C**

2 0 2 0 3

COURSE OBJECTIVES:

1. Understand the fundamentals of distributed computing, inter-process communication, and data processing paradigms.
2. Gain in-depth knowledge of the Hadoop ecosystem including HDFS, MapReduce, and tools like Pig, Hive, and HBase.
3. Develop skills in data transformation, analytics, and engineering using Spark and Databricks.
4. Learn how to model and query data in NoSQL databases like MongoDB and Redis.
5. Explore real-time and batch processing using Delta Lake and understand the Data Lakehouse concept.

LIST OF EXPERIMENTS: **30**

Periods

1. Implement Remote Procedure Call and Mutual Exclusion program (CO1)
2. Setup Hadoop in standalone and pseudo distributed mode (CO2)
3. Implement Hadoop File Operations, Map reduce jobs (Word Count) (CO2)
4. Implement basic operations such as load, filter, foreach, group, order, limit using Pig Latin Scripts (CO3)
5. Analyze web logs or sales data using Pig. (CO3)
6. Create and manipulate data using HiveQL queries (CO3)
7. Implement Data Exploration with Spark SQL (CO4)
8. Implement Movie recommendation system using Mllib (CO4)
9. Create a real-time leaderboard using Redis (CO5)
10. Create a document store with MongoDB and run CRUD operations (CO5)

COURSE OUTCOMES:

1. Apply the core concepts of distributed computing and big data processing frameworks.
2. Perform data ingestion, processing, and storage using Hadoop components like HDFS and MapReduce.
3. Implement data transformation, querying, and analytics using Pig, Hive, and HBase.
4. Analyze large-scale datasets using PySpark and Delta Lake on Databricks.

   Design and query NoSQL databases and distinguish between different NoSQL types and use cases.

SOFTWARE REQUIREMENTS

Language: Java JDK 8/ Python / C/C++

OS: Ubuntu 20.04+ / Windows 10 with WSL

Software:: Apache Hadoop 3.x, Apache Pig 0.17+, MongoDB, pymongo / Compass

Tools: Eclipse

QUALITY POLICY

To establish a system of Quality Enhancement, which would on a continuous basis evaluate and enhance the quality of teaching – learning, research and extension activities of the institution, leading to improvements in all processes, enabling the institution to attain excellence.

INSTITUTION VISION

To be recognized as a premier institution, grooming students into globally acknowledged engineering professionals.

INSTITUTION MISSION

- Providing outcome and value-based engineering education
- Nurturing research and entrepreneurial culture
- Enabling students to be industry ready and fulfil their career aspirations
- Grooming students through behavioural and leadership training programs
- Making students socially responsible

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

DEPARTMENT VISION

To become a model hub for Computer Science and Engineering-Artificial Intelligence and Machine Learning education and research by acquiring, disseminating, and generating knowledge in order to meet societal demands.

DEPARTMENT MISSION

- Evolve curriculum and delivery approaches to provide broad and wide exposure to the learner to gain adequate knowledge in their field of study.
- Provide opportunities for faculties to enhance their domain knowledge and skills required for the programs offered.
- Establish connections with local, national and global experts to share, utilize and exchange domain expertise.
- Conduct Outreach activities for the society that involve the use of artificial intelligence and machine learning solutions to deal with societal issues.

- Create and provide a conducive ecosystem and facilities for offering education related to artificial intelligence and machine learning.

PROGRAM EDUCATIONAL OBJECTIVES

- **PEO1:** Graduates will have successful careers in AI and ML-related fields, including positions such as data scientists, machine learning engineers, AI researchers, and consultants.
- **PEO2:** Graduates will recognize the dynamic nature of AI and ML technologies and engage in continuous learning to stay abreast of the latest advancements, tools, and techniques in the field.
- **PEO3:** Graduates will have the knowledge and skills to identify opportunities for applying AI and ML in various domains, potentially founding startups, or contributing to the growth and competitiveness of existing organizations.

PROGRAM OUTCOMES

- **PO1: Engineering Knowledge:** Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.
- **PO2: Problem Analysis:** Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)
- **PO3: Design/Development of Solutions:** Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)
- **PO4: Conduct Investigations of Complex Problems:** Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).
- **PO5: Engineering Tool Usage:** Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)
- **PO6: The Engineer and The World:**Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with

reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

- **PO7: Ethics:** Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)
- **PO8: Individual and Collaborative Team work:** Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.
- **PO9: Communication:** Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences
- **PO10: Project Management and Finance:** Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
- **PO11: Life-Long Learning:** Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)

PROGRAM SPECIFIC OUTCOMES

- **PSO1:** Apply Artificial Intelligence and Machine Learning techniques to design systems that support informed decision-making through data analysis and pattern recognition.
- **PSO2:** Integrate machine learning, deep learning, computer vision, and natural language processing with modern tools to develop effective solutions for complex data interpretation tasks.

Sri Eshwar College of Engineering
Coimbatore | Tamilnadu
An Autonomous Institution
Affiliated to Anna University, Chennai

DEPARTMENT OF CSE
(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of work done by

Name:Mr./Ms.....……Regi

ster No: ..... of third

year **B.E.CSE (Artificial Intelligence and Machine Learning)** in the **U23AM592 – LARGE SCALE DATA ENGINEERING** during the **fifth semester** of the academic year **2025 –2026 (Odd Semester).**

Signature of Faculty In-charge **Head of the Department**

Submitted for the practical examinations held on……

Internal Examiner **External Examiner**

Sri Eshwar College of Engineering
Coimbatore | Tamilnadu
An Autonomous Institution
Affiliated to Anna University, Chennai

List of Experiments

S. No	Date	Name of the Experiments	Page No	Marks (100)	Signature of Faculty
1.		Implement Remote Procedure Call and Mutual Exclusion program			
2.		Setup Hadoop in standalone and pseudo distributed mode			
3.		Implement Hadoop File Operations, Map reduce jobs (Word Count)			
4.		Implement basic operations such as load, filter, foreach, group, order, limit using Pig Latin Scripts			
5.		Analyze web logs or sales data using Pig.			
6.		Create and manipulate data using HiveQL queries			
7.		Implement Data Exploration with Spark SQL			
8.		Implement Movie recommendation system using MLlib			
9.		Create a real-time leaderboard using Redis			
10.		Create a document store with MongoDB and run CRUD operations			

Average Marks:

Average(in words): **Signature of the Faculty**

Ex No: 01 Date:	Implement Remote Procedure Call and Mutual Exclusion program

Aim: To implement a basic RPC client-server application, demonstrating how to call a function located on a remote machine as if it were a local function call.

Objectives:

1. Understand the fundamental concepts of RPC, including client-server interaction, stubs, marshalling, and unmarshalling.
2. Implement a simple RPC client that calls a remote procedure on a server.
3. Implement a server that hosts the remote procedure and handles requests from the client.

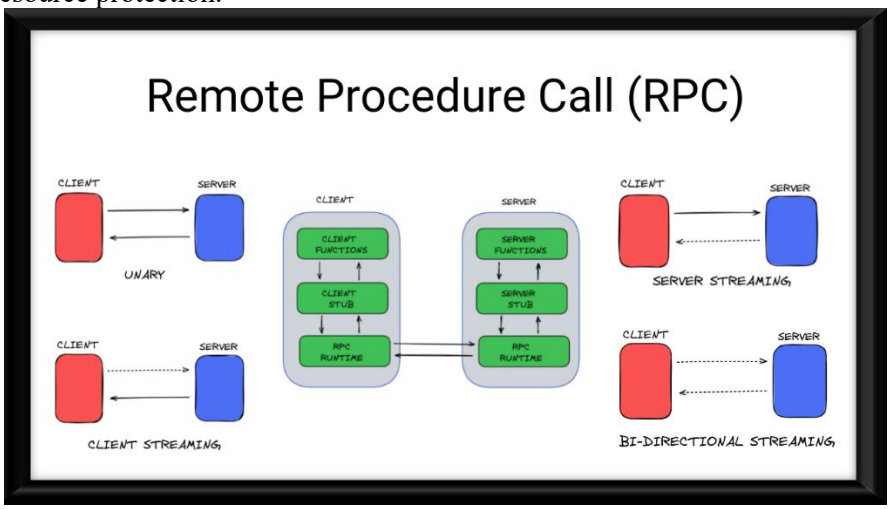
Background:

RPC is a technique for inter-process communication that allows a program to execute a subroutine in a different address space, such as on a remote computer, without explicit message-passing. It simplifies distributed application development by abstracting network complexities and uses a client-server model. Key components include client and server stubs which handle marshalling (packing parameters) and unmarshalling (unpacking parameters). An RPC runtime system manages network protocols and call data. RPC allows a program to cause a procedure to execute on another address space (commonly on another computer). It abstracts network communication like a local function call. A common pattern of communication used by application programs structured as a client/server pair is the request/reply message transaction: A client sends a request message to a server, and the server responds with a reply message, with the client blocking (suspending execution) to wait for the reply.

Key Points:

- Client-server model
- Stubs for function proxying
- Request-response mechanism

Implementing a combined Remote Procedure Call (RPC) and Mutual Exclusion (mutex) program involves creating a system where a client can request a service from a server, and the server ensures only one client accesses a shared resource at a time. This requires both RPC mechanisms for communication and mutexes for resource protection.



Mutual Exclusion

Mechanism that ensures that **only one process accesses a shared resource** at a time to prevent race conditions.

Algorithms:

- Centralized algorithm
- Distributed algorithm (Ricart-Agrawala)
- Token-based approaches

Conceptual Structure:

1. RPC Mechanism:

- The client sends a request to the server via RPC. This involves packing the request parameters into a message and sending it to the server's designated address.
- The server receives the request, unpacks the parameters, and executes the requested procedure.
- The server sends a response back to the client, again using RPC mechanisms.

2. Mutual Exclusion:

- The server's shared resource access is protected by a mutex.
- Before accessing the shared resource, the server acquires the mutex.
- After accessing the shared resource, the server releases the mutex.
- If another client requests access while the mutex is held, it will be blocked until the mutex is released.

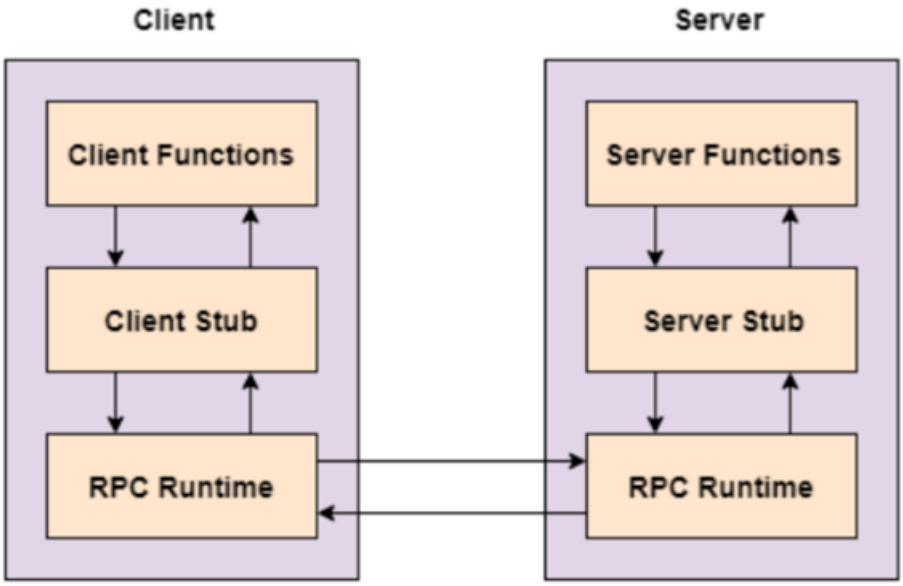
The sequence of events in a remote procedure call are given as follows:

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.

The message is sent from the client to the server by the client's operating system.

- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

The figure illustrates the basic interaction between the client and server in such an exchange.



Working of RPC

Stub: The stub is a client-side object that functions as a gateway. It is via which all outbound requests are routed. It is a client-side object that represents a distant object. The caller does the following duties when calling a method on the stub object:

1) It establishes a connection with a remote Virtual Machine (JVM), then writes and sends (marshals) the parameters to the remote Virtual Machine (JVM).
2) It sits and waits for the outcome. It reads (un-marshals) the return value or exception after receiving the result, and then returns the value to the caller.

