||Jai Sri Gurudev ||

BGSKH Education Trust (R.) – A unit of Sri Adichunchanagiri Shikshana Trust(R.)

BGS College of Engineering and Technology (BGSCET)

(Approved by AICTE, New Delhi and Affiliated to VTU, Belagavi)

Adjacent to Mahalakshmi Metro Station, Mahalakshmipuram, West of Chord Road, Bengaluru -560 086, Karnataka

BIG DATA ANALYTICS
Lab Manual

For VII Semester BE [VTU/CBCS, 2024-25 Syllabus]
Subject Code:

B	C	S	7	1	4	D

Name: …………………………………………………………..….

Branch: ………………………………………………………..……

USN: ………………………….…………………………….….…

PREFACE

A big data analytics lab aims to equip students with the skills to handle and analyze large, complex datasets. Key objectives include understanding fundamental Big Data concepts, mastering MapReduce for parallel processing, learning NoSQL database techniques, applying Hadoop ecosystems, and utilizing modern reporting and machine learning tools. Outcomes include the ability to perform data gathering, analyze datasets, apply statistical measures, and generate insights from large datasets using various tools and techniques.

This lab emphasizes store big data for data processing and analysis of the results using MongoDB. The lab also covers various Big data tasks on Hadoop such as adding files, directories, retrieving files, deleting files and execution of Map-Reduce Programs.

Key areas of learning include:

- Implement MongoDB based application to store big data for data processing and analyzing the results
- Install Hadoop and Implement the following file management such as Adding files and directories, Retrieving files, Deleting files and directories and execute Map-Reduce based programs.

Upon completion of the course, the student will be able to understand big Data and its analytics in the real world, analysis of Big Data framework like Hadoop and MongoDB to efficiently store and process the big data to generate analytics.

Department of Computer science & Engineering

ABOUT

The department of Computer Science and Engineering was established in the year 2020.The department offer undergraduate programmed in Computer Science & Engineering. The department has a very good infrastructure and faculty to provide excellent education to meet the industry standards.

Today, the department caters to the needs of more than 60 UG Students. It houses state of the art computing facilities with high end servers which support the LAN, provide a Linux/Unix environment, also provides exclusive library facility to its students and boasts of well trained and experienced faculty teaching the departments various courses in the areas of Computer Networks, Computer Architecture, Database Systems, Microprocessor, Operating Systems, Analysis and Design of Algorithms and Software Engineering.

The department lays stress on the practical and application based aspects through laboratories, seminars, group discussions, viva-vice and project work, keeping pace with the growth in Computer Science & Engineering technology.

The Students are given scope to conduct experiments on innovative ideas. A sound theoretical and practical work prepares the students in wider field of Computer Science & Engineering to take up challenging jobs in the area of Big Data,System Software etc.

BGS College Of Engineering

(Affiliated to Visvesvaraya Technological University, Belagavi)
Bengaluru-86

Department Of Computer Science & Engineering

BGS College of Engineering & Technology

Vision

To transform students into innovative professionals by cracking real worldproblems using artificial intelligence and machine learning approaches.

Mission

M1: Enriching students with core knowledge which will avail themselves toresolve real world problems.

M2: Empowering students by collaborating with industries, motivating them totake up internships and projects.

M3: Equipping students with the skills making them industry ready, self-employable, & researchers.

Program Educational Objectives (PEOs)

After 3 or 4 years of the graduation the CSE graduates shall be able,

PEO 1:	To achieve sustainable growth as Computer Science Engineers in reputed organizations.
PEO 2:	To accomplish higher education and research with highest degree of professionalism andintegrity.
PEO 3:	To extend the services to mankind by exhibiting leadership qualities and ethical values.

Program Specific Outcomes (PSOs)

PSO1	Apply the strong knowledge and principles of Computer Science and Engineering to model and design various computing systems.
PSO2	Develop the diverse applications in well promised domains by adopting the practices of Computer Science & Engineering.

Dept. of CSE, BGSCET

PROGRAM OUTCOMES

PO's	DESCRIPTION
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals,and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering-problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in-diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to-manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE DETAILS

Course Name: Big Data analytics Lab

Course Code: BCS714D

Course Credit: 1 :

COURSE OUTCOME

CO 1. Develop programs using HADOOP framework
CO 2. Use Hadoop Cluster to deploy Map Reduce jobs
CO 3: Develop program using MongoDB
CO 3. Analyze the given data set to identify deep insights

	PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO1 0**	**PO1 1**	**PO1 2**
CO1												
CO2												
CO3												

COURSE OUTCOME	**PSO1**	**PSO 2**
CO1	**3**	**2**
CO2	**3**	**3**
CO3	**2**	**1**
Big Data Lab (BCS714D)	**PSO1**	**PSO2**
	3	**3**

LAB EVALUATION PROCESS

WEEK WISE EVALUATION OF EACH PROGRAM		
SL.NO	**ACTIVITY**	**MARKS**
1	Record	5
2	Program Execution	5
	TOTAL	**10**

INTERNAL ASSESSMENT EVALUATION (End of Semester)		
SL.NO	**ACTIVITY**	**MARKS**
1	Write-Up	20
2	Execution	40
3	Result Verification	20
4	Viva Voce	20
	TOTAL	**100**

FINAL INTERNAL ASSESSMENT CALCULATION		
SL.NO	**ACTIVITY**	**MARKS**
1	Average of Weekly Entries	30
2	Average of internal Assessment	20
	TOTAL	**50**

LIST OF PROGRAMS	
Sl.NO	**Experiments**
1	Installation of Hadoop
2	Implement the following file management such as Adding files and directories, Retrieving files, Deleting files and directories in Hadoop
3	Execute Map- Reduce based programs for Word count in Hadoop
4	Implement MongoDB based application to store big data for data processing and analyzing the results 4.1 Simple Queries to access MongoDB database 4.2 CRUD operations on student database
5	Implement MongoDB based application to store big data for data processing and analyzing the results 4.3 Demonstrate dealing with NULL values 4.4 Demonstrate Count, Limit, Sort, and Skip operations 4.5 Demonstrate array operations like create, find , update
6	4.6 demonstrate aggregate function for a customer database : Consider the collection “Customers" as given below. It has four documents. We would like to filter out those documents where the "AccType" has a value other than "S". After the filter, we should be left with three documents where the "Acctype": "S". It is then required to group the docu- ments on the basis of CustID and sum up the "AccBal" for each unique "CustID". This is similar to the output received with group by clause in RDBMS. Once the groups have been formed [as per the example below, there will be only two groups: (a) "CustID" : "C123" and (b) “CustID" : "C111”], filter and display that group where the "TotAccBal" column has a value greater than 1200.

Introduction to Big Data Analytics Lab

What is the Big Data?

Big data refers to extremely large and complex datasets that are difficult to process using traditional data processing tools. It's characterized by high volume, velocity, and variety of data, and often requires specialized tools and techniques for analysis. The "big" in big data refers not only to the size of the data but also to the complexity and speed at which it is generated and collected.

Key Characteristics (the 5 Vs):

Volume:
Refers to the massive amount of data generated and stored, often measured in petabytes or exabytes.

Velocity:
Describes the speed at which data is generated and processed. This includes data streams from sources like social media, sensors, and financial transactions.

Variety:
Indicates the different formats and types of data, including structured (e.g., databases), unstructured (e.g., text, images, audio), and semi-structured (e.g., JSON, XML).

Veracity:
Concerns the accuracy and reliability of the data. Big data can be noisy or contain errors, requiring careful cleaning and validation.

Value:
Represents the potential insights and business value that can be derived from analyzing big data

What is Hadoop?

Hadoop is an open-source framework for storing and processing massive datasets on clusters of commodity hardware, enabling scalable, parallel, and distributed computing for big data. It breaks down large data workloads into smaller tasks that are processed simultaneously across many inexpensive computers, a method inspired by Google's MapReduce model . Key components include the Hadoop Distributed File System (HDFS) for storage and YARN for resource management and job scheduling.

Key components of Hadoop:

- **Hadoop Distributed File System (HDFS):**
  Provides reliable, distributed storage for large files across the cluster.
- **YARN (Yet Another Resource Negotiator):**
  A framework for managing cluster resources and scheduling jobs, supporting various workloads.
- **MapReduce:**
  The programming model that transforms and aggregates data in parallel to process large datasets.
- **Hadoop Common :**
  A set of shared libraries and utilities that provide necessary services for other Hadoop modules, including cluster management and security features.

MongoDB is a popular document database and a type of NoSQL database that uses a flexible, schema-less format to store data in documents resembling JSON. It is known for its ability to handle unstructured data , scalability , and high availability , making it a common choice for modern web and mobile applications. Unlike traditional relational databases that use tables and rows, MongoDB stores data in JSON-like documents.

Key Characteristics

- **Document-Oriented:**

  Data is stored in flexible, JSON-like documents that can contain various data types, including numbers, strings, booleans, arrays, and nested documents.
- **NoSQL:**

  It is a non-relational database, meaning it does not rely on a fixed, tabular structure like traditional SQL databases .
- **Schema Flexibility:**

  MongoDB features a dynamic schema, allowing developers to store different types of data within the same collection without needing to define the exact structure beforehand.
- **Scalability:**

  It supports horizontal scaling through sharding , which distributes large datasets across multiple servers to handle high loads.
- **High Availability:**

MongoDB offers features like replication to ensure data protection and application availability.

Experiment-1

Install Hadoop

HDFS is a scalable distributed file system designed to scale to petabytes of data while running on top of the underlying file system of the operating system. HDFS keeps track of where the data resides in a network by associating the name of its rack (or network switch) with the dataset. This allows Hadoop to efficiently schedule tasks to those nodes that contain data, or which are nearest to it, optimizing bandwidth utilization. Hadoop provides a set of command line utilities that work similarly to the Linux file commands, and serve as your primary interface with HDFS.

Hadoop installation
1. Setup Files:

- The setup files required for this installation are provided in a Google Drive link.

Download Setup Files of Hadoop https://shorturl.at/iMEDM

Download Eclipse: https://www.eclipse.org/downloads/

Official Hadoop Documentation: https://hadoop.apache.org/docs/current/

2. DLL File (msvcr120.dll):

- This file is a commonly required runtime component of Microsoft Visual C++ 2013 Redistributable.
- If your system lacks this file, you can download and install the official Microsoft Visual C++ Redistributable package from Microsoft's website.
- Manual placement of DLL files in the System32 folder is not recommended unless you are fully aware of the implications. Instead, use the official installer to avoid potential system instability.

3. Steps Covered :

- Installing Java JDK
- Setting up Hadoop
- Configuring environment variables
- Testing the setup

Pre-steps: Open command prompt in "Run as administrator" mode

- **Check Java version installed: java –version**
- **Check Hadoop version installed: hadoop version**
- **Initiate HDFS daemons: start-dfs.cmd**

Please note:

1. It is important to make sure the previous JDK is completely uninstalled. Install the JDK
2. provided in the video 1.

2. Make sure even visual redistributable is also uninstalled completely. Please use the VC.exe provided in the video 1.
3. Once installation is done
Follow these commands in the command prompt to check if installation is successful
i. Open command prompt in "Run as Adminstrator" mode
ii. Check java version: java –version
iii. Check hadoop version: hadoop version
iv. Start the data node and name node: start-dfs.cmd
v. Start resource manager : start-yarn.cmd
vi. Check if all the nodes are succcesfully installed:
a. Type jps

b. You should see all these services started
Namenode
Datanode
Resourcemanager
Nodemanager
4. Don't close the services window directly. Instead type: stop-all.cmd

Note: After initiating HDFS daemon, such as after giving "start-dfs.cmd", data node or name node may shutdown .
This is because the cluster ids of data node and name node are in different cluster.

Go to your local directory, of data node and name node and make the cluster ids same
Eg: Go to C:/Hadoop/dfs/datanode(this path can be different in your local system)
Eg: Go to C:/Hadoop/dfs/namenode
Under datanode and namenode folder, check the cluster ids. Copy the cluster id of datanode to namenode or vice-versa.

This command -dfs.cmd brings up the core components necessary for HDFS to function, allowing you to store and manage data within your Hadoop environment. Before running this command, it is often necessary to format the NameNode using **hdfs namenode -format,** though this should only be done once during the initial setup as it will erase existing HDFS data.

- **Initiate YARN daemons: start-yarn.cmd**

Before you can run Hadoop programs on data stored in HDFS, you‘ll need to put the data into HDFS first. Let‘s create a directory and put a file in it. HDFS has a default working directory of /user/$USER, where $USER is your login user name. This directory isn‘t automatically created for you, though, so let‘s create it with the mkdir command.

Experiment - 2

2. Implement the following file management such as Adding files and directories, Retrieving files, Deleting files and directories

1. To Create a directory in HDFS at given path(s).
Usage:
hadoop fs -mkdir <paths>
Example:

- hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2
- hadoop fs –mkdir /user1

Now open http://localhost:9870/ in your browser. Under Utilities->Browse the file system -> click Go
You should be able to see your directory "user1" created

2. To get list of directories and files at the root of HDFS
Hadoop fs –ls /

3. To get list of complete directories and files of HDFS
Hadoop fs –ls R /

4. To get List the contents of a directory.
hadoop fs -ls <args>
Example:

- hadoop fs -ls /user/saurzcode
- hadoop fs –ls /user1

3. Upload a file/files in HDFS.
Upload:
hadoop fs -put:
Copy single src file, or multiple src files from local file system
to the Hadoop data file system
Usage:
hadoop fs -put <localsrc> ... <HDFS_dest_Path>
Example:

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

- hadoop fs –put E:\BGS\BigDataLab\Hadoop /user1

open http://localhost:9870/ in your browser.
In browse directory, you should be able to see the files under the directory /user1

4. Download file/files:
hadoop fs -get:
Copies/Downloads files to the local file system
Usage:
hadoop fs -get <hdfs_src> <localdst>
Example:

- hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/
- hadoop fs –get /user1/Hadoop E:\BGS

Now, open your local machine and check whether the files have been downloaded.

To copy a file from HDFS to local file system

- hadoop fs –get /user1/Hadoop/model_QP1.pdf E:\BGS

To copy a file from local file system to HDFS via **copyFromLocal** command

- hadoop fs –copyFromLocal E:\BGS\ model_QP1 /user2/Hadoop

4. See contents of a file
Same as unix cat command:
Usage:
hadoop fs -cat <path[filename]>
Example:

- hadoop fs -cat /user/saurzcode/dir1/abc.txt
- hadoop fs –cat /user1/Hadoop/Module5_BDS306B_QA.pdf

The ASCII contents of the file is displayed in command prompt

5. Remove a file or directory in HDFS.
Remove files specified as argument. Deletes directory only when it is empty
Usage :
hadoop fs -rm <arg>
Example:

- hadoop fs -rm /user/saurzcode/dir1/abc.txt
- hadoop fs –rm /user1/Hadoop/Module5_BDS306B_QA.pdf

check the browser. The file is deleted from “Browse directory”

To display contents of an HDFS file on console

```
hadoop fs –cat /user1/Hadoop/module3_bds306b_Quick_revision.pdf
```

To copy a file from one directory to another on HDFS

```
hadoop fs –cp /user1/Hadoop/ Lecture1_Introduction.mp4 /user2
```

Recursive version of delete.

Usage :

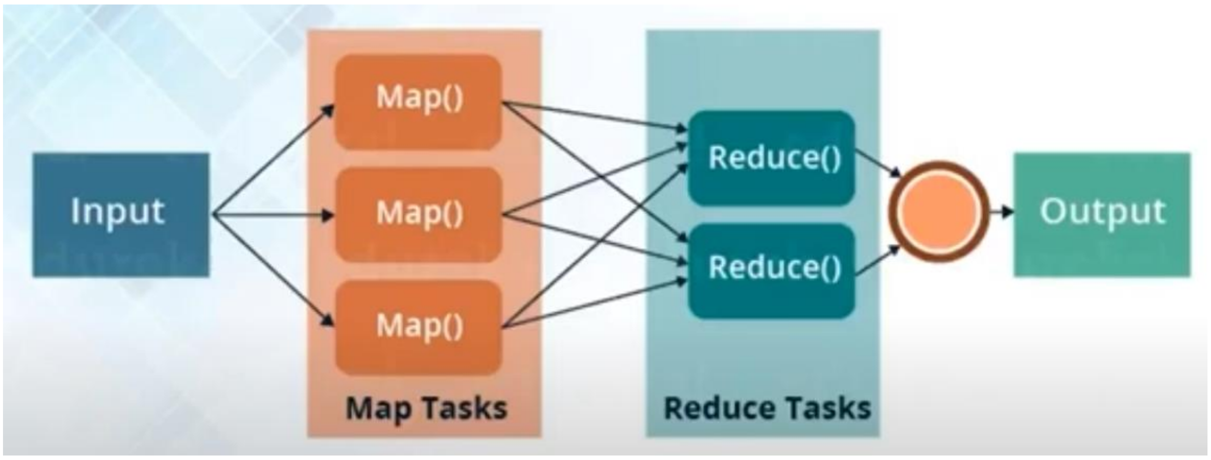
```
hadoop fs -rmr <arg>
```

Example:

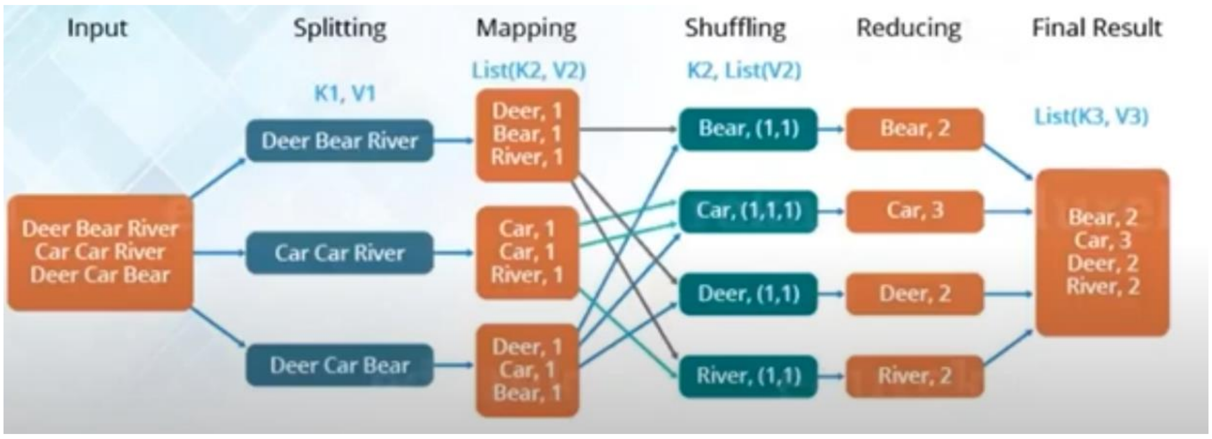
- hadoop fs -rmr /user/saurzcode/
- hadoop fs –rmr /user1/Hadoop/**Module5_BDS306B_QA.docx**

Experiment 3:
Execute Map- Reduce based programs for Word count

<u>Map- Reduce based programs</u>
<u>Word count Map Reduce Programming using Java</u>



Input file is divided into input splits which is passed to Map() function, which is passed on to reduce() functions and aggregation task is performed at Reducer. Finally the complete output is sent back to client.

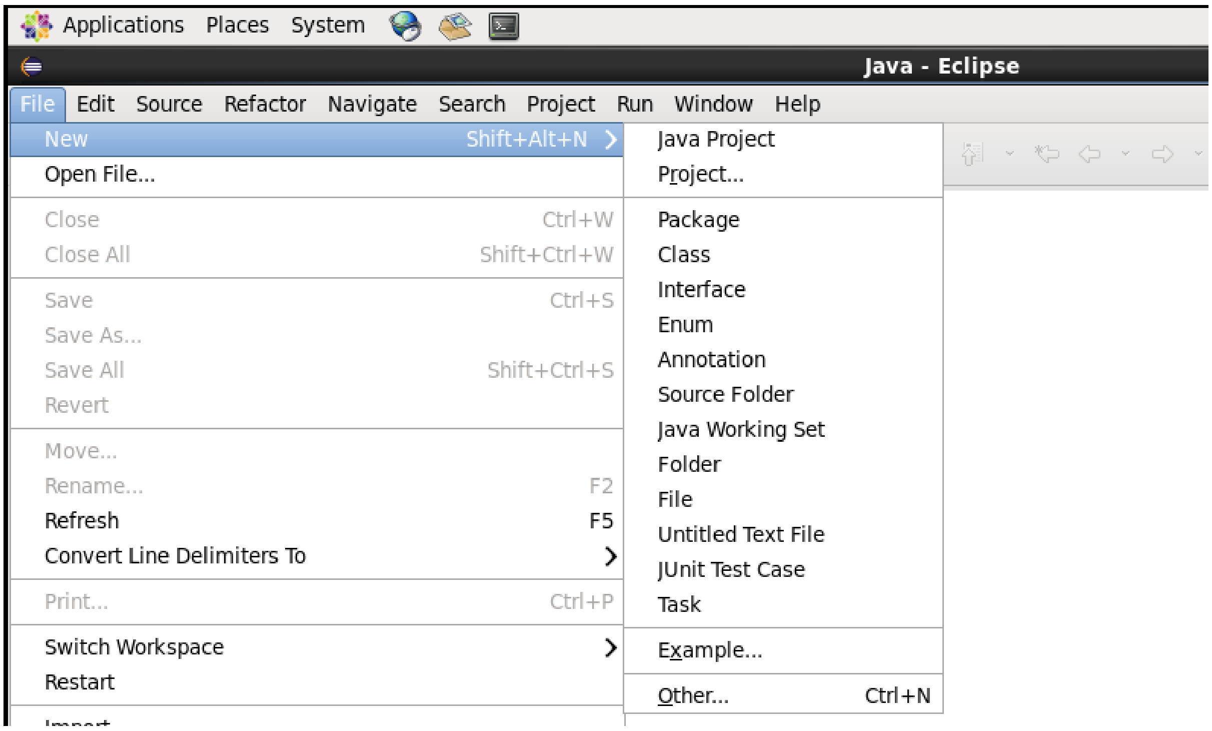


https://www.youtube.com/watch?v=gEm4XY04WAU

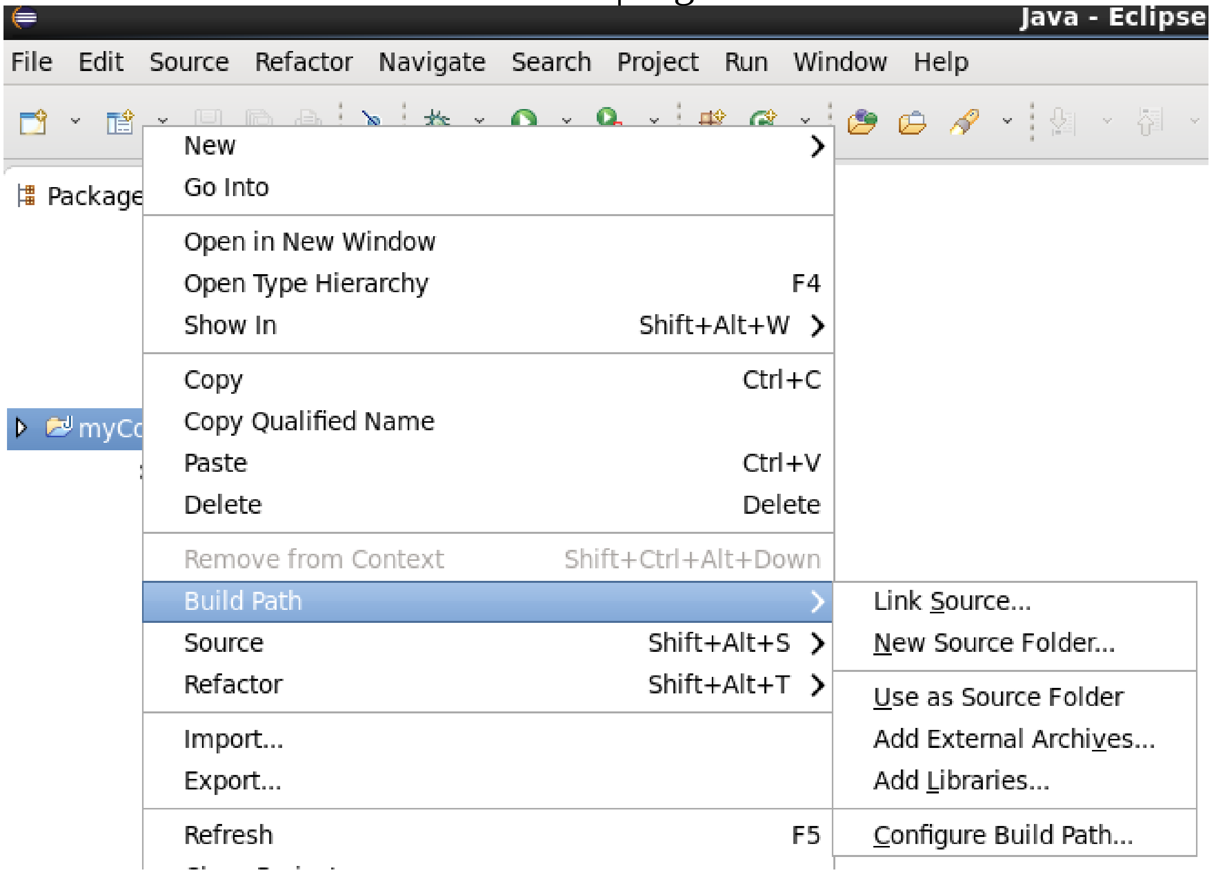
<u>https://www.youtube.com/watch?v=gEm4XY04WAU</u>
Steps:

- First Open **Eclipse** -> then select **File** -> **New** -> **Java Project** ->

Name it **WordCount** -> then **Finish**.