Algorithm / Design Steps
Client-Server RPC with Mutual Exclusion
1.**Server Initialization**
o Listen for client connections
o Handle RPC requests
2.**Client Initialization**
o Connect to server
o Request critical section access
3.**Mutual Exclusion Mechanism**
o Implement token-based or Ricart-Agrawala algorithm
o Ensure one client at a time accesses critical section
4.**Communication**
o Use message-passing (TCP Sockets / RPC Framework)
o Handle ACKs and REPLYs
5.**Logging and Exit**
o Record access sequence
o Clean disconnection

Sample Program Code

Server-side file:
RPCServer.java

```
import java.util.*;
import java.net.*;
class RPCServer
{
DatagramSocket ds;
DatagramPacket dp;
String str,methodName,result;
int val1,val2;
RPCServer()
{
try
{
ds=new DatagramSocket(1200);
byte b[]=new byte[4096];
while(true)
{
dp=new DatagramPacket(b,b.length);
ds.receive(dp);
str=new String(dp.getData(),0,dp.getLength());
if(str.equalsIgnoreCase("q"))
{
System.exit(1);
}
else
{
StringTokenizer st = new StringTokenizer(str," ");
```

```
int i=0;
while(st.hasMoreTokens())
{
String token=st.nextToken();
methodName=token;
val1 = Integer.parseInt(st.nextToken());
val2 = Integer.parseInt(st.nextToken());
}
}
System.out.println(str); Remote Procedure Call
InetAddress ia = InetAddress.getLocalHost();
if(methodName.equalsIgnoreCase("add"))
{
result= "" + add(val1,val2);
}
else if(methodName.equalsIgnoreCase("sub"))
{
result= "" + sub(val1,val2);
}
else if(methodName.equalsIgnoreCase("mul"))
{
result= "" + mul(val1,val2);
}
else if(methodName.equalsIgnoreCase("div"))
{
result= "" + div(val1,val2);
}
byte b1[]=result.getBytes();
DatagramSocket ds1 = new DatagramSocket();
DatagramPacket dp1 = new
DatagramPacket(b1,b1.length,InetAddress.getLocalHost(), 1300);
System.out.println("result : "+result+"\n");
ds1.send(dp1);
}
}
catch (Exception e)
{
e.printStackTrace();
}
}
public int add(int val1, int val2)
{
return val1+val2;
}
public int sub(int val3, int val4)
{
return val3-val4;
}
public int mul(int val3, int val4)
{
return val3*val4;
}
public int div(int val3, int val4)
```

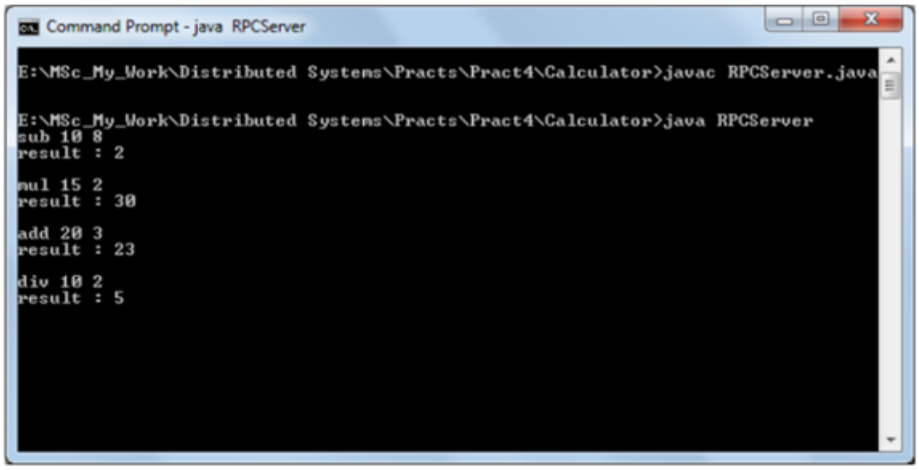
```
{
return val3/val4;
}
public static void main(String[] args)
{
new RPCServer();
}
}
```

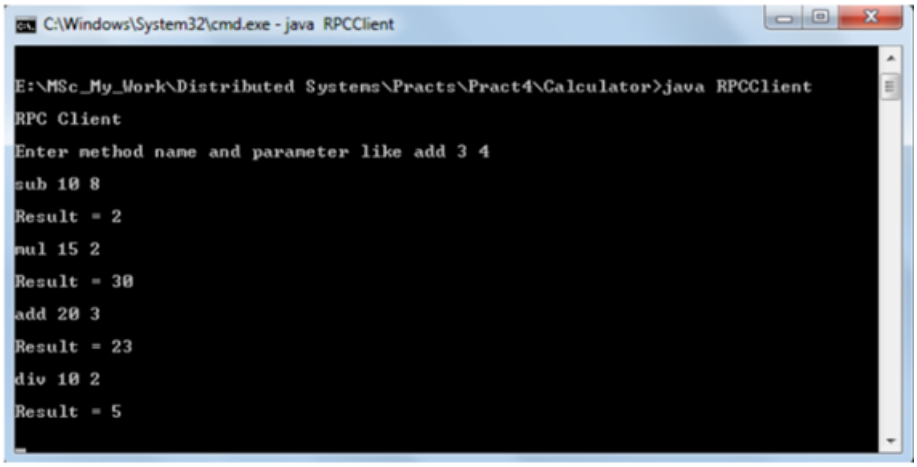
Client-side java file:

RPCClient.java

```
import java.io.*;
import java.net.*;
class RPCClient
{
RPCClient()
{
try
{
InetAddress ia = InetAddress.getLocalHost();
DatagramSocket ds = new DatagramSocket();
DatagramSocket ds1 = new DatagramSocket(1300);
System.out.println("\nRPC Client\n");
System.out.println("Enter method name and parameter like add 34\n");
while (true)
{
BufferedReader br = new BufferedReader(new Remote Procedure Call
InputStreamReader(System.in));
String str = br.readLine();
byte b[] = str.getBytes();
DatagramPacket dp = new
DatagramPacket(b,b.length,ia,1200);
ds.send(dp);
dp = new DatagramPacket(b,b.length);
ds1.receive(dp);
String s = new String(dp.getData(),0,dp.getLength());
System.out.println("\nResult = " + s + "\n");
}
}
catch (Exception e)
{
e.printStackTrace();
}
}
public static void main(String[] args)
{
new RPCClient();
}
}
```

Output:


Command Prompt - java RPCServer
E:\MSc_My_Work\Distributed Systems\Practs\Pract4\Calculator>javac RPCServer.java
E:\MSc_My_Work\Distributed Systems\Practs\Pract4\Calculator>java RPCServer
sub 10 8
result : 2
mul 15 2
result : 30
add 20 3
result : 23
div 10 2
result : 5


C:\Windows\System32\cmd.exe - java RPCClient
E:\MSc_My_Work\Distributed Systems\Practs\Pract4\Calculator>java RPCClient
RPC Client
Enter method name and parameter like add 3 4
sub 10 8
Result = 2
mul 15 2
Result = 30
add 20 3
Result = 23
div 10 2
Result = 5

Scenario 1: Basic RPC Communication

Suppose a client needs to compute the square of a number by calling a function on a remote server. How will RPC facilitate this?

Solution:
RPC abstracts the network communication. The client calls a local stub (e.g., square(n)), which marshals the request and sends it over the network to the server. The server's stub receives, unmarshals it, calls the actual square() function, and returns the result. The client gets the result as if it were a local call.

Scenario 2: RPC Failure Handling
In your RPC implementation, what happens if the server crashes after receiving the request but before sending a response?

Solution:
RPC systems may face at-most-once or at-least-once semantics. If the server crashes after processing but before responding:

At-most-once: The client retries, but no duplicate execution is guaranteed.

At-least-once: The client may resend the request, possibly leading to duplicate execution (unless handled with unique request IDs).

Scenario 3: RPC and Heterogeneous Systems

Can you use RPC between a Java client and a C server? How is data compatibility handled?

Solution:

Yes, but interoperability requires a standard interface definition language (IDL) like CORBA or gRPC with Protocol Buffers, which define data formats independent of programming languages. These tools handle serialization, data types, and conversion between different platforms.

Scenario 4: Asynchronous vs Synchronous RPC
Your client is blocked while waiting for a remote function result. How would you change this using asynchronous RPC?

Solution:
Asynchronous RPC allows the client to send a request and continue execution without blocking. When the response is ready, a callback or polling mechanism retrieves the result. This improves responsiveness, especially in GUI or real-time systems.

Scenario 5: Critical Section Handling
Suppose three distributed processes want to access a shared log file. How does mutual exclusion ensure correctness?

Solution:
Mutual exclusion ensures only one process enters the critical section (accessing the shared log) at a time. Algorithms like Ricart-Agrawala or token-based methods coordinate access using timestamps or tokens to prevent race conditions or data inconsistency.

Scenario 6: Ricart-Agrawala Example

Q6. In Ricart-Agrawala's algorithm, what if a process receives a request with a lower timestamp while it is in the critical section?

Solution:

It queues the request but does not reply immediately. Once it exits the critical section, it replies to all queued requests. This ensures timestamp-based ordering and prevents multiple entries into the critical section.

Pre-Lab Questions and Answers

1. What is a Remote Procedure Call (RPC)?

Answer:

A Remote Procedure Call is a protocol that allows a program to request a service or execute a function on another address space (commonly on a remote server or computer) as if it were a local procedure.

2. What are the key components of an RPC system?

Answer:

Client: Invokes the remote procedure.
Server: Provides the implementation of the remote procedure.
Stub: Acts as a proxy on both the client and server sides to marshal and unmarshal parameters.
RPC Runtime: Handles communication, request forwarding, and error handling.

3. What is Mutual Exclusion in distributed systems?

Answer:

Mutual Exclusion ensures that multiple processes or nodes do not enter the critical section (i.e., access shared resources) at the same time, preventing race conditions and data inconsistency.

4. Name and briefly describe any two mutual exclusion algorithms.

Answer:

Ricart-Agrawala Algorithm: A distributed algorithm using timestamped messages for granting access to the critical section.
Token Ring Algorithm: A logical ring where a token circulates, and a process must possess the token to enter the critical section.

5. Why is synchronization important in RPC-based distributed systems?

Answer:

Synchronization ensures that multiple RPC calls do not interfere with each other when accessing shared resources or executing time-critical operations.

6. What is marshalling and unmarshalling in RPC?

Answer:

Marshalling: Packing parameters into a standard format before sending across the network.
Unmarshalling: Extracting data from the received message to its original format.

Viva Questions

1. What is an RPC and how does it differ from a local procedure call?

2. Explain the role of stubs in RPC.

3. How does mutual exclusion help in distributed systems?

4. Name and explain any two mutual exclusion algorithms.

5. How does RPC handle failures?

6. What are the advantages and disadvantages of RPC?

7. Compare message passing vs RPC.

8. What is the Ricart-Agrawala algorithm?

9. Can RPC be asynchronous? If yes, how?

Post-Lab Questions

1. How would you modify the program to handle multiple clients simultaneously?

2. Can you implement a token ring mutual exclusion algorithm in this scenario?

3. How would you ensure fault tolerance in the RPC implementation?

4. List possible real-time applications of RPC in distributed systems.

5. Suggest improvements for performance optimization in your implementation.

Result: The program was successfully implemented using RPC with a mutual exclusion mechanism, ensuring only one client accessed the shared resource at a time.

Ex No: 02 Date:	Setup Hadoop in Standalone and Pseudo Distributed Mode

Aim: To perform setting up and Installing Hadoop in its operating modes: Standalone and Pseudo distributed.

Background:

Supported Platforms

GNU/Linux is supported as a development and production platform. Hadoop has been demonstrated on GNU/Linux clusters with 2000 nodes.

Required Software

Required software for Linux include:

1. Java™ must be installed. Recommended Java versions are described at https://cwiki.apache.org/confluence/display/HADOOP/Hadoop+Java+Versions
2. ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons if the optional start and stop scripts are to be used. Additionally, it is recommmended that pdsh also be installed for better ssh resource management.

Installing Software

If your cluster doesn't have the requisite software you will need to install it.
For example on Ubuntu Linux:
$ sudo apt-get install ssh
$ sudo apt-get install pdsh

Download

To get a Hadoop distribution, download a recent stable release from one of the Apache Download Mirrors at https://www.apache.org/dyn/closer.cgi/hadoop/common/

Prepare to Start the Hadoop Cluster

Unpack the downloaded Hadoop distribution. In the distribution, edit the file etc/hadoop/hadoop-env.sh to define some parameters as follows:

```
# set to the root of your Java installation
export JAVA_HOME=/usr/java/latest
```

Try the following command:

```
$ bin/hadoop
```

This will display the usage documentation for the hadoop script.

Now you are ready to start your Hadoop cluster in one of the three supported modes:

Local (Standalone) Mode

Pseudo-Distributed Mode

1. **Standalone Mode:**

   By default, Hadoop is configured to run in a non-distributed mode, as a single Java process. This is useful for debugging.

   The following example copies the unpacked conf directory to use as input and then finds and displays every match of the given regular expression. Output is written to the given output directory.

```
$ mkdir input
$ cp etc/hadoop/*.xml input
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.4.1.jar grep input output 'dfs[a-z.]+'
$ cat output/*
```

In Standalone Mode, Hadoop operates as a single monolithic Java process, beneficial for debugging and initial testing of MapReduce applications with small datasets. Here's a revised breakdown of the setup process:

1. Download Hadoop Binary Package:
   - Visit the Hadoop releases page and download the latest binary package.
   - Extract the downloaded package to a directory (e.g., C:\hadoop-3.1.2).
2. Set Environment Variables:
   - Set the following environment variables:
     - HADOOP_HOME: Path to the Hadoop installation directory (e.g., C:\hadoop-3.1.2).
     - HADOOP_BIN: Path to the Hadoop binary directory (e.g., C:\hadoop-3.1.2\bin).
     - JAVA_HOME: Path to your JDK installation (ensure it's version 1.8)
3. Edit PATH Environment Variable:
   - Add %HADOOP_HOME%, %HADOOP_BIN%, and %JAVA_HOME%\bin to your system's PATH variable.
4. Create Folders for Datanode and Namenode:
   - Inside the Hadoop installation directory (C:\hadoop-3.1.2), create an input folder.
   - Inside the input folder, create two subfolders: datanode and namenode.
5. Configure Hadoop:
   - Edit the Hadoop configuration files (e.g., core-site.xml, hdfs-site.xml, etc.) to specify the paths and settings.
6. Run Hadoop Jobs:
   - You can now run Hadoop jobs in Standalone Mode.

2. Pseudo-distributed Mode:

In Pseudo-Distributed Mode, Hadoop emulates a distributed environment on a single machine, running each daemon as a separate Java process. Hadoop software is installed on a Single Node, where various daemons of Hadoop run on the same machine as separate Java processes. These daemons include NameNode, DataNode, SecondaryNameNode, JobTracker, and TaskTracker, all operating within the confines of a single server.

1. Repeat Steps 1-4 from Standalone Mode.
2. Configure Hadoop:
   - Edit the Hadoop configuration files to set the appropriate values for the pseudo-distributed environment.
   - Specify localhost or 127.0.0.1 as the hostname for all daemons.
3. Start Hadoop Services:
   - Start the Hadoop daemons (Namenode, Datanode, ResourceManager, NodeManager, etc.) using the appropriate scripts.

Hadoop can also be run on a single-node in a pseudo-distributed mode where each Hadoop daemon runs in a separate Java process.

Configuration

Use the following:

etc/hadoop/core-site.xml:

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

etc/hadoop/hdfs-site.xml:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

Setup passphraseless ssh

Now check that you can ssh to the localhost without a passphrase:

```
$ ssh localhost
```

If you cannot ssh to localhost without a passphrase, execute the following commands:

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

Execution

The following instructions are to run a MapReduce job locally. If you want to execute a job on YARN, see YARN on Single Node.

1. Format the filesystem:
2. $ bin/hdfs namenode -format
3. Start NameNode daemon and DataNode daemon:
4. $ sbin/start-dfs.sh

The hadoop daemon log output is written to the $HADOOP_LOG_DIR directory (defaults to $HADOOP_HOME/logs).

5. Browse the web interface for the NameNode; by default it is available at:
    - NameNode - http://localhost:9870/
6. Make the HDFS directories required to execute MapReduce jobs:
7. $ bin/hdfs dfs -mkdir -p /user/<username>
8. Copy the input files into the distributed filesystem:
9. $ bin/hdfs dfs -mkdir input
10. $ bin/hdfs dfs -put etc/hadoop/*.xml input
11. Run some of the examples provided:
12. $ bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.4.1.jar grep input output 'dfs[a-z.]+'
13. Examine the output files: Copy the output files from the distributed filesystem to the local filesystem and examine them:
14. $ bin/hdfs dfs -get output output
15. $ cat output/*

or

View the output files on the distributed filesystem:

```
$ bin/hdfs dfs -cat output/*
```

16. When you're done, stop the daemons with:
17. $ sbin/stop-dfs.sh

YARN on a Single Node

You can run a MapReduce job on YARN in a pseudo-distributed mode by setting a few parameters and running ResourceManager daemon and NodeManager daemon in addition.

The following instructions assume that 1. ~ 4. steps of the above instructions are already executed.

1. Configure parameters as follows:

etc/hadoop/mapred-site.xml:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.application.classpath</name>

<value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/*</value>
  </property>
</configuration>
```

etc/hadoop/yarn-site.xml:

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.env-whitelist</name>

<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_HOME,PATH,LANG,TZ,HADOOP_MAPRED_HOME</value>
  </property>
</configuration>
```

2. Start ResourceManager daemon and NodeManager daemon:
3. $ sbin/start-yarn.sh
4. Browse the web interface for the ResourceManager; by default it is available at:
    - ResourceManager - http://localhost:8088/
5. Run a MapReduce job.
6. When you're done, stop the daemons with:

$ sbin/stop-yarn.sh

Scenario 1:

You are setting up Hadoop in standalone mode on a fresh Ubuntu system. After installing Java and Hadoop, you try to run a sample MapReduce job, but it fails with a "Permission Denied" error. What could be the cause and how would you resolve it?

Scenario 2:

You configured Hadoop and run a sample program. It runs fine, but you notice that it's not utilizing any daemons like name node, data node, resource manager. Why is this happening?

Scenario 3:

You want to quickly test a custom MapReduce job without deploying HDFS or YARN. Which mode of Hadoop should you use and why?

Scenario 4:

You installed Hadoop, but running hadoop version gives an error: JAVA_HOME is not set. What step did you likely miss during setup?

Scenario 5:

You've configured Hadoop in pseudo-distributed mode. When starting the NameNode with start-dfs.sh, you see a connection refused error for localhost:9000. What could be causing this? Which configuration file and property might be incorrect?

Scenario 6:

While formatting the NameNode using hdfs namenode -format, you get a permission denied error on the tmp directory. How do file permissions and ownership affect Hadoop? How should you configure them before formatting?

Pre-Lab Questions

1. What is Hadoop? Briefly explain its core components.

2. What are the different modes in which Hadoop can run?

3. What are the prerequisites for installing Hadoop?

4. What is the role of HDFS and MapReduce in Hadoop?

5. What is the difference between NameNode and DataNode?

6. Why is Java required for running Hadoop?

7. What are SSH and its role in the Hadoop setup?

8. What configuration files need to be edited for setting up pseudo-distributed mode?

9. What is the function of core-site.xml, hdfs-site.xml, and mapred-site.xml?

10. What is the importance of environment variables like JAVA_HOME and HADOOP_HOME?

Post-Lab Questions

1. What challenges did you face while installing Hadoop in standalone mode?

2. What differences did you observe between standalone and pseudo-distributed modes?

3. How did you verify that Hadoop was installed and working correctly?

4. What changes were made to hadoop-env.sh and why?

5. How does pseudo-distributed mode simulate a real cluster environment?

6. What are the advantages and limitations of running Hadoop in standalone mode?

7. How can you monitor Hadoop daemons after starting the services?

8. What is the command to format HDFS, and why is it needed?

9. Explain how SSH key-based authentication is set up for Hadoop nodes.

10. Describe the process of starting and stopping Hadoop services.

Viva Questions

1. What are the different modes of Hadoop operation?

2. Why do we use standalone mode?

3. What is the role of the Hadoop Distributed File System (HDFS)?

4. What are the configuration files in Hadoop?

5. What is the role of the Hadoop Distributed File System (HDFS)?

6. How is Hadoop different from traditional file systems?

7. What is the function of the JobTracker and TaskTracker in Hadoop 1.x?

8. What happens when you run a Hadoop job in standalone mode?

9. Can you list the default ports used by Hadoop services?

10. What are the benefits of using Hadoop for big data processing?

Result: Hadoop was successfully installed and tested in both Standalone Mode and Pseudo-Distributed Mode.

Ex No:03 Date:	Implement Hadoop File Operations, Map reduce jobs (Word Count)

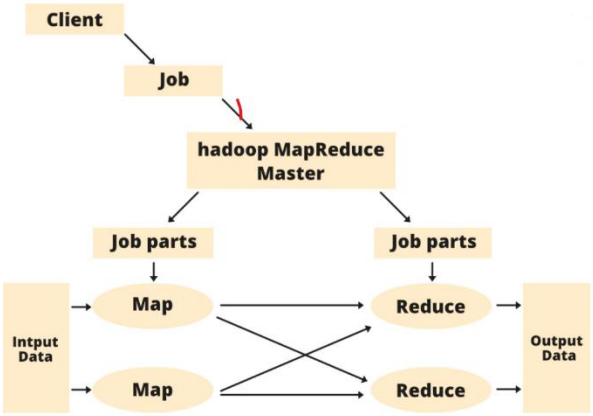
Aim:

To implement basic Hadoop Distributed File System (HDFS) operations and execute a MapReduce job to perform word count from a given text file using Hadoop framework.

Background:

MapReduce in Hadoop is a software framework for ease in writing applications of software, processing huge amounts of data. MapReduce provides the facility to distribute the workload (computations) among various nodes(analogous to commodity hardware). Hence, reducing the processing time as data on which the computation needs to be done is now divided into small chunks and individually processed. Through MapReduce you can achieve parallel processing resulting in faster execution of the job.
MapReduce Word Count is a framework which splits the chunk of data, sorts the map outputs and input to reduce tasks. A File-system stores the output and input of jobs. Re-execution of failed tasks, scheduling them and monitoring them is the task of the framework.

Figure below shows the architecture as well as working of MapReduce with an example:



MapReduce Architecture is the backbone of Hadoop's processing, offering a framework that splits jobs into smaller tasks, executes them in parallel across a cluster, and merges results. Its design ensures parallelism, data locality, fault tolerance, and scalability, making it ideal for applications like log analysis, indexing, machine learning, and recommendation systems.

Core Components of MapReduce Architecture

The MapReduce Architecture follows a master–slave model, where the Job Tracker (master) coordinates tasks and Task Trackers (slaves) execute them on cluster nodes. Below are its main components:

1. Client

The Client is the entry point into MapReduce. It submits the job for execution by packaging the Mapper and Reducer logic into a JAR file and specifying the input and output paths. After submission, the Client's role ends.

2. Job

A Job represents the complete processing request from the client.

Internally, it is divided into job parts (tasks).
Each job part is assigned to either the Map or Reduce phase.

3. Hadoop MapReduce Master
The Master Node coordinates job execution. It:

Accepts jobs from the Client.
Splits them into job parts.
Assigns these parts to the Map and Reduce tasks.
Tracks execution progress and reassigns failed tasks.

4. Job Parts (Tasks)
Every job is divided into smaller units called job parts.

Map job parts: Process input data splits into intermediate key–value pairs.
Reduce job parts: Aggregate intermediate results into final outputs.

5. Map Phase
Input data is split and given to Map tasks.
Each Map task processes its local split and produces intermediate results in the form of key–value pairs.

6. Shuffle & Sort (Between Map and Reduce)
Intermediate key–value pairs from the Map phase are grouped by key.
Sorting ensures ordered keys before passing them to the Reducers.

7. Reduce Phase
Reducers take grouped keys and values from Shuffle & Sort.
They aggregate them to produce the final output data, which is written back to HDFS.
Phases in MapReduce Architecture
The MapReduce model processes large datasets in two main phases—Map and Reduce-with an intermediate Shuffle & Sort stage that organizes data between them.

Map Phase
The input dataset is divided into splits, each processed by a Map Task on the node storing the data (ensuring data locality). A RecordReader converts raw input into (key, value) pairs, which the Mapper transforms into intermediate results (e.g., "Hello Hadoop" → (Hello, 1), (Hadoop, 1)). The Map Phase generates but does not aggregate data.

Shuffle & Sort Phase
After mapping, the intermediate outputs are reorganized so they can be processed efficiently by reducers. Shuffling groups identical keys, e.g., (Hadoop, 1), (Hadoop, 1), (Hadoop, 1) becomes (Hadoop, [1,1,1]). Sorting then arranges the keys in order. This stage ensures balanced distribution of work and is essential for linking the Map and Reduce phases.