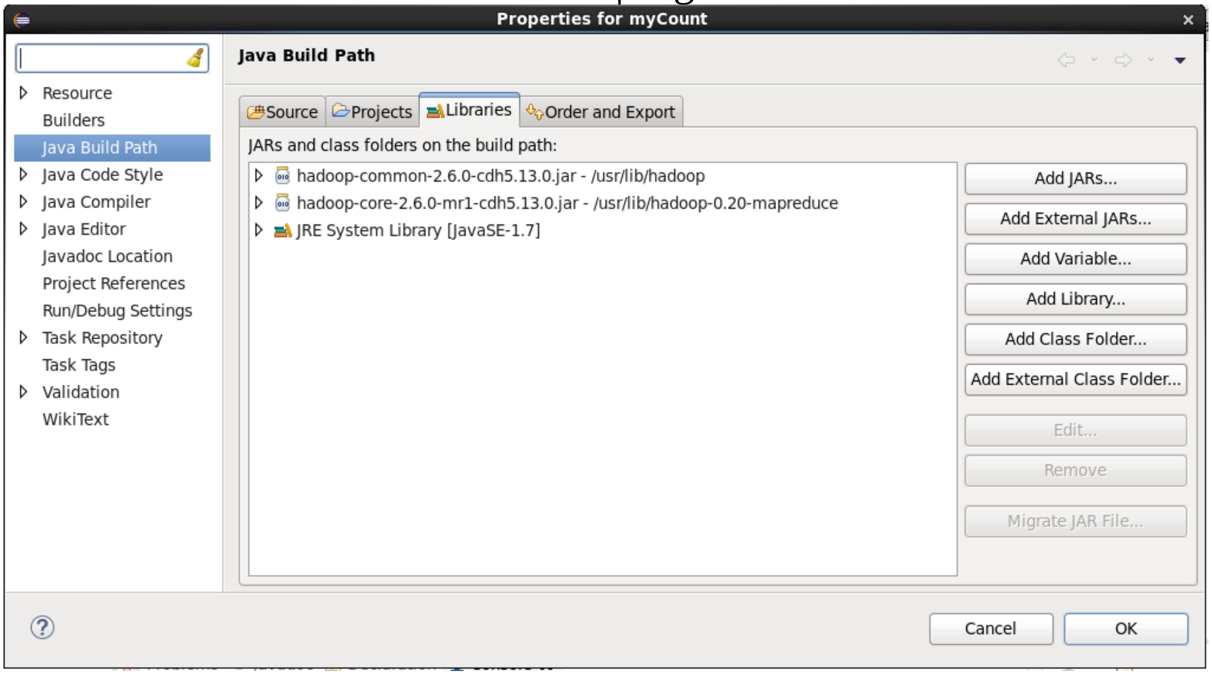
- Create Three Java Classes into the project. Name them **WCDriver**(having the main function), **WCMapper**, **WCReducer**.
- You have to include two Reference Libraries for that:
  Right Click on **Project** -> then select **Build Path**-> Click on **Configure Build Path**



In the below figure, you can see the Add External JARs option on the Right Hand Side. Click on it and add the below mention files. You can find these files in */usr/lib/*
1. /usr/lib/hadoop-0.20-mapreduce/hadoop-core-2.6.0-mr1-cdh5.13.0.jar
2. /usr/lib/hadoop/hadoop-common-2.6.0-cdh5.13.0.jar

In windows, search where the jar files are and add it

- Eg: C:\hadoop\share\hadoop\mapreduce



JARs and class folders on the build path:

- hadoop-common-2.6.0-cdh5.13.0.jar - /usr/lib/hadoop
- hadoop-core-2.6.0-mr1-cdh5.13.0.jar - /usr/lib/hadoop-0.20-mapreduce
- JRE System Library [JavaSE-1.7]

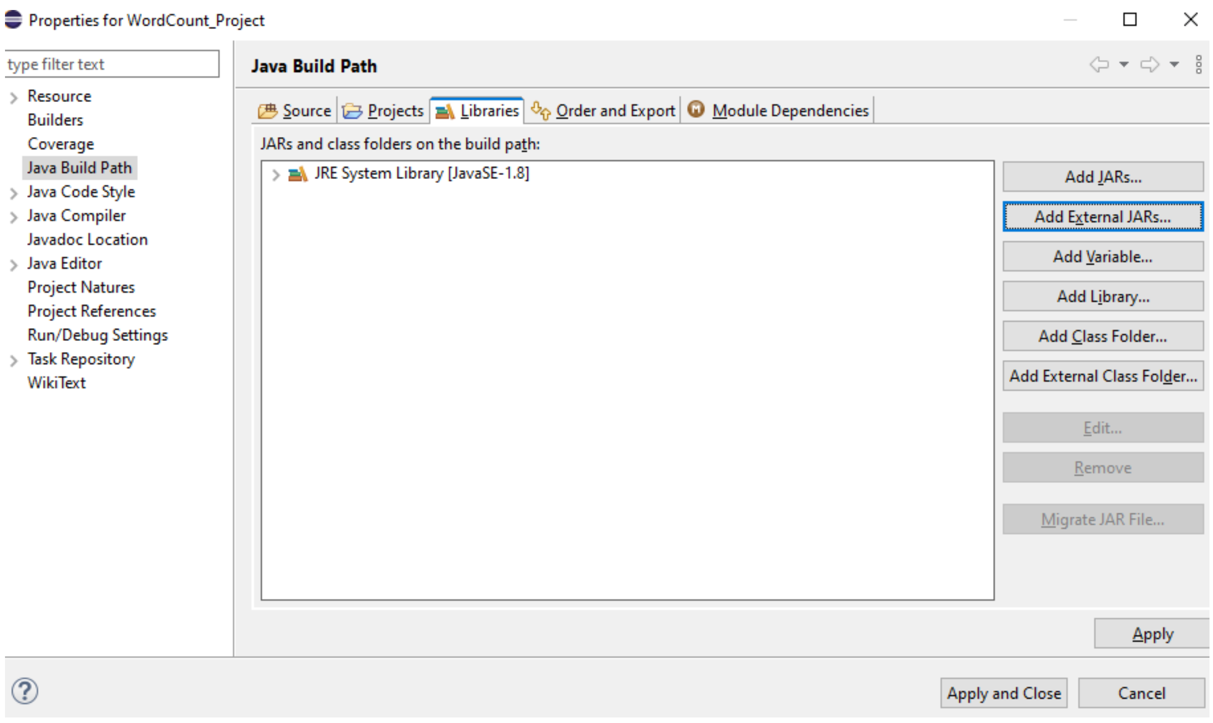
Mapper Code: You have to copy paste this program into the WCMapper Java Class file.

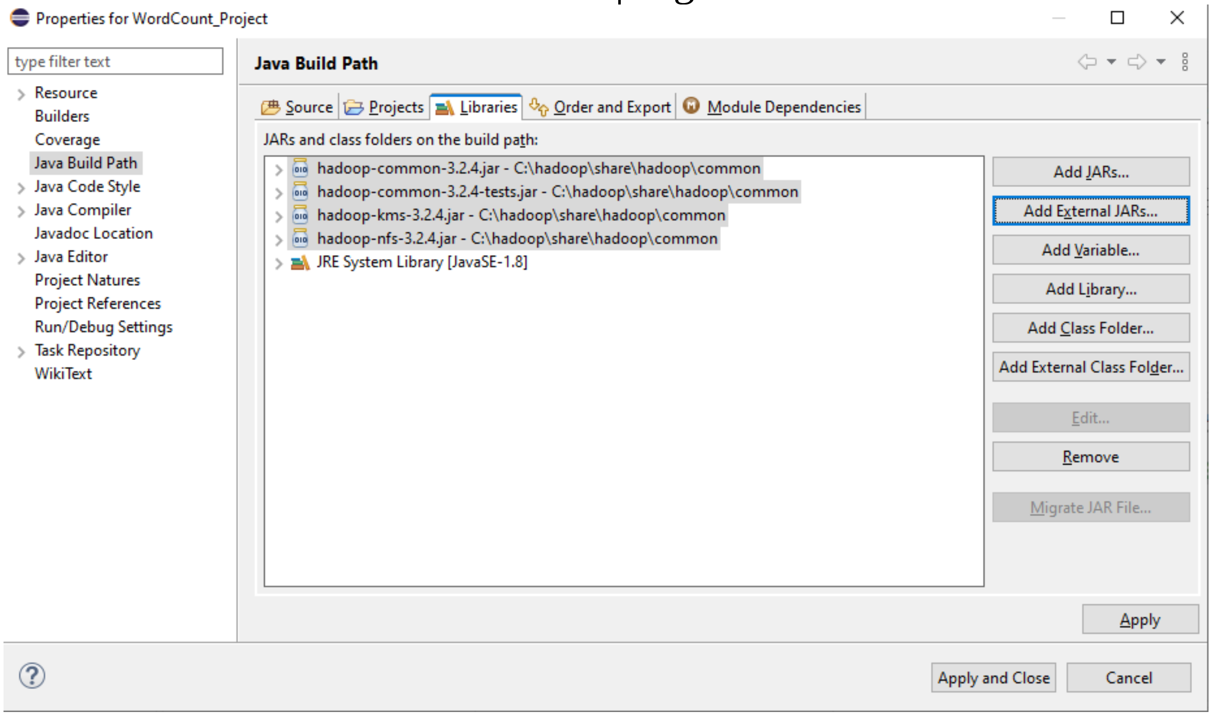
```
// Importing libraries
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable,
                                        Text, Text, IntWritable> {

    // Map function
    public void map(LongWritable key, Text value, OutputCollector<Text,
                IntWritable> output, Reporter rep) throws IOException
    {

        String line = value.toString();

        // Splitting the line on spaces
        for (String word : line.split(" "))
        {
            if (word.length() > 0)
            {
                output.collect(new Text(word), new IntWritable(1));
            }
        }
    }
}
```