Reduce Phase
Finally, reducers take the grouped keys and their associated values from the Shuffle & Sort stage. The reduce() function aggregates or summarizes them to produce the final output. For example, (Hadoop, [1,1,1]) becomes (Hadoop, 3). The consolidated results are written back into HDFS at the client-specified output location.

Execution Flow of MapReduce Architecture

Client submits the job includes JAR file, Mapper & Reducer logic, input path and output path.
Job Tracker accepts the job, splits it into tasks and assigns them to Task Trackers based on data locality.
Task Trackers execute Map Tasks on input splits stored locally.
Intermediate results are shuffled and sorted.
Reduce Tasks process grouped results and generate the final output.
Final output is written back to HDFS.
The design is optimized for data locality, which minimizes network transfer by moving computation close to where the data is stored.

Optimizations in MapReduce Architecture

Speculative Execution : If a task is running unusually slow, Hadoop launches a duplicate copy on another node. The task that finishes first is accepted, improving reliability and efficiency.
Fault Tolerance : If a task or node fails, the Job Tracker reassigns the task to another node that has a replica of the data.
Load Balancing : The Job Tracker distributes tasks across available Task Trackers to utilize the cluster efficiently.
Pipeline Execution : Multiple tasks (Map and Reduce) run concurrently across the cluster, ensuring maximum throughput.

Advantages of MapReduce Architecture
Parallelism : Tasks are executed simultaneously across hundreds or thousands of nodes.
Scalability : Handles increasing data volumes by simply adding more nodes to the cluster.
Fault Tolerance : Automatically recovers from task or node failures using replicated data.
Efficiency : Reduces network traffic by processing data locally on the nodes where it resides.
Simplicity : Developers only need to implement map() and reduce() functions; Hadoop handles distribution, fault tolerance and scheduling.
Splitting: The parameter of splitter can be anything. By comma, space, by a new line or a semicolon.

Mapping: This is done as explained below.

Shuffle/Intermediate splitting: The process is usually parallel on cluster keys. The output of the map gets into the Reducer phase and all the similar keys of data are aligned in a cluster.

Reducing: This is done as explained below. Final result — All the data is clustered or combined to show the together form of a result.

For understanding the working of MapReduce and its architecture we will implement our own Mapper and Reducer codes.

First, we will import our dataset into the HDFS (Hadoop Distributed File System).

The dataset can be a simple txt file with some words or sentences written in it.

Please follow these steps to import the dataset:

Step 1. Download and extract the compressed file of the text-based dataset from a source and save it as a txt file in your local storage.

Step 2. Create a directory in the Hadoop HDFS by using the following command:

hdfs fs -mkdir /input_wordCount
Step 3. Copy the txt file in the created directory from the local directory using -ls option. For example:

hdfs fs -put T:\Tirth\Documents\word_count.txt
Step 4. To check whether file is present in the directory, run the following command:

hdfs fs -ls /input_wordCount
Screenshot of running the ls command to check the presence of file
Fig. 1.1 Verifying the presence of file in HDFS directory.
Now we will prepare the source code for word count. For this purpose it is strongly suggested to use an IDE (such as Eclipse or IntelliJ) which allows to create an executable JAR file for your project. Here we are using Eclipse IDE.

Step 1. Create a Java Project and declare a package within it.

Step 2. Create a class in the package.

Step 3. Insert this code in the class file.

WordCount.java

```
package com.mapreduce.wc;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
  public static void main(String [] args) throws Exception
  {
    Configuration c=new Configuration();
    String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
    Path input=new Path(files[0]);
    Path output=new Path(files[1]);
    Job j=new Job(c,"wordcount");
    j.setJarByClass(WordCount.class);
    j.setMapperClass(MapForWordCount.class);
    j.setReducerClass(ReduceForWordCount.class);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(j, input);
    FileOutputFormat.setOutputPath(j, output);
    System.exit(j.waitForCompletion(true)?0:1);
  }
public static class MapForWordCount extends Mapper<LongWritable, Text, Text,
 IntWritable>{
  public void map(LongWritable key, Text value, Context con) throws
IOException, InterruptedException
  {
    String line = value.toString();
    String[] words=line.split(" ");
    for(String word: words )
    {
      Text outputKey = new Text(word.toUpperCase().trim());
      IntWritable outputValue = new IntWritable(1);
      con.write(outputKey, outputValue);
    }
  }
}
public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text,
IntWritable>
```

```
{
 public void reduce(Text word, Iterable<IntWritable> values, Context con)
throws IOException, InterruptedException
 {
  int sum = 0;
  for(IntWritable value : values)
  {
   sum += value.get();
  }
  con.write(word, new IntWritable(sum));
 }
}
}
```

The code above extends the predefined framework classes provided by Hadoop for Mapper and Reducer. For the current scenario we are only focusing on Mapping and Reducing phases.

Initially you will be prompted multiple error messages in the import statements involving apache.hadoop package. This is because we need to import the utility classes provided by Hadoop to implement MapReduce. This classes are compressed into executable jar files which are provided in the installation folder of Hadoop.

These jar files must be included to run any MapReduce task in Hadoop. It can be done by following steps:

Right Click on your project in the files navigator to the left.
Select Build Paths and click on Configure Project.
Head to Libraries Tab and select Add External Jar button.
In the pop-up window, navigate through the 'share' folder in your Hadoop installation directory.
You will notice some folders and JAR files.
Include those JARfiles add jar files from all the folders as well. [For help, refer this Medium Article]
Once, all the jar files are included, save and close the config window.
You will notice that all the errors which were prompted are now gone. If not then you need to delete a fie (probably a config file automatically created in the same package while creating the package).
After all the above steps are completed error-free, you need to create a JAR file of your project. To do this, Right click on you project and select Export.
Select JAR from Java folder section.
Browse the directory you want to save and give the name to the file.
The source code is now ready to be executed in the MapReduce environment.

We need to run the JAR file for which our input file will be the dataset txt file and output will be stored in another file. Our aim is to run the JAR file on the input file which is present in HDFS.

To do this, we have a command with following syntax:

hadoop jar <jar_path> <class_path> <input_file_path_in_hdfs> <output_file_path_in_hdfs>
Note that the JAR file need not to be uploaded to HDFS, we can directly connect it from our local storage. The class path includes package name and class name since classes reside in a package. Example:

hadoop jar T:\Tirth\Documents\MapReduceWordCount.jar com.mapreduce.wc/WordCount /input_wordCount /output_dir
After running this command, Mapper job start getting executed. After a while Reducer job starts consequently. The

details of the job are being displayed in the command prompt.

MapReduce job logs
Fig. 1.2 Snapshot of logs describing the running job of MapReduce
Once the execution is finished, you need to access the output using ‘cat’ command. Note that the output will be stored in the file mentioned in the above command by you and it will be in HDFS. To access the output, write the following command:

hadoop dfs -cat /output_dir/*
Output of word and its respective frequency
Fig. 1.3 Snapshot of the MapReduce output, word and its respective count.
As in Fig 1.3 we can see the format of output is Word Number. The number represents the total occurrence of that word in the entire text file. By using Map Reduce, we can reduce the computing time by a considerable amount. It is much faster than the traditional algorithm used in programming languages.

You can also access the application in Hadoop Dashboard. Head to a browser and open http://localhost:8088/clusters (make sure that all your Hadoop components are running).

Hadoop dashboard to show executed tasks.
Fig 1.4 shows highlighted field of Start Time and Finish Time, the difference of both the values equal to 23 seconds, which is considerably lower compared to traditional frequency counting algorithm applied on a big size file.

Scenario based Questions

Scenario 1:

You are given a dataset student_records.txt stored locally. Describe the Hadoop commands required to:

- Create a directory /education/data in HDFS,
- Upload the file,
- List the contents, and
- Verify file replication.

(Write the exact sequence of commands.)

Scenario 2:

A data engineer accidentally deleted a file from HDFS. Which command can recover it if the Trash feature is enabled? Explain the configuration needed to enable Trash.

Scenario 3:
Your HDFS cluster shows slow read operations. Explain how block size and replication factor impact read performance, using an example of a 2GB file stored with block size 128MB.

Pre Lab Questions

1. What is Hadoop?

Hadoop is an open-source framework developed by Apache for processing and storing large datasets in a distributed computing environment using simple programming models. It allows applications to work with thousands of nodes and petabytes of data efficiently.

2. What are the main components of the Hadoop ecosystem?

The two core components of Hadoop are:

HDFS (Hadoop Distributed File System): Provides distributed storage across multiple machines.

MapReduce: A programming model for parallel processing of large datasets.
Other ecosystem tools include Hive, Pig, Sqoop, Flume, and YARN (Yet Another Resource Negotiator).

3. What is HDFS and what are its key features?

HDFS (Hadoop Distributed File System) is a distributed file system that stores data across multiple nodes.
Key Features:

Fault tolerance through data replication

High throughput access to data

Scalability to handle petabytes of data

Support for write-once, read-many access pattern

4. What is the default block size in HDFS and why is it large?

The default block size in HDFS is 128 MB (in newer Hadoop versions; older versions used 64 MB).
It is large to minimize the overhead of managing metadata and to optimize throughput during large data reads.

5. What are the basic Hadoop file system commands?

Some commonly used HDFS commands are:

hdfs dfs -ls / → List files in HDFS directory

hdfs dfs -mkdir /user → Create a directory

hdfs dfs -put file.txt /input → Upload local file to HDFS

hdfs dfs -cat /output/part-r-00000 → Display output file content

hdfs dfs -get /output result.txt → Download from HDFS to local

6. What are the main phases in a MapReduce job?

A MapReduce job has three main phases:

Map Phase: Processes input data and generates key-value pairs.

Shuffle and Sort Phase: Transfers and groups data by key.

Reduce Phase: Aggregates or summarizes values for each key.

7. What is the role of Mapper and Reducer in WordCount example?

Mapper: Reads each line of text, splits it into words, and emits each word with a count of 1 → (word, 1).

Reducer: Receives all values associated with a word and sums them up to get the final count → (word, total_count).

Post Lab Questions

1. What are the main components of the Hadoop Distributed File System (HDFS)? Explain their roles in file storage and processing.

2. Differentiate between the Hadoop NameNode and DataNode in terms of their functionality.

3. Why does Hadoop use the concept of "block" storage? What is the default block size in HDFS and why?

4. What is the purpose of the replication factor in HDFS? How does it affect fault tolerance?

5. Define a MapReduce job. What are the key phases in the MapReduce execution pipeline

VIVA Questions:

1. What are the main components of the Hadoop ecosystem?

2. What is HDFS?

3. What is the purpose of the NameNode and DataNode?

4. What is the default block size in HDFS?

5. What are the basic file operations supported in HDFS?

Result:

Thus, the program was executed and the output was verified successfully.

Ex No: 04 Date:	Implement basic operations such as load, filter, foreach, group, order, limit using Pig Latin Scripts

Aim :

To perform basic data manipulation and analysis tasks in Apache Pig using Pig Latin commands such as:

- LOAD
- FILTER
- FOREACH
- GROUP
- ORDER
- LIMIT

Software/Tools Required:

- Hadoop (HDFS configured)
- Apache Pig (local or MapReduce mode)
- Sample dataset (e.g., student.csv)

Step 1: Start Hadoop and Pig

start-dfs.sh

start-yarn.sh

pig

You will enter the Pig Grunt shell:

grunt>

Step 2: Load the Dataset

student_data = LOAD '/pigdata/student.csv' USING PigStorage(',')

AS (student_id:int, name:chararray, age:int, department:chararray, marks:int);

Purpose: Loads CSV data into a Pig relation.

Step 3: Display Sample Data

DUMP student_data;

Output: Displays all tuples of the relation.

Step 4: Filter Students with Marks > 80

high_scorers = FILTER student_data BY marks > 80;

DUMP high_scorers;

Purpose: Filters records based on condition.

Step 5: Select Specific Columns

names_marks = FOREACH high_scorers GENERATE name, marks;

DUMP names_marks;

Purpose: Projects specific fields from the dataset.

Step 6: Group Data by Department

group_dept = GROUP student_data BY department;

DUMP group_dept;

Purpose: Groups tuples by department.

Step 7: Calculate Average Marks per Department

avg_marks = FOREACH group_dept GENERATE group AS department,
AVG(student_data.marks) AS avg_marks;

DUMP avg_marks;

Purpose: Performs aggregation on grouped data.

Step 8: Order Data by Marks

ordered_data = ORDER student_data BY marks DESC;

DUMP ordered_data;

Purpose: Sorts the dataset by marks in descending order.

Step 9: Limit the Output

top3 = LIMIT ordered_data 3;

DUMP top3;

Purpose: Displays only the top 3 records.

Step 10: Store the Final Result

STORE top3 INTO '/pigoutput/top_students' USING PigStorage(',');

Purpose: Writes the output to HDFS or local storage.

Sample Dataset: student_marks.csv

id	name	age	marks	gender
1	Arun	18	85	M
2	Priya	17	92	F
3	Karthi	18	67	M
4	Divya	19	75	F
5	Manoj	18	89	M

Assume this file is stored in HDFS at:
/user/pig/student_marks.csv

Pig Latin Script: student_operations.pig

```
-- Load the data from HDFS

student_data = LOAD '/user/pig/student_marks.csv'

USING PigStorage(',')

AS (id:int, name:chararray, age:int, marks:int, gender:chararray);

-- Display the loaded data

DUMP student_data;

-- Filter: Select students with marks greater than 80

high_scorers = FILTER student_data BY marks > 80;

DUMP high_scorers;

-- Foreach: Display name and marks of each student with some transformation

student_summary = FOREACH student_data GENERATE name, marks, (marks * 1.1) AS
marks_with_bonus;
```

```
DUMP student_summary;
-- Group: Group students by gender
group_by_gender = GROUP student_data BY gender;
DUMP group_by_gender;
-- For each group, calculate average marks
avg_marks_by_gender = FOREACH group_by_gender GENERATE
    group AS gender,
    AVG(student_data.marks) AS avg_marks;
DUMP avg_marks_by_gender;
-- Order: Arrange students by marks descending
ordered_students = ORDER student_data BY marks DESC;
DUMP ordered_students;
-- Limit: Display only top 3 students
top3_students = LIMIT ordered_students 3;
DUMP top3_students;
```

Explanation of Each Command

Pig Latin Command	Purpose
LOAD	Reads data from HDFS or local file system into a relation.
FILTER	Extracts tuples that meet a specific condition.
FOREACH ... GENERATE	Performs column selection and transformation.
GROUP	Groups tuples based on one or more fields.
AVG()	Built-in function used after grouping to calculate average.
ORDER	Sorts the relation by one or more fields.
LIMIT	Restricts the number of tuples returned.
DUMP	Displays the output on the console.

Execution Steps

1. Start Pig in MapReduce mode:
2. pig
3. Run the script:
4. pig -x mapreduce student_operations.pig
5. View results on console.

Output Summary

- **FILTER:** Students with marks > 80 → *Arun, Priya, Manoj*
- **FOREACH:** Shows names, marks, and marks_with_bonus
- **GROUP:** Groups by gender → {M: [Arun, Karthi, Manoj]}, {F: [Priya, Divya]}
- **ORDER:** Sorted by marks descending → *Priya, Manoj, Arun, Divya, Karthi*
- **LIMIT:** Top 3 → *Priya, Manoj, Arun*

Scenario based Questions:

Scenario 1: Sales Data Analysis

A file named sales.txt contains the following fields:
transaction_id, date, product_id, quantity, price
Example:
T1,2025-05-01,P01,3,200
T2,2025-05-01,P02,5,150
T3,2025-05-02,P01,2,200
Questions:

1. Write a Pig Latin script to load this file into Pig using the correct schema.
2. Filter only those transactions where quantity >

Scenario 2: Employee Dataset

File: employees.txt

emp_id, name, department, salary, location

Questions:

1. Load the file into Pig using the proper data types.
2. Filter the employees working in the "IT" department.

Scenario 3: Student Marks Analysis
File: students.txt
roll_no, name, subject, marks
Questions:

1. Load the data and display all records.
2. Filter students who scored more than 70.

PRELAB QUESTIONS

1. What is Apache Pig?

Apache Pig is a high-level data flow scripting language used with Hadoop to analyze large datasets. It provides an abstraction over MapReduce, allowing developers to write scripts in Pig Latin, which are internally converted into MapReduce jobs for execution on a Hadoop cluster.

2. What is Pig Latin?

Pig Latin is a data flow language used in Apache Pig to express data analysis programs. It consists of a series of transformations (like LOAD, FILTER, GROUP, JOIN, etc.) applied to input data. Each statement produces a dataset that can be used by subsequent statements.

3. What is the difference between Pig and Hive?

Feature	Pig	Hive
Language Type	Procedural (Pig Latin)	Declarative (SQL-like: HiveQL)
Data Flow	Explicit (step-by-step)	Implicit (focuses on what to do, not how)
Use Case	Complex data flows, ETL tasks	Querying structured data
Execution Engine	MapReduce / Tez / Spark	MapReduce / Tez / Spark

4. What is the purpose of the LOAD operation in Pig?

LOAD is used to read data from a file system (like HDFS or local filesystem) into a Pig relation (dataset).

Example:

```
data = LOAD 'student_data.txt' USING PigStorage(',') AS (id:int, name:chararray, marks:int);
```

This command loads a CSV file into a relation named data.

5. What does the FILTER operator do in Pig Latin?

FILTER removes unwanted records based on a given condition.

Example:

```
highScorers = FILTER data BY marks > 80;
```

This filters out students who scored more than 80 marks.

6. What is the use of the FOREACH operator?

FOREACH applies transformations or projections to each element of a dataset.

Example:

nameMarks = FOREACH data GENERATE name, marks;

This extracts only the name and marks fields from the dataset.

7. Explain the purpose of the GROUP operator.

GROUP collects records with the same key together. It is similar to SQL's GROUP BY.

Example:

groupedData = GROUP data BY department;

This groups all student records by their department.

8. What is the function of the ORDER operator in Pig?

ORDER sorts the data based on one or more fields.

Example:

sortedData = ORDER data BY marks DESC;

This sorts students in descending order of marks.

9. Why is the LIMIT operator used in Pig scripts?

LIMIT restricts the number of output records, useful for sampling or debugging.

Example:

topFive = LIMIT sortedData 5;

This returns only the first 5 records from the sorted dataset.

10. What is a relation in Pig Latin?

Answer:

A relation in Pig Latin is a collection of tuples (similar to a table in a database). Each relation is the output of a Pig Latin operation, and can be used as input for subsequent operations.

Viva / Post-Lab Questions:

1. What are the advantages of using Pig over traditional MapReduce programming?

2. How does FOREACH ... GENERATE differ from FILTER in Pig?

3. Explain the difference between GROUP and COGROUP.

4. How can Pig be integrated with HDFS and Hive?

5. What happens internally when a Pig script is executed?

Result: Thus, the iimplementation of basic operations such as load, filter, foreach, group, order, limit using Pig Latin Scripts has been done successfully.

Ex No: 05 Date:	Analyze web logs or sales data using Pig

Aim:

To analyze large-scale **web log** or **sales transaction data** using **Apache Pig** for data summarization, filtering, grouping, and aggregation.

Objectives

- To load structured or semi-structured data into Pig.
- To perform data transformations such as filtering, grouping, and joining.
- To compute useful analytics such as total sales, page hits, or user behavior.
- To store and visualize processed output.

Software & Tools Required

- **Hadoop** (Single Node or Pseudo Distributed Mode)
- **Apache Pig**
- **Sample Dataset**:
    - *Web Logs*: access_log file (from web server)
    - *Sales Data*: CSV file with fields like Date, Product, Quantity, Price, Region

Theory

Apache Pig is a high-level platform for processing large data sets in Hadoop.
It uses **Pig Latin**, a data flow language, to express data transformations such as:

- **LOAD**: Load data from HDFS
- **FILTER**: Filter records based on conditions
- **GROUP**: Group data by a field
- **FOREACH…GENERATE**: Project and compute expressions
- **JOIN**: Join multiple datasets
- **DUMP / STORE**: Display or save results

Pig scripts are converted into **MapReduce jobs** automatically by the Pig engine.

Algorithm / Procedure

Case 1: Web Log Data Analysis

Dataset Example (access_log.txt):

192.168.1.1 - - [24/Oct/2025:10:05:32 +0530] "GET /index.html HTTP/1.1" 200 1043

192.168.1.2 - - [24/Oct/2025:10:06:10 +0530] "GET /contact.html HTTP/1.1" 404 321

...

Steps:

1. **Start Hadoop and Pig**
2. start-dfs.sh
3. start-yarn.sh
4. pig
5. **Load the data into Pig**
6. logs = LOAD '/user/hadoop/access_log.txt' USING TextLoader() AS (line:chararray);
7. **Extract required fields using regular expressions**
8. parsed = FOREACH logs GENERATE
9. REGEX_EXTRACT(line, '^(\\S+)', 1) AS ip,
10. REGEX_EXTRACT(line, '\\[(.*?)\\]', 1) AS timestamp,
11. REGEX_EXTRACT(line, '"(.*?)"', 1) AS request,
12. REGEX_EXTRACT(line, 'HTTP/1\\.1" (\\d+)', 1) AS status;
13. **Group and Count requests per IP**
14. grp = GROUP parsed BY ip;
15. count_requests = FOREACH grp GENERATE group AS ip, COUNT(parsed) AS total_requests;
16. **Filter failed requests (e.g., status = 404)**
17. failed = FILTER parsed BY status == '404';
18. **Store or display results**
19. STORE count_requests INTO '/user/hadoop/output/web_hits' USING PigStorage(',');

DUMP failed;

Case 2: Sales Data Analysis

Dataset Example (sales.csv):

Date,Product,Quantity,Price,Region

2025-10-01,Mobile,2,15000,South

2025-10-01,Laptop,1,50000,North

2025-10-02,Mobile,3,15000,East

...

Steps:

1. **Load data**
2. sales = LOAD '/user/hadoop/sales.csv' USING PigStorage(',')
3. AS (date:chararray, product:chararray, quantity:int, price:int, region:chararray);
4. **Calculate total sales per product**
5. totals = FOREACH sales GENERATE product, (quantity * price) AS total_amount;
6. **Group by product and sum**
7. grp = GROUP totals BY product;
8. result = FOREACH grp GENERATE group AS product, SUM(totals.total_amount) AS total_sales;
9. **Filter high-value products**
10. high_sales = FILTER result BY total_sales > 50000;
11. **Store results**
12. STORE result INTO '/user/hadoop/output/sales_summary' USING PigStorage(',');

DUMP high_sales;

Sample Output

Product	**Total_Sales**
Mobile	75000
Laptop	50000
Tablet	30000

or for web logs:

IP Address	Total Requests
192.168.1.1	20
192.168.1.2	5

Scenario based Questions:

Scenario 1: Website Traffic Analysis

A company maintains a large web log file (access_log) containing fields such as timestamp, user_id, page_url, referrer_url, response_code, response_time.
You are required to analyze user behavior.

1. Write a Pig script to count how many times each page URL was accessed.
2. Using Pig, compute the total number of unique users who visited the site.

Scenario 2: User Session Analytics
A web portal logs each user's activity with the following fields:
session_id, user_id, action, time_spent, device_type.
Write a Pig script to find the total time spent per user in a day.
Compare the average session time between *mobile* and *desktop* users.
Interpret how this could influence UX design.

Scenario 3: Retail Sales Analysis
You are provided with a sales dataset:
transaction_id, date, store_id, product_id, quantity, price, payment_type.
Write a Pig script to calculate total revenue per store.
Generate a Pig relation to find the total quantity sold for each product.

Pre lab questions

1. What is Apache Pig?

Apache Pig is a high-level data flow platform built on top of Hadoop used to analyze large datasets. It uses a scripting language called Pig Latin, which abstracts complex MapReduce programs into simpler operations like filtering, grouping, and joining.

2. What is Pig Latin?

Pig Latin is the data processing language used in Apache Pig. It is a procedural language that allows users to express data transformations such as loading, filtering, grouping, and joining in a sequence of steps.

3. What types of data can be analyzed using Apache Pig?

Apache Pig can process structured, semi-structured, and unstructured data such as:

Web server logs (e.g., access logs)

Sales transaction data

Sensor data

Social media feeds

4. How is Apache Pig different from traditional MapReduce?

Aspect	MapReduce	Apache Pig
Programming Model	Low-level Java API	High-level scripting (Pig Latin)
Complexity	Complex and verbose	Simplified and readable
Productivity	Low	High
Learning Curve	Steep	Easier for data analysts

5. What are the main modes in which Pig can be run?

Local Mode:

Runs Pig on a single machine using the local file system.

Suitable for testing small datasets.

MapReduce (Hadoop) Mode:

Runs Pig scripts on a Hadoop cluster using HDFS.

Suitable for large-scale data processing.