Driver Code: You have to copy paste this program into the WCDriver Java Class file.

```
// Importing libraries
import java.io.IOException;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WCDriver extends Configured implements Tool {

    public int run(String args[]) throws IOException
    {
        if (args.length < 2)
        {
            System.out.println("Please give valid inputs");
            return -1;
        }

        JobConf conf = new JobConf(WCDriver.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        conf.setMapperClass(WCMapper.class);
        conf.setReducerClass(WCReducer.class);
        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        JobClient.runJob(conf);
        return 0;
    }

    // Main Method
    public static void main(String args[]) throws Exception
    {
        int exitCode = ToolRunner.run(new WCDriver(), args);
        System.out.println(exitCode);
    }
}
```

- Go to your project->right click->build path->configure build path->click Libraries->click Add external JARs-> browse to hadoop default location(C:\hadoop\share\hadoop\mapreduce)->select all jars->



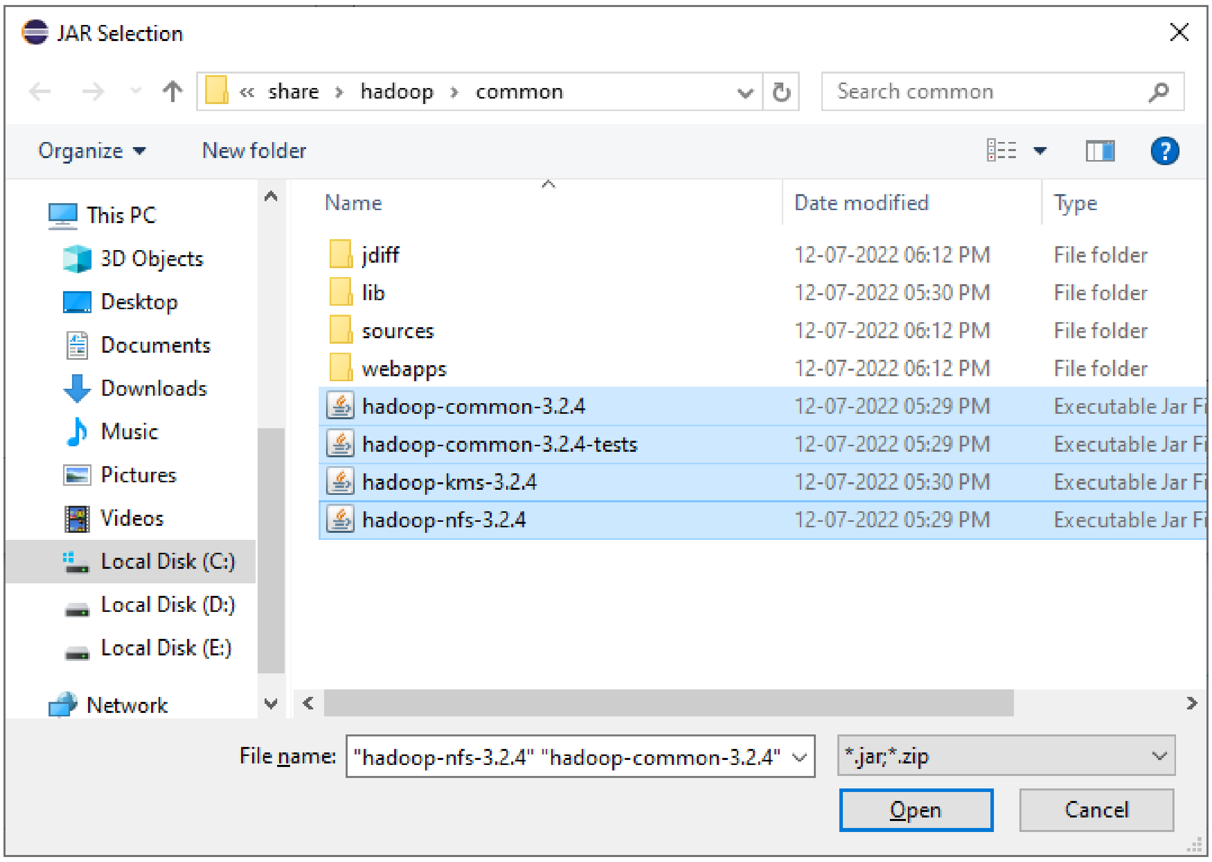


JARs and class folders on the build path:

- hadoop-common-3.2.4.jar - C:\hadoop\share\hadoop\common
- hadoop-common-3.2.4-tests.jar - C:\hadoop\share\hadoop\common
- hadoop-kms-3.2.4.jar - C:\hadoop\share\hadoop\common
- hadoop-nfs-3.2.4.jar - C:\hadoop\share\hadoop\common
- JRE System Library [JavaSE-1.8]

Repeat same:Click "Add external JARs

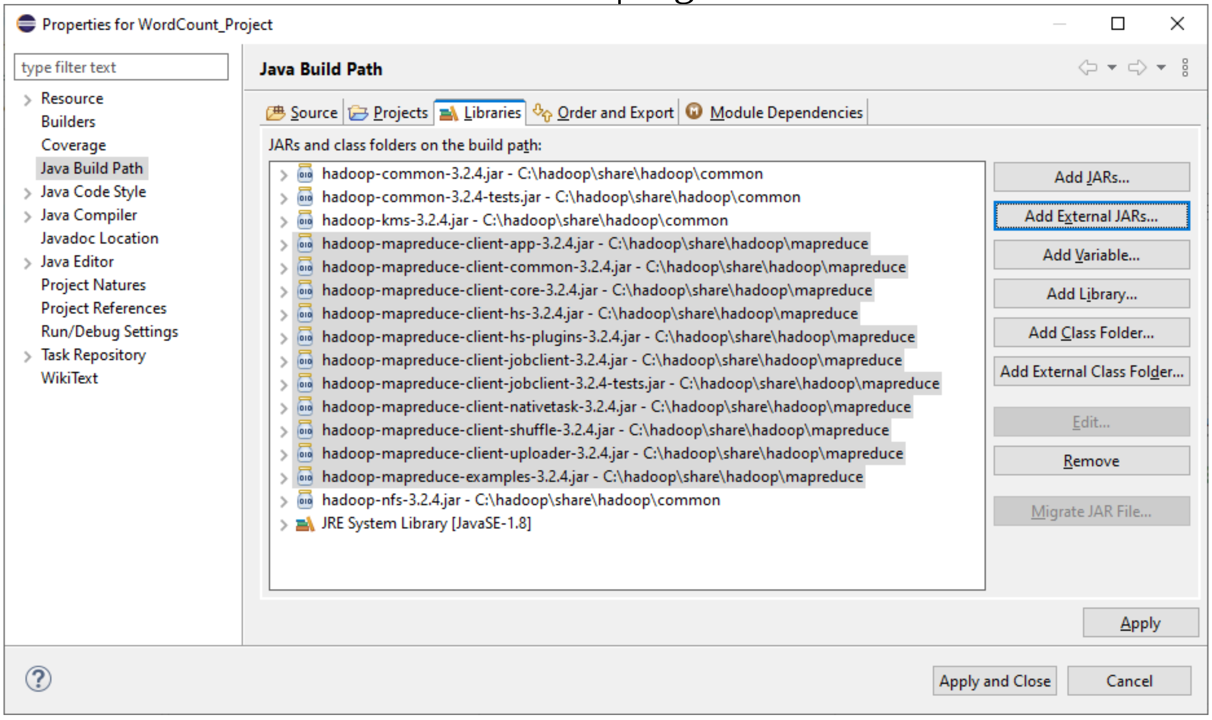
- Go to your project->right click->build path->configure build path->click Libraries->click Add external JARs-> browse to hadoop default location(C:\hadoop\share\hadoop\common)->select all jars->apply and close



Name	Date modified	Type
jdiff	12-07-2022 06:12 PM	File folder
lib	12-07-2022 05:30 PM	File folder
sources	12-07-2022 06:12 PM	File folder
webapps	12-07-2022 06:12 PM	File folder
hadoop-common-3.2.4	12-07-2022 05:29 PM	Executable Jar Fi
hadoop-common-3.2.4-tests	12-07-2022 05:29 PM	Executable Jar Fi
hadoop-kms-3.2.4	12-07-2022 05:30 PM	Executable Jar Fi
hadoop-nfs-3.2.4	12-07-2022 05:29 PM	Executable Jar Fi

File name: "hadoop-nfs-3.2.4" "hadoop-common-3.2.4"

.jar;.zip



JARs and class folders on the build path:

```
hadoop-common-3.2.4.jar - C:\hadoop\share\hadoop\common
hadoop-common-3.2.4-tests.jar - C:\hadoop\share\hadoop\common
hadoop-kms-3.2.4.jar - C:\hadoop\share\hadoop\common
hadoop-mapreduce-client-app-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-common-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-core-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-hs-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-hs-plugins-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-jobclient-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-jobclient-3.2.4-tests.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-nativetask-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-shuffle-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-client-uploader-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-mapreduce-examples-3.2.4.jar - C:\hadoop\share\hadoop\mapreduce
hadoop-nfs-3.2.4.jar - C:\hadoop\share\hadoop\common
JRE System Library [JavaSE-1.8]
```

Now all the errors should disappear from the code because all external jars for hadoop has been added to eclipse.

Syntax for hadoop
Hadoop jar jar-file-location input directory output directory

Click on java file() -> right click ->select export ->select JAR file under Java -> click Next->select project check box->select 2 checkboxes for classpath and project->give name of JAR-> eg:abcd.jar->next->next-> givemain class name-> browse-.select your main class name(eg: **WCDriver)->finish**

Note: where the jar file is stored. You need it later
Eg: E:\BGS\BigDataLab\abcjar.jar

Now cross check whether jar file (eg:abcd.jar) file is available.

Prepare you text file for word count and store in your local disk.

Now , send the file to HDFS from local disk

Go to terminal and start hadoop configuration:
Cmd->right click as administrator

Open hadoop configuration

Hadoop dfs –put <path of text file to count> <root directory>

Eg: hadoop fs–put E:\BGS\BigDataLab\text_sample.txt \user12

Now Browse to http://localhost:9870/ - > utilities->browse the file system-> directory text_sample.txt \should be created under user12 directory

Now execute word count program
Hadoop jar <jar location> <hdfsfilepath> <outdirectory that would be created>
Eg: hadoop jar E:\BGS\BigDataLab\wordcountjar.jar \user12 \output12

1 .create a directory
hadoop fs -mkdir /user12

Browse to http://localhost:9870/ - > utilities->browse the file system-> directory user12 should be created

2. Now in command prompt

hadoop jar E:\BGS\BigDataLab\abcdjar.jar dir11 dir11/outdir

dir11 is my source file

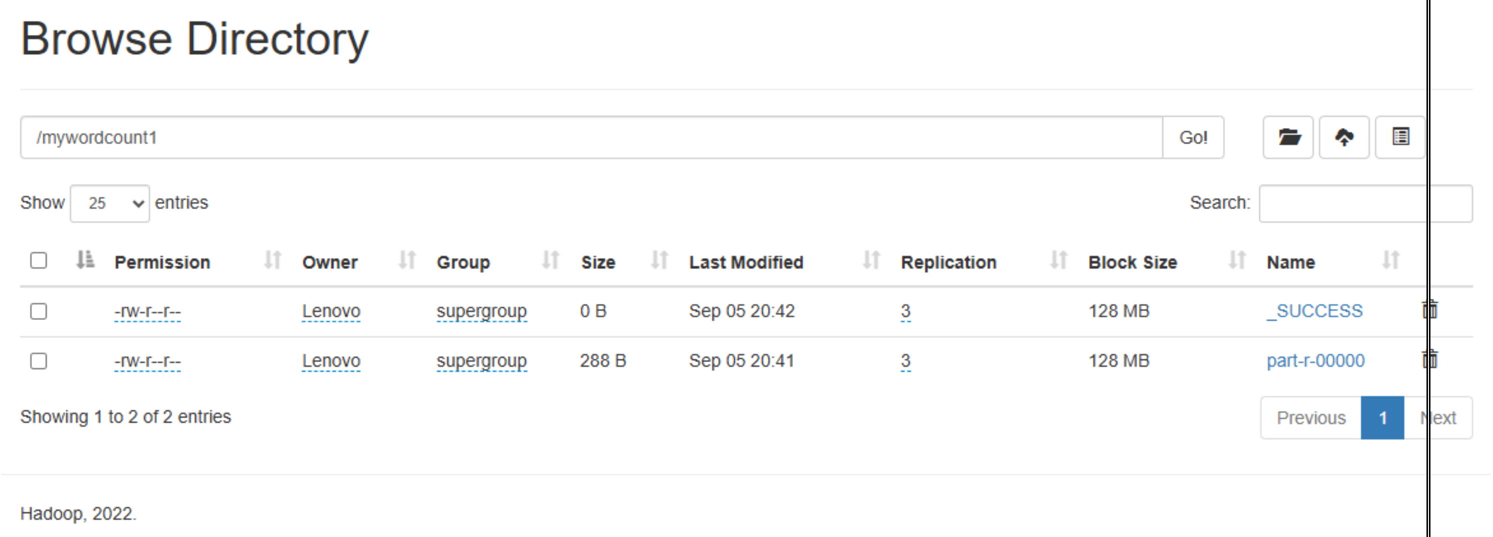
under dir11/outdir the output file called “part-r-0000” is created which gives count of the words

3. now shut down all the hadoop configuration successfully

stop-dfs.sh ; stops the NameNode and DataNodes

stop-yarn.sh; stops the ResourceManager and NodeManagers.

OUTPUT: PART FILE CREARED UNDER /mywordcount shows the word count



Browse Directory

/mywordcount1 Go!

Show 25 entries Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	Lenovo	supergroup	0 B	Sep 05 20:42	3	128 MB	_SUCCESS
☐	-rw-r--r--	Lenovo	supergroup	288 B	Sep 05 20:41	3	128 MB	part-r-00000

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2022.

Expertiment - 4

Implement MongoDB based application to store big data for data processing and analyzing the results

1 . To display all databases : >show dbs

2 . To create database: >use Database_name

Eg:>use db1

>use db2

3.To Drop database: >db.dropDatabase();

4. To report name of current database
>db

5. To switch to a new database: >use db2

6. To display list of all collections(tables) in current database
>show collections

7. to display current version of MongoDB server
>db.version()

8. To display statistics that reflect use state of a database
>db.stats()

8. To get list of commands
>db.help()

Case 1: Consider a table 'Students' with following columns:
StudRollNo
StudName
Grade
Hobbies
DOJ

1. **Insert details in to Students table:**

```
db.Students.insert({_id:1,
StudRollNo:'$101',
StudName:'Simon',
```

Grade:’VII’,
Hobbies:’Surfing’,
DOJ:’10-Oct-2012’});

>db.Students.insert({_id:1,StudRollNo:'$101',StudName:'Simon',Grade:'VII',Hobbies:'Surfing',DOJ:'10-Oct-2012'});

Update students:
db.Students.update({StudRollNo:’$101’},{$set:{Hobbies:’Ice Hockey’}})

db.Students.update({},{$set:{Hobbies:’Ice Hockey’}},
{multi:true})

2. **Delete from students where RollNo=’$101’**

Db.Students.remove({RollNo=’$101’})

Delete from Students
Db.Students.remove({})

3. **Select * from students**

Db.Students.find()
Db.Students.find().pretty()

Select * from students where StudRollNo=’$101’
Db.Students.find({StudRollNo=’$101’})

Select StudRollNo, StudName,Hobbies from Students
Db.Students.find({},{StudRollNo:1,StudName:1,Hobbies:1,_id:0})

Select StudRollNo, StudName,Hobbies from Students where StudRolNo=’$101’
Db.Students.find({},{StudRollNo:’$101’,StudName:1,Hobbies:1,_id:0})

4. **Conditinal insert Aryan David only if not already present in collection.If present, Update his hobbies.**

   Check current documents:

db.Students.find().pretty()

Insert using upsert:

db.Students.update({_id:3}, {StudName:"Aryan David", Grade:"VII", $set:{Hobbies:"Skating"}}, {upsert:true}) Confirm insertion: db.Students.find().pretty()

5. Update existing documents: using upsert

Update Aryan David's hobbies from "Skating" to "Chess".

```
db.Students.update({_id:3},      {StudName:"Aryan     David",
                                 Grade:"VII",
$set:{Hobbies:"Chess"}}, {upsert:true})
```

6. Insert document using save()

If the document exists, it replaces the existing one.