6. What are some key Pig Latin operators?

LOAD – to read data

FILTER – to remove unwanted data

FOREACH … GENERATE – to transform data

GROUP – to group records

JOIN – to combine datasets

ORDER – to sort data

DUMP / STORE – to display or save results

7. How can you load a web log or sales data file in Pig?

Example:

```
logs = LOAD 'hdfs:/data/weblogs/access.log' USING PigStorage(' ')
    AS (ip:chararray, datetime:chararray, method:chararray, url:chararray, status:int, bytes:int);
```

This command loads a space-separated log file into Pig with the specified schema.

8. What are common preprocessing steps when analyzing web logs?

Removing invalid or incomplete entries

Extracting fields like IP address, timestamp, URL, status code

Filtering specific HTTP status codes (e.g., 200 for success)

Counting visits per user or page

Grouping and aggregating data for traffic analysis

9. Give an example of a Pig command to count page hits per IP.

```
grouped_data = GROUP logs BY ip;
hits_per_ip = FOREACH grouped_data GENERATE group AS ip, COUNT(logs) AS hits;
DUMP hits_per_ip;
```

10. What is the significance of using Pig in analyzing sales data?

Pig allows businesses to:

Process large volumes of transactional data efficiently.

Compute total sales, average purchase value, and customer trends.

Perform joins across multiple data sources (e.g., customer and product data).

Generate quick analytical summaries for decision-making.

Viva Questions

1. **What is the difference between Pig and Hive?**

2. **Explain how Pig converts scripts into MapReduce jobs.**

3. **What are Pig's execution modes?**

4. **How can we handle structured vs unstructured data in Pig?**

5. **Why is Pig suitable for ETL operations?**

Result

The data was successfully analyzed using **Apache Pig**, and aggregated results were stored in HDFS. Pig simplified the large-scale data analysis through high-level scripting.

Ex No: 06 Date:	Create and manipulate data using HiveQL queries

Aim:

To perform data definition and manipulation operations in Hive using HiveQL commands — such as creating databases, tables, loading data, querying data, and performing analytical operations.

Software / Tools Required:

- Hadoop Framework (preferably Hadoop 3.x)
- Apache Hive (2.x or 3.x)
- Ubuntu / Linux environment
- Dataset file (e.g., employee.csv or student.csv)

heory:

Hive is a **data warehouse infrastructure** built on top of Hadoop that provides **data summarization, query, and analysis** using a SQL-like language called **HiveQL**.
Hive translates HiveQL queries into MapReduce jobs for execution in the Hadoop cluster.

Key Components:

- **Database** – Logical grouping of tables.
- **Table** – Similar to an RDBMS table, stores data in rows and columns.
- **Partition** – Organizes data for faster query execution.
- **Managed Table** – Hive controls both metadata and data.
- **External Table** – Hive controls only metadata; data stays in HDFS.

HiveQL Operations:

1. **DDL (Data Definition Language):**
CREATE, DROP, ALTER

2. **DML (Data Manipulation Language):**
LOAD DATA, INSERT, UPDATE, DELETE

3. **DQL (Data Query Language):**
SELECT, WHERE, GROUP BY, ORDER BY

Procedure:

Step 1: Start Hive

$ hive

Step 2: Create a Database

```
CREATE DATABASE company_db;

SHOW DATABASES;

USE company_db;
```

Step 3: Create a Managed Table

```
CREATE TABLE employee (
   emp_id INT,
   emp_name STRING,
   dept STRING,
   salary FLOAT,
   city STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Step 4: Load Data into the Table

Assume your dataset is stored in HDFS or local system as /home/hadoop/employee.csv.

```
LOAD DATA LOCAL INPATH '/home/hadoop/employee.csv'
INTO TABLE employee;
```

Step 5: Display the Table Data

```
SELECT * FROM employee;
```

Step 6: Perform Data Manipulation Queries

(a) Insert a new record

```
INSERT INTO TABLE employee VALUES (106, 'Rahul', 'Finance', 45000, 'Delhi');
```

(b) Update a record

```
UPDATE employee SET salary = 60000 WHERE emp_id = 103;
```

(c) Delete a record

DELETE FROM employee WHERE emp_id = 102;

(Note: UPDATE and DELETE require ACID table configuration in Hive 3.x.)

Step 7: Query Operations

(a) Display employees in “IT” department

SELECT emp_name, salary FROM employee WHERE dept = 'IT';

(b) Count number of employees by department

SELECT dept, COUNT(*) AS total_emp FROM employee GROUP BY dept;

(c) Find highest salary

SELECT MAX(salary) AS highest_salary FROM employee;

(d) Sort employees by salary

SELECT emp_name, salary FROM employee ORDER BY salary DESC;

Sample Dataset: employee.csv

emp_id	emp_name	dept	salary	city
101	John	IT	50000	Chennai
102	Maya	HR	40000	Bangalore
103	Ravi	IT	55000	Hyderabad
104	Sneha	Sales	30000	Chennai
105	Kiran	IT	60000	Mumbai

Expected Output (Examples):

Query: SELECT dept, COUNT(*) FROM employee GROUP BY dept;

dept	count
IT	3
HR	1
Sales	1

Secenario Based Questions:

Scenario 1: E-Commerce Transactions

A company maintains sales data in Hive with fields:
(order_id, customer_id, product_id, quantity, price, order_date, region)

Questions:

1. **Table Creation:**

Write a HiveQL query to create a table sales_data with the above structure stored as **ORC** format, partitioned by region.

2. **Data Insertion:**

Suppose new sales data is available in /user/hive/warehouse/input/sales_2025.csv.
Write a query to load this data into the sales_data table.

Scenario 2: University Database

The university stores data in Hive for course enrollments:
(student_id, name, dept, course_id, semester, marks)

Questions:

1. Create a table enrollment using **TEXTFILE** format and specify a comma as a field delimiter.
2. Load student enrollment data from the HDFS location /data/enrollments.csv.

Scenario 3: Banking Data

A bank keeps customer transaction data as:

(acc_no, name, branch, trans_date, trans_type, amount)

Questions:

1. Create a **managed table** transactions with appropriate data types and store it as a **Parquet** file.
2. Insert 5 sample rows using the INSERT INTO command.

1. What is Hive and why is it used in Hadoop?

Hive is a data warehouse infrastructure built on top of Hadoop that provides data summarization, query, and analysis. It allows users to query and manage large datasets stored in HDFS using a language similar to SQL (HiveQL), which is then converted into MapReduce jobs for execution.

2. What are the main components of Hive architecture?

1. **Hive Client:** Provides an interface (CLI, JDBC, WebUI) to interact with Hive.
2. **Driver:** Manages the lifecycle of a HiveQL statement.
3. **Compiler:** Parses and compiles the query into a DAG of MapReduce or Tez/Spark jobs.
4. **Metastore:** Stores metadata (schemas, tables, columns, partitions).
5. **Execution Engine:** Executes the query plan using Hadoop framework.

3. What is HiveQL?

HiveQL (Hive Query Language) is a SQL-like language used in Hive to perform data definition (DDL), data manipulation (DML), and data query operations. It abstracts the complexity of writing MapReduce programs.

4. What are the data types supported by Hive?

Hive supports:

- **Primitive types:** INT, STRING, FLOAT, DOUBLE, BOOLEAN, BIGINT, TINYINT
- **Complex types:** ARRAY, MAP, STRUCT, UNIONTYPE

5. What are the different modes in which Hive can run?

- **Local mode:** Runs queries locally on a single machine.
- **Map Reduce mode:** Executes Hive queries as MapReduce jobs across a Hadoop cluster.

6. Differentiate between *internal* and *external* tables in Hive.

Aspect	**Internal Table**	**External Table**
Data Storage	Managed by Hive (stored in warehouse directory)	Stored externally (user-defined location)
Data Deletion	Dropping the table deletes data	Dropping the table keeps data intact
Use Case	When Hive exclusively manages data	When data is shared with other tools/processes

7. Write the syntax to create a database and use it.

```
CREATE DATABASE sales_db;
```

USE sales_db;

8. Write the syntax to create a Hive table.

CREATE TABLE employee (

id INT,

name STRING,

salary FLOAT,

department STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

9. How can you load data into a Hive table?

LOAD DATA LOCAL INPATH '/home/hadoop/employee.csv' INTO TABLE employee;

- LOCAL indicates the file is on the local filesystem.
- Without LOCAL, the file must be in HDFS.

10. List some basic data manipulation operations in Hive.

- **INSERT:** Add new records.
- **UPDATE:** Modify existing data (supported from Hive 0.14+ with ACID tables).
- **DELETE:** Remove records.
- **SELECT:** Retrieve data.

VIVA QUESTIONS

What is Hive?

What is HiveQL?

How is Hive different from a traditional RDBMS?

What are the data types supported in Hive?

What is a Hive table?

What is the default storage location for Hive tables?

Result:

Thus, the data definition and manipulation operations in Hive using HiveQL commands — such as creating databases, tables, loading data, querying data, and performing analytical operations has been executed successfully.

Ex No: 07 Date:	Implement Data Exploration with Spark SQL

Aim:

To perform **data exploration** using **Apache Spark SQL** by loading, querying, and analyzing datasets using SQL-like operations.

Objectives

- To understand the integration of **Spark Core** and **Spark SQL**.
- To perform **data ingestion**, **schema inference**, and **data exploration** using SQL queries.
- To use **DataFrame API** and **Spark SQL queries** for analysis.
- To visualize and interpret dataset statistics.

Software Requirements

- Apache Spark (version ≥ 3.0)
- Hadoop (optional for HDFS integration)
- Java (JDK 8 or above)
- Python (if using PySpark)
- Dataset: e.g., sales.csv or students.json
- IDE or environment: Jupyter Notebook / Spark Shell / IntelliJ / VS Code

Theory

Spark SQL is a Spark module for structured data processing. It provides:

- A programming abstraction called **DataFrame**.
- The ability to run **SQL queries**.
- Integration with **Hive**, **Avro**, **Parquet**, **ORC**, **JSON**, and **CSV**.

Key Concepts:

- **SparkSession**: Entry point to Spark SQL functionality.
- **DataFrame**: Distributed collection of data organized into named columns.
- **Temporary View / Table**: Allows SQL queries using spark.sql().

Typical Workflow:

1. Load dataset into a DataFrame.
2. Create a temporary SQL view.
3. Run SQL queries for data exploration.
4. Display and interpret results.

Procedure

Step 1: Start Spark Session

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("Data Exploration with Spark SQL") \
    .getOrCreate()
```

Step 2: Load Dataset

Example: Load a CSV file.

```
df = spark.read.csv("sales.csv", header=True, inferSchema=True)
df.show(5)
```

Step 3: Explore Schema

```
df.printSchema()
df.columns
```

Step 4: Create Temporary View

```
df.createOrReplaceTempView("sales")
```

Step 5: Perform SQL Queries

```
# View all records
spark.sql("SELECT * FROM sales LIMIT 10").show()
# Count total records
spark.sql("SELECT COUNT(*) AS total_records FROM sales").show()
# Summary statistics
spark.sql("SELECT AVG(amount) AS avg_sales, MAX(amount) AS max_sales FROM sales").show()
```

Group by operations

spark.sql("SELECT region, SUM(amount) AS total_sales FROM sales GROUP BY region").show()

Step 6: Apply Filters and Ordering

spark.sql("SELECT * FROM sales WHERE amount > 1000 ORDER BY amount DESC").show()

Step 7: Perform Aggregations

```
spark.sql("""
SELECT category,
    COUNT(*) AS num_sales,
    ROUND(AVG(amount),2) AS avg_amount
FROM sales
GROUP BY category
ORDER BY avg_amount DESC
""").show()
```

Step 8: Data Summary using DataFrame API

df.describe().show()

Step 9: Stop Spark Session

spark.stop()

Sample Dataset (sales.csv)

sale_id	customer	region	category	amount	date
1	John	East	Grocery	250.75	2024-05-12
2	Alice	West	Electronics	1250.00	2024-05-13
3	David	North	Grocery	500.00	2024-05-14

Output (Expected)

- Schema of dataset displayed.
- SQL query results showing:
    - Record count

- Average and maximum values
- Grouped statistics by region or category
- Filtered and sorted data

Scenario based Questions:

Scenario 1 : Imagine you are a **Data Engineer** working for an e-commerce company, **ShopSmart**, that stores large amounts of customer and transaction data in a distributed system. You are tasked with **analyzing and exploring this data using Apache Spark SQL**.
The datasets include:

- **Customers** (customer_id, name, age, city, gender)
- **Orders** (order_id, customer_id, order_date, amount, status)
- **Products** (product_id, category, price, rating)

Your goal is to perform **data exploration** — understanding data distributions, relationships, and patterns — using **Spark SQL queries**.