Insert Vamsi Bapat without specifying _id.

db.Students.save({StudName:"Vamsi Bapat", Grade:"VII"})

Check final documents:

db.Students.find().pretty()

Case 2:

Insert the document of "Hersch Gibbs" into the Students collection using the update() method with the upsert option.

Step 1: Check existing documents in the "Students" collection

Shows the existing documents with their _id, StudName, Grade, and Hobbies.

Step 2: Use update with upsert: false

```
db.Students.update(
  {_id:4, StudName:"Hersch Gibbs", Grade:"VII"},
  {$set: {Hobbies: "Graffiti"}},
  {upsert: false}
);
```

- No document is inserted because a document with _id:4 doesn't exist.
- Result shows nUpserted: 0 meaning no document was inserted.

Step 3: Use update with upsert: true

```
db.Students.update(
  {_id:4, StudName:"Hersch Gibbs", Grade:"VII"},
  {$set: {Hobbies: "Graffiti"}},
  {upsert: true}
);
```

- A new document with _id:4 is inserted.
- Result shows nUpserted: 1, meaning one document was inserted.

Step 4: Confirm the new document

db.Students.find() now shows the new document of "Hersch Gibbs" with the Hobbies: "Graffiti" included.

Case 3: Add a new field to an existing document-Update Method

Syntax of update method

Add a new field "Location" with value "Newark" to the document with _id:4 in the "Students" collection.

Check the document with _id:4 before updating:

db.Students.find({_id:4}).pretty();

Add the new field "Location" with the value "Newark":

```
db.Students.update(
  {_id:4},
  {$set: {Location: "Newark"}}
);
```

Confirm the new field has been added:

db.Students.find({_id:4}).pretty();

Case 4: Removing an Existing Field from an Existing Document – Remove Method

To remove the field "Location" with the value "Newark" from a document with _id: 4 in the Students collection.

Inspect the current document: **db.Students.find({_id:4}).pretty()**

Execute the update command to remove the "Location" field:

db.Students.update({_id:4}, { $unset: { Location: "Newark" } })

Verify the document again:

db.Students.find({_id:4}).pretty()

Case5: Removing a Document with remove() Method **db.Students.remove({Age: {$gt: 18}})**

- Removes all documents in the Students collection where the Age is greater than 18.

Case 6: Removing a Field from a Document with $unset **db.Students.update({_id: 4}, {$unset: {Location: "Newark"}})**

- Removes the Location field from the document with _id: 4.

Case 7:Updating Documents with $set **db.Students.update({_id: 4}, {$set: {Grade: "X"}})**

- Updates the Grade field of the document with _id: 4 to "X".

Case 8: Adds a new field Sports with the value "Football" to the document with _id: 4.

db.Students.update({_id: 4}, {$set: {Sports: "Football"}})

Case 9: Updates all documents where Grade is "VII" by setting Sports to "Cricket".

Using multi: true to Update Multiple Documents

```
db.Students.update(
  {Grade: "VII"},
  {$set: {Sports: "Cricket"}},
  {multi: true}
)
```

Case 9: Replacing an Entire Document:
Replaces the entire document with _id: 4.

```
db.Students.replaceOne(
  {_id: 4},
  {
    _id:          4,
    Grade: "X",
    StudName:   "Hersch   Gibbs",
    Hobbies: "Graffiti",
    Sports:  "Football"
  }}
```

Case 10: Upsert Operation (update + upsert: true)

```
db.Students.update(
  {_id: 5},
  {$set: {StudName: "Paul Adams", Grade: "VIII"}},
  {upsert: true}
)
```

- If a document with _id: 5 doesn't exist, MongoDB inserts it with the specified fields.

Case 10: Finding Elements Based on Some Criteria – findOne() Method

To retrieve a **single document** from the "Students" collection where a specific field (e.g., Grade) matches a value.

Command:

```
db.Students.findOne({Grade: "VII"})
```

- This will return **only one document** (not all matches).
- Equivalent in SQL:

```
SELECT * FROM Students WHERE Grade = 'VII' LIMIT 1;
```

Case 11: Finding Specific Elements – find() with Projections

```
db.Students.find({Grade: "VII"}, {StudName: 1, _id: 0})
```

- This will return only the StudName of students in Grade "VII".
- _id: 0 hides the _id field.

Case 12: Sort documents in ascending or descending order by a specified field.

```
db.Students.find().sort({Grade: 1})    //  Ascending
db.Students.find().sort({Grade: -1}) // Descending
```

Case 13: Retrieve only a certain number of documents

```
db.Students.find().limit(3)
```

- Returns the **first 3 documents** from the collection.

Case 14: **Finding Documents based on Search Criteria - Find Method Objective:**

Find the document wherein the "StudName" has value "Aryan David”.

db.Students.find({StudName:"Aryan David"});

To format the above output, use the pretty() method:

db.Students.find({StudName:"Aryan David"}).pretty();

Case 15: To display only the StudName from all the documents of the Student's collection. The identifier "_id" should be suppressed and NOT displayed.

db.Students.find({}, {StudName: 1,_id:0});

Case 16: To display only the StudName and Grade from all the documents of the Students collection. The identifier _id should be suppressed and NOT displayed.

db.Students.find({}, {StudName:1,Grade: 1,_id:0});

Case 17: To find those documents where the Grade is set to 'VII'.

db.Students.find({Grade: {$eq:'VII'}}).pretty();

Case 18: To find those documents where the Grade is NOT set to 'VII'.

db.Students.find({Grade: {$ne: 'VII'}}).pretty();

Case 19: To find those documents from the Students collection where the Hobbies is set to either 'Chess' or is set to 'Skating'.

db.Students.find ({Hobbies :{ $in: ['Chess', 'Skating']}}).pretty ();

Case 20: To find those documents from the Students collection where the Hobbies is set neither to 'Chess' nor is set to 'Skating'

db.Students.find({Hobbies :{ $nin: ['Chess','Skating']}}).pretty ();

Case 21: To find those documents from the Students collection where the Hobbies is set to 'Graffiti' and the StudName is set to 'Hersch Gibbs' (AND condition).

db.Students.find({Hobbies:'Graffiti', StudName: 'Hersch Gibbs'}).pretty();

Case 22: To find documents from the Students collection where the StudName begins with "M".

db.Students.find({StudName:/^M/}).pretty();

Case 23: To find documents from the Students collection where the StudName ends in "s”

db.Students.find({StudName:/s$/}).pretty();

Case 24: To find documents from the Students collection where the StudName has an "e" in any position.

db.Students.find({StudName:/e/}).pretty(); OR

```
db.Students.find({StudName:/.*e.*/}).pretty();
```

OR

```
db.Students.find({StudName:        {$regex:"e"}}).pretty();
```

Case 25: To find documents from the Students collection where the StudName ends in "a".

```
db.Students.find({StudName:        {$regex:"a$"}}).pretty();
```

Case 26: To find documents from the Students collection where the StudName begins with "M".

```
db.Students.find({StudName:{$regex:"^M"}}).pretty();
```

Case 26: Dealing with NULL Values

To add or manage a field (Location) with a NULL value in documents of the Students collection.

- NULL indicates a missing or unknown value.
- This is useful when we don’t know the value at the moment but may update it later.