Scenario 2:

The marketing team wants to know which city has the highest number of customers.

Write a Spark SQL query to find the top 3 cities with the most customers.

Scenario 3:

The finance team asks for the average order amount per customer. Join the Customers and Orders tables using Spark SQL.

Prelab Questions:

1. What is Spark SQL?

Spark SQL is a component of Apache Spark that provides a programming interface for working with structured and semi-structured data. It allows users to query data using SQL syntax and integrates SQL queries with Spark programs.

2. What are DataFrames in Spark SQL?

A DataFrame is a distributed collection of data organized into named columns, similar to a relational database table. It is built on top of RDDs and optimized using Spark's Catalyst optimizer for better performance.

3. How is Spark SQL different from traditional SQL?

Feature	Spark SQL	Traditional SQL
Execution	Runs on a distributed cluster	Runs on a single machine
Scalability	Can handle large datasets	Limited scalability
Integration	Works with Python, Java, Scala APIs	Limited to SQL syntax
Optimization	Uses Catalyst optimizer	Uses DBMS-specific optimizers

4. What are the data sources supported by Spark SQL?

Spark SQL supports multiple data sources such as:

JSON files

Parquet files

CSV files

Avro files

ORC files

Hive tables

JDBC (relational databases)

5. What is the role of the SparkSession object in Spark SQL?

SparkSession is the entry point to use Spark SQL. It allows creation of DataFrames, execution of SQL queries, and interaction with various data sources.
Example:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("DataExploration").getOrCreate()
```

6. What is the Catalyst Optimizer in Spark SQL?

The Catalyst Optimizer is Spark SQL's query optimizer. It analyzes and optimizes SQL queries for better performance using rule-based and cost-based optimization techniques.

7. How can you run SQL queries on a DataFrame?

You can register a DataFrame as a temporary view and then execute SQL queries using the SparkSession's sql() method.
Example:

```
df.createOrReplaceTempView("sales")
spark.sql("SELECT category, SUM(amount) FROM sales GROUP BY category").show()
```

8. What are some common DataFrame operations used for data exploration?

show() – Displays records

select() – Select specific columns

filter() or where() – Filter records based on condition

groupBy() – Group data for aggregation

describe() – Summary statistics of columns

orderBy() – Sort the data

9. What are advantages of using Spark SQL for data exploration?

Supports large-scale data processing

Allows integration of SQL and programming APIs

High performance due to in-memory computation

Supports schema inference and data optimization

10. What is the difference between a temporary view and a global temporary view in Spark SQL?

Type	Scope	Example
Temporary View	Limited to the current SparkSession	df.createOrReplaceTempView("myview")
Global Temporary View	Shared across all SparkSessions	df.createGlobalTempView("myview")

VIVA QUESTIONS:

1. What is Spark SQL?

2. What are the advantages of using Spark SQL for data exploration?

3. What is a DataFrame in Spark SQL?

4. How do you create a DataFrame in Spark SQL?

5. What are SparkSession and its role in Spark SQL?

Result

Data exploration using **Spark SQL** was successfully implemented. Various SQL queries were executed to analyze the dataset and extract insights.

Ex No: 08 Date:	Implement Movie recommendation system using MLlib

Aim

To implement a **Movie Recommendation System** using **Apache Spark MLlib** based on user ratings and collaborative filtering techniques.

Objectives

- To understand **collaborative filtering** and its application in recommendation systems.
- To use **Spark MLlib's ALS (Alternating Least Squares)** algorithm to build a recommendation model.
- To train, evaluate, and generate movie recommendations using a real-world dataset.

Software and Hardware Requirements

Category	Requirements
Software	Apache Spark (≥ 3.0), Hadoop (optional), Python (≥ 3.8) or Scala, Jupyter Notebook / PySpark shell
Libraries	pyspark, numpy, pandas
Dataset	MovieLens Dataset (e.g., ratings.csv, movies.csv)
Hardware	System with at least 8 GB RAM, multi-core CPU

Theory

Recommendation Systems

Recommendation systems predict user preferences for items and suggest the most relevant ones.
There are three types:

1. **Content-based filtering**
2. **Collaborative filtering**
3. **Hybrid systems**

Collaborative Filtering

Collaborative filtering recommends items by learning user-item interactions.
It assumes that users who agreed in the past will agree again in the future.

ALS (Alternating Least Squares) Algorithm

MLlib implements **ALS**, a matrix factorization technique that:

- Learns latent factors for users and items.

- Predicts missing ratings.
- Works well with large-scale sparse datasets.

The formula:

$$R_{m\times n} \approx P_{m\times k} \times Q^{T}_{n\times k}$$

where

- R: User-item rating matrix
- P: User-feature matrix
- Q: Item-feature matrix
- k: Number of latent features

Procedure

Step 1: Load Libraries

from pyspark.sql import SparkSession

from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.ml.recommendation import ALS

Step 2: Initialize Spark Session

spark = SparkSession.builder.appName("MovieRecommendation").getOrCreate()

Step 3: Load Dataset

ratings = spark.read.csv("ratings.csv", header=True, inferSchema=True)

movies = spark.read.csv("movies.csv", header=True, inferSchema=True)

ratings.show(5)

movies.show(5)

Step 4: Data Preprocessing

- Select relevant columns: userId, movieId, rating
- Handle missing values or duplicates (if any)

ratings = ratings.select("userId", "movieId", "rating").dropna()

Step 5: Split Dataset

(train, test) = ratings.randomSplit([0.8, 0.2])

Step 6: Build ALS Model

```
als = ALS(
    maxIter=10,
    regParam=0.1,
    userCol="userId",
    itemCol="movieId",
    ratingCol="rating",
    coldStartStrategy="drop"
)
model = als.fit(train)
```

Step 7: Make Predictions

```
predictions = model.transform(test)
predictions.show(5)
```

Step 8: Evaluate Model

```
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print(f"Root-mean-square error = {rmse}")
```

Step 9: Generate Recommendations

```
# Recommend top 5 movies for each user
user_recs = model.recommendForAllUsers(5)
user_recs.show(5, truncate=False)
# Recommend top 5 users for each movie
movie_recs = model.recommendForAllItems(5)
movie_recs.show(5, truncate=False)
```

Step 10: Stop Spark Session

```
spark.stop()
```

Sample Output

```
+-------+-------+------+-------------+
|userId |movieId|rating|prediction   |
+-------+-------+------+-------------+
|1      |31     |4.0   |3.85         |
|1      |1029   |3.0   |3.12         |
...
Root-mean-square error = 0.89
```

Scenario Based Questions:
Scenario 1: A movie recommendation system uses collaborative filtering with the Alternating Least Squares (ALS) algorithm from Spark MLlib.
The dataset (e.g., ratings.csv) contains user ratings for movies in the format:
userId, movieId, rating, timestamp
The system trains a model to predict unknown ratings and recommend movies to users.

You have a dataset of 100,000 movie ratings by 10,000 users.
Question: How will you split this dataset for training and testing to evaluate your MLlib ALS model?

Scenario 2: A movie recommendation system uses collaborative filtering with the Alternating Least Squares (ALS) algorithm from Spark MLlib.
The dataset (e.g., ratings.csv) contains user ratings for movies in the format:
userId, movieId, rating, timestamp
The system trains a model to predict unknown ratings and recommend movies to users.

The Spark cluster is running out of memory during ALS training.

Question: Which parameters in the ALS algorithm can you tune to reduce memory usage and why?

Scenario 3: A movie recommendation system uses collaborative filtering with the Alternating Least Squares (ALS) algorithm from Spark MLlib.
The dataset (e.g., ratings.csv) contains user ratings for movies in the format:
userId, movieId, rating, timestamp
The system trains a model to predict unknown ratings and recommend movies to users.

A new user signs up but has not rated any movie yet.
Question: How would your recommendation system handle this "cold start" problem?

Prelab Questions

1. What is a recommendation system?

A recommendation system is an application that predicts the preference or rating a user would give to an item (such as a movie, product, or song). It helps suggest items that are most relevant to the user based on their behavior or preferences.

2. What are the main types of recommendation systems?

Content-Based Filtering: Recommends items similar to what the user liked in the past, based on item features.

Collaborative Filtering: Recommends items based on the behavior and preferences of similar users.

Hybrid Systems: Combine both content-based and collaborative filtering methods for better accuracy.

3. What is MLlib in Apache Spark?

MLlib is Spark's scalable machine learning library. It provides algorithms for classification, regression, clustering, and recommendation (including collaborative filtering using ALS).

4. What is ALS (Alternating Least Squares) in MLlib?

ALS is a matrix factorization algorithm used in collaborative filtering. It predicts missing ratings by decomposing the user-item interaction matrix into two low-rank matrices — user features and item features — and alternately optimizes them to minimize prediction error.

5. What type of filtering does ALS implement?

ALS implements Collaborative Filtering, particularly Matrix Factorization based on user-item interactions.

6. What are the main inputs to the ALS algorithm?

User ID

Item ID (Movie ID)

Rating (e.g., 1 to 5)

These inputs represent the interactions between users and movies.

7. What are the important parameters of the ALS algorithm in MLlib?

- rank: Number of latent factors (features).
- maxIter: Number of iterations to run.
- regParam: Regularization parameter to prevent overfitting.
- coldStartStrategy: Strategy to handle new users/items (e.g., "drop").

8. How is the performance of a recommendation model evaluated?

By comparing predicted ratings with actual ratings using metrics like:

- RMSE (Root Mean Square Error)
- MAE (Mean Absolute Error)

Lower RMSE or MAE indicates better model performance.

9. What is the typical workflow of building a movie recommendation system in MLlib?

1. Load movie and rating datasets.
2. Parse and prepare training data (userId, movieId, rating).
3. Split data into training and test sets.
4. Train ALS model on the training set.
5. Evaluate using RMSE on the test set.
6. Generate recommendations for users.

10. What datasets are commonly used for movie recommendation experiments?

MovieLens Dataset (by GroupLens): A widely used open dataset containing user ratings of movies. Example: ratings.csv (userId, movieId, rating, timestamp) and movies.csv (movieId, title, genres).

VIVA QUESTIONS:

1. What is MLlib in Apache Spark?

2. What is a recommendation system?

3. What type of recommendation system is used in this experiment?

4. What is Collaborative Filtering?

5. What are the two types of Collaborative Filtering?

Result

A **Movie Recommendation System** was successfully implemented using **Spark MLlib's ALS algorithm**. The system predicts movie ratings and recommends top movies to users with a reasonable RMSE performance.

Ex No: 09 Date:	Create a real-time leaderboard using Redis

Aim

To design and implement a **real-time leaderboard** using **Redis** data structures (Sorted Sets) to rank players dynamically based on their scores.

Objectives

- To understand Redis data structures, particularly **Sorted Sets (ZSET)**.
- To implement real-time updates and retrieval of leaderboard data.
- To demonstrate the **use of Redis commands** for insertion, ranking, and retrieval.
- To simulate a real-world leaderboard for gaming or analytics applications.

Software / Hardware Requirements

- **Software:**
  - Redis Server (version ≥ 6.0)
  - Redis CLI or Python (with redis library)
  - Python 3.x / Node.js (optional for application layer)
- **Hardware:**
  - Any system with ≥ 4GB RAM and internet connectivity

Theory

Redis is an **in-memory key-value store** known for **speed and simplicity**.
The **Sorted Set** (ZSET) data structure stores **unique members** with an associated **score**, automatically ordered by that score.

Commands used:

- ZADD key score member → Add a member with a score
- ZINCRBY key increment member → Increment a member's score
- ZRANGE key start stop [WITHSCORES] → Get members by rank (ascending)
- ZREVRANGE key start stop [WITHSCORES] → Get top-ranked members (descending)
- ZRANK key member / ZREVRANK key member → Get rank of a member
- ZSCORE key member → Get score of a member

Leaderboard Concept:
A **Leaderboard** ranks users (e.g., players, employees, students) in real-time based on their performance score. Redis ensures constant-time updates and sorted ranking retrieval.

5. Procedure

Step 1: Start Redis Server

redis-server

Step 2: Open a new terminal and connect via CLI

redis-cli

Step 3: Create a new sorted set for leaderboard

ZADD game_leaderboard 1500 "Alice"

ZADD game_leaderboard 1200 "Bob"

ZADD game_leaderboard 1800 "Charlie"

ZADD game_leaderboard 1000 "David"

Step 4: Display the leaderboard (highest to lowest)

ZREVRANGE game_leaderboard 0 -1 WITHSCORES

Step 5: Update a player's score in real-time

ZINCRBY game_leaderboard 300 "Bob"

Step 6: Check updated ranking

ZREVRANGE game_leaderboard 0 -1 WITHSCORES

Step 7: Retrieve rank and score of a specific player

ZRANK game_leaderboard "Alice"

ZREVRANK game_leaderboard "Alice"

ZSCORE game_leaderboard "Alice"

Step 8: Remove a player (if needed)

ZREM game_leaderboard "David"

Python Implementation

import redis

```
# Connect to Redis
r = redis.Redis(host='localhost', port=6379, db=0)
# Add initial scores
r.zadd('leaderboard', {'Alice': 1500, 'Bob': 1200, 'Charlie': 1800, 'David': 1000})
# Display leaderboard
print("Initial Leaderboard:")
for user, score in r.zrevrange('leaderboard', 0, -1, withscores=True):
    print(user.decode(), int(score))
# Update score dynamically
r.zincrby('leaderboard', 300, 'Bob')
# Display updated leaderboard
print("\nUpdated Leaderboard:")
for user, score in r.zrevrange('leaderboard', 0, -1, withscores=True):
    print(user.decode(), int(score))
```

Expected Output

Initial Leaderboard:

Charlie 1800

Alice 1500

Bob 1200

David 1000

After Bob's update (+300):

Charlie 1800

Bob 1500

Alice 1500

David 1000

SCENARIO BASED QUESTIONS

SCENARIO 1: Storing and Updating Player Scores

You need to maintain a real-time leaderboard where each player's score changes after every game.
Which Redis data structure would you use and why?
How would you add or update a player’s score?

SCENARIO 2: Displaying Top 10 Players

Your frontend needs to display the **top 10 players** in real time.
Which Redis command will you use, and how do you ensure it lists players in descending order of scores?

SCENARIO 3: Getting a Player's Rank and Score

Question:
The system must show a player's **current rank** and **score** when they log in.
Which commands will retrieve this information?

PRELAB QUESTIONS:

1. What is Redis and why is it suitable for building a real-time leaderboard?

Redis (Remote Dictionary Server) is an open-source, in-memory data structure store used as a database, cache, and message broker. It is suitable for real-time leaderboards because it offers:

- Extremely low latency and high throughput.
- Support for sorted data structures (Sorted Sets).
- Atomic operations, ensuring leaderboard scores are updated consistently in real time.

2. Which Redis data structure is most commonly used for implementing leaderboards, and why?

The Sorted Set (ZSET) data structure is used because it stores unique elements with an associated score, automatically sorted by score. This allows for:

- Efficient ranking (ZRANK, ZREVRANK).
- Fetching top N players (ZREVRANGE).
- Updating scores (ZINCRBY).

3. What is the command to add or update a player's score in a Redis sorted set?

ZADD leaderboard 1500 "player1"

or to increment a score:

ZINCRBY leaderboard 100 "player1"

These commands either add or update a player’s score atomically.

4. How can you retrieve the top 5 players from a Redis leaderboard?

Using the ZREVRANGE command (since it sorts in descending order of score):

ZREVRANGE leaderboard 0 4 WITHSCORES

This retrieves the top 5 players along with their scores.

5. How can you find the rank of a specific player in the leaderboard?

Use the ZREVRANK command:

ZREVRANK leaderboard "player1"

This returns the player’s rank (0-based index) in descending order.

6. What is the difference between ZRANGE and ZREVRANGE commands?

- ZRANGE returns elements in ascending order of score.

- ZREVRANGE returns elements in descending order of score (useful for leaderboards).

7. Why is Redis preferred over traditional relational databases for real-time leaderboard applications?

Redis is preferred because:

- It stores data in memory for fast access.
- It supports atomic score updates and rank queries.
- It scales easily for large datasets.
- No complex joins or disk I/O delays as in SQL databases.

8. What type of persistence mechanisms does Redis offer to ensure data durability?

Redis provides two persistence options:

- RDB (Redis Database Backup): Periodic snapshots of data.
- AOF (Append Only File): Logs every write operation.
  These can be combined for better durability and faster recovery.

9. How can Redis Pub/Sub be used in a real-time leaderboard?

Redis Pub/Sub can be used to broadcast real-time score updates to connected clients whenever the leaderboard changes, enabling live updates without polling.

10. What are some use cases of Redis-based real-time leaderboards?

- Online gaming leaderboards (tracking top players).
- E-commerce sites (top-selling products).
- Social media apps (most liked or shared posts).
- Live contests or quizzes (fastest or highest scorers).

VIVA QUESTIONS

1. What is the difference between a Redis Set and a Sorted Set?

2. How does Redis ensure ordering in Sorted Sets?

3. Which Redis command is used to increase a player's score?

4. Can Redis handle concurrent leaderboard updates?

5. Explain how Redis achieves high performance in real-time applications.

Result

A **real-time leader board** was successfully created and updated using Redis **Sorted Sets**. The experiment demonstrated real-time ranking updates and efficient data retrieval.

Ex No: 10 Date:	Create a document store with MongoDB and run CRUD operations

Aim

To create a document-oriented database using MongoDB and perform **CRUD (Create, Read, Update, Delete)** operations.

Objectives

- To understand the architecture and data model of MongoDB.
- To create databases and collections in MongoDB.
- To insert, retrieve, update, and delete documents using MongoDB commands.
- To perform queries using **MongoDB Query Language (MQL)**.

Software / Hardware Requirements

Requirement	Specification
Software	MongoDB Community Edition (v6.0 or later)
IDE / Tool	Mongo Shell / MongoDB Compass / VS Code MongoDB Extension
Hardware	Intel Core i3/i5 processor, 4 GB RAM (minimum)
Operating System	Windows / Linux / macOS

Theory

MongoDB is a **NoSQL, document-oriented database** that stores data in **JSON-like BSON (Binary JSON)** format.
Each record is called a **document**, which is a collection of key-value pairs.
Documents are grouped into **collections**, similar to tables in relational databases.

CRUD operations are the fundamental operations to interact with data:

- **C – Create:** Insert new documents.
- **R – Read:** Retrieve documents using queries.
- **U – Update:** Modify existing documents.
- **D – Delete:** Remove documents from the collection

Command Summary

Operation	MongoDB Command
Create Database	**use db_name**
Create Collection	**db.createCollection("collection_name")**
Insert Document	**db.collection_name.insertOne({...}) or insertMany([...])**
Read Documents	**db.collection_name.find()**
Update Document	**db.collection_name.updateOne({...}, {$set:{...}})**
Delete Document	**db.collection_name.deleteOne({...})**
Drop Collection	**db.collection_name.drop()**
Drop Database	**db.dropDatabase()**

Procedure

Step 1: Start MongoDB

- Launch MongoDB service:
- mongod
- Open Mongo shell in a new terminal:
- mongosh

Step 2: Create a Database

use studentDB

This switches to (or creates) a database named studentDB.

Step 3: Create a Collection

db.createCollection("students")

Step 4: Insert Documents (CREATE)

```
db.students.insertMany([
  { "roll_no": 101, "name": "Alice", "dept": "CSE", "cgpa": 8.7 },
  { "roll_no": 102, "name": "Bob", "dept": "ECE", "cgpa": 7.9 },
  { "roll_no": 103, "name": "Charlie", "dept": "CSE", "cgpa": 9.1 }
```

])

Step 5: Retrieve Documents (READ)

```
// Display all documents
db.students.find()
// Pretty print format
db.students.find().pretty()
// Conditional query
db.students.find({ dept: "CSE" })
// Projection (show selected fields)
db.students.find({}, { name: 1, cgpa: 1, _id: 0 })
```

Step 6: Update Document (UPDATE)

```
db.students.updateOne(
  { roll_no: 102 },
  { $set: { cgpa: 8.3 } }
)
```

Step 7: Delete Document (DELETE)

```
db.students.deleteOne({ roll_no: 103 })
```

Step 8: Drop Collection or Database

```
db.students.drop()
db.dropDatabase()
```

Sample Output

```
> use studentDB
switched to db studentDB
> db.createCollection("students")
{ ok: 1 }
> db.students.insertOne({"roll_no":101,"name":"Alice","dept":"CSE","cgpa":8.7})
{ acknowledged: true, insertedId: ObjectId("6742d...") }
```

```
> db.students.find().pretty()
{
  "_id": ObjectId("6742d..."),
  "roll_no": 101,
  "name": "Alice",
  "dept": "CSE",
  "cgpa": 8.7
}
```

SCENARIO BASED QUESTIONS:

SCENARIO 1: Online Bookstore Database

You are designing a document-oriented database for an online bookstore using MongoDB.

A sample document in the books collection looks like this:

```
{
  "_id": "B101",
  "title": "Learning MongoDB",
  "author": "John Smith",
  "price": 499,
  "genre": "Technology",
  "stock": 25,
  "ratings": [4, 5, 5, 3]
}
```

Create Operation

Q1: Write a MongoDB command to insert three new book documents into the books collection.

Q2: How would you ensure that each book document has a unique _id field automatically generated by MongoDB?

SCENARIO 2: Online Bookstore Database

You are designing a document-oriented database for an online bookstore using MongoDB.
A sample document in the books collection looks like this:

```
{
  "_id": "B101",
  "title": "Learning MongoDB",
  "author": "John Smith",
  "price": 499,
  "genre": "Technology",
  "stock": 25,
  "ratings": [4, 5, 5, 3]
}
```

Read Operation

Q3: Write a query to display all books where the price is greater than 400 and genre is "Technology".

Q4: How can you find the average rating for each book and display the book titles with their average rating?

SCENARIO 3: Online Bookstore Database

You are designing a document-oriented database for an online bookstore using MongoDB.
A sample document in the books collection looks like this:

```
{
  "_id": "B101",
  "title": "Learning MongoDB",
  "author": "John Smith",
  "price": 499,
  "genre": "Technology",
  "stock": 25,
  "ratings": [4, 5, 5, 3]
}
```

Update Operation

- **Q5: The publisher decides to increase all Technology book prices by 10%. Write an update query to reflect this.**

- **Q6: A book's stock has dropped to zero. Update the document to include a new field availability: "Out of Stock".**

PRELAB QUESTIONS:

1. What is MongoDB?

MongoDB is a NoSQL, document-oriented database that stores data in a flexible, JSON-like format called BSON (Binary JSON). It allows dynamic schemas, making it suitable for applications requiring scalability and fast data access.

2. What is a document store?

A document store is a type of NoSQL database that stores, retrieves, and manages data as documents—usually in JSON, BSON, or XML formats. Each document is a self-contained record that can include nested fields and arrays.

3. How is MongoDB different from traditional RDBMS?

Feature	RDBMS	MongoDB
Data model	Tables with rows and columns	Collections with JSON-like documents
Schema	Fixed	Dynamic
Joins	Supported	Not supported (uses embedding or referencing)
Transactions	Supported (ACID)	Supported (multi-document in latest versions)
Scalability	Vertical	Horizontal (sharding)

4. What is a collection in MongoDB?

A collection in MongoDB is analogous to a table in RDBMS. It is a group of MongoDB documents that share similar characteristics but can have different structures (schema-less).

5. What is BSON in MongoDB?

BSON (Binary JSON) is a binary representation of JSON documents used by MongoDB for efficient storage and traversal. It supports additional data types like Date, ObjectId, and Binary.

6. Define CRUD operations.

CRUD stands for:

- Create → Insert new documents
- Read → Query documents
- Update → Modify existing documents
- Delete → Remove documents

These are the four basic operations used in database management.

7. What command is used to insert a document into a MongoDB collection?

db.collection_name.insertOne({ key1: "value1", key2: "value2" }); or

db.collection_name.insertMany([{...}, {...}]);

8. How do you retrieve all documents from a collection?

db.collection_name.find();

9. What is the difference between updateOne() and updateMany() in MongoDB?

- updateOne() modifies only the first matching document.
- updateMany() modifies all documents that match the filter.

Example:

db.collection.updateOne({name:"Alice"}, {$set:{age:25}});

db.collection.updateMany({status:"active"}, {$set:{verified:true}});

10. What command deletes documents from a collection?

db.collection_name.deleteOne({ filter });

db.collection_name.deleteMany({ filter });

VIVA QUESTIONS:

1. **What is BSON and how does it differ from JSON?**

2. **What are the advantages of MongoDB over relational databases?**

3. **How are collections in MongoDB similar to or different from tables in SQL?**

4. **What is the purpose of _id in MongoDB documents?**

5. **How do you perform an update operation on multiple documents?**

Result

A document store database named studentDB was successfully created using MongoDB, and CRUD operations were performed successfully.