Step 1: Viewing Existing Documents

To view specific documents before updating:

```
db.Students.find({$or: [{_id: 3}, {_id: 4}]})
```

Step 2: Adding NULL Values

To insert a NULL value in the "Location" field:

```
db.Students.update({_id: 3}, {$set: {Location: null}});
db.Students.update({_id: 4}, {$set: {Location: null}});
```

Step 3: Searching for NULL Values

To find documents where Location is NULL or does not exist:

db.Students.find({Location: {$eq: null}});

Step 4: Removing Fields with NULL Values

To **remove the Location field** where it's NULL:
db.Students.update({_id: 3}, {$unset: {Location: null}});
db.Students.update({_id: 4}, {$unset: {Location: null}});

Step 5: Confirming the Change
To verify that the fields have been removed:

db.Students.find()

Case 27: Count, Limit, Sort, and Skip

To find the number of documents in the Students collection.

db.Students.count()

To find the number of documents in the Students collection wherein the Grade is VII.
db.Students.count({Grade:"VII"});

To retrieve the first 3 documents from the Students collection wherein the Grade is VII.
db.Students.find({Grade:"VII"}).limit(3).pretty();

Case 28: Sort documents in asceneding order of StudName
db.Students.find().sort({StudName:1}).pretty();

Sort documents in descending order of StudName
db.Students.find().sort({StudName:-1}).pretty();

To sort the documents from the Students collection first on Grade in ascending

order and then on Hobbies in descending order.

```
db.Students.find().sort((Grade:1,   Hobbies:-1)).pretty();
```

To sort the documents from the Students collection first on Grade in ascending order and then on Hobbies in ascending order.

db.Students.find().sort((Grade:1,   Hobbies:1}).pretty();

To skip the first 2 documents from the Students collection.

db.Students.find().skip (2).pretty();

To sort the documents from the Students collection and skip the first document from the output.

```
db.Students.find().skip  (1).pretty().sort({StudName:1});
```

To display the last 2 records from the Students collection.

db.Students.find().pretty().skip(db.Students.count()-2);

To retrieve the third, fourth, and fifth document from the Students collection.
db.Students.find().pretty().skip(2).limit(3);

Case 28: Arrays

To create a collection by the name "food" and then insert documents into the "food" collection. Each document should have a "fruits" array.

```
db.food.insert({_id:1,fruits:[   'banana','apple',   'cherry'   ]   })

db.food.insert({_id:2,fruits:[   'orange','butterfruit','mango'   ]}) db.food.insert({_id:3,fruits:[
'pineapple',         'strawberry','grapes']});         db.food.insert({_id:4,fruits:[          'banana',
'strawberry','grapes']});
db.food.insert((_id:5,fruits:  [  'orange','grapes']});
```

To find those documents from the "food" collection which has the "fruits array" constituted of "banana", "apple" and "cherry".

```
db.food.find({fruits: ['banana','apple', 'cherry']}).pretty()
```

To find those documents from the "food" collection which has the "fruits" array having "banana", as an element

```
db.food.find({fruits:'banana'})
```

To find those documents from the "food" collection which have the "fruits" array having "grapes" in the first index position. The index position begins at 0.

```
db.food.find({'fruits. 1':'grapes'})
```

To find those documents from the "food" collection where "grapes" is present in the 2nd index position of the "fruits" array.

```
db.food.find({'fruits.2':'grapes'})
```

To find those documents from the "food" collection where the size of the array is two. The size implies that the array holds only 2 values.

```
db.food.find({"fruits":{$size:2}})
```

To find those documents from the "food" collection where the size of the array is three. The size implies that the array holds only 3 values.

```
db.food.find({"fruits":{$size:3}})
```

To find the document with (id: 1) from the "food" collection and display the first two elements from the array "fruits".

db.food.find({_id:1},{"fruits":{$slice:2}})

To find all documents from the "food" collection which have elements "orange" and "grapes" in the array "fruits".

db.food.find ((fruits: {$all: ["orange", "grapes"]}}).pretty ();

To find those documents from the "food" collection which have the element "orange" in the 0th index position in the array "fruits".

db.food.find({ "fruits.0" : "orange" }).pretty();

To find the document with (id: 1) from the "food" collection and display two elements from the array "fruits", starting with the element at 0th index position.

db.food.find({id:1},{"fruits": {$slice: [0,2]}})

To find the document with (id: 1) from the "food" collection and display two elements from the array "fruits", starting with the element at 1" index position.

db.food.find({_id:1},{"fruits": {$slice:[1,2]}})

To find the document with (id: 1) from the "food" collection and display three elements from the array "fruits", starting with the element at 2nd index position. Since we have only 3 elements in the array "fruits" for the document with _id:1, it displays only one element, the element at 2nd index position, that is, "cherry".

db.food.find({_id:1},{"fruits": {$slice: [2,3]}})

Case 30: Update on the Array

To update the document with "_id:4" and replace the element present in the 1st index position of the "fruits" array with "apple".

db.food.update({_id:4}, {$set:{'fruits.1': 'apple'}})

To update the document with "_id:4" and replace the element present in the 1st index position of the "fruits" array with "apple".

db.food.update({_id:4}, {$set:{'fruits.1': 'apple'}})

To update the document with "_id:1" and replace the element "apple" of the "fruits" array with "An apple".
Act:

db.food.update({_id:1, 'fruits':'apple'}, {$set: {'fruits.$': 'An apple' }})

To update the document with "_id:2" and push new key value pairs in the "fruits" array.
Act:

db.food.update({_id:2},{$push:{price:{orange:60,butterfruit:200,mango: 120}}})

To update the document with "_id:4" by adding an element "orange" to the list of elements in the array "fruits".
Act:

db.food.update({_id:4}, {$addToSet: {fruits:"orange"}});

To update the document with "_id:4" by popping an element from the list of elements present in the array "fruits". The element popped is the one from the end of the array.
Act:

db.food.update({_id:4},{$pop: {fruits:1}});

To update the document with "_id:4" by popping an element from the list of elements present in the array "fruits". The element popped is the one from the beginning of the:
Act:

db.food.update({_id:4}, {$pop:{fruits:-1}});

To update the document with "_id:3" by popping two elements from the list of elements present in the array "fruits". The elements popped are "pineapple" and "grapes".

The document with "_id:3" before the update is

```
db.food.update({_id:3},{$pullAll:{fruits:    [    'pineapple','grapes'    ]}});
```

To update the documents having "banana" as an element in the the element "banana" from those documents.
The "food" collection before the update is as follows:

```
db.food.update({fruits:'banana'},   {$pull:{fruits:'banana'}})
```

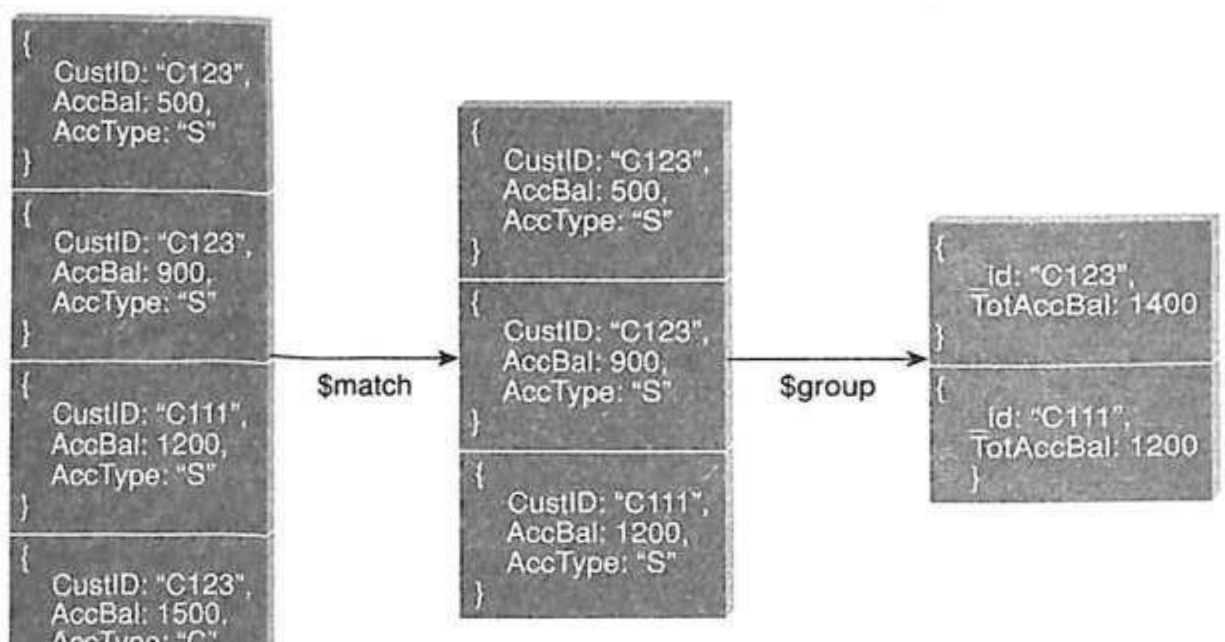
To pull out an array element based on index position.

There is no direct way of pulling the array elements by looking up their index numbers. However a workaround is available. The document with "_id:4" in the food collection prior to the update is as follows:

```
db.food.update({_id:4}, {$unset: {"fruits. 1": null }});
db.food.update({_id:4}, {$pull: {"fruits": null}});
```

Case 31: Aggregate Function

Objective: Consider the collection "Customers" as given below. It has four documents. We would like to filter out those documents where the "AccType" has a value other than "S". After the filter, we should be left with three documents where the "Acctype": "S". It is then required to group the documents on the basis of CustID and sum up the "AccBal" for each unique "CustID". This is similar to the output received with group by clause in RDBMS. Once the groups have been formed [as per the example below, there will be only two groups: (a) "CustID" : "C123" and (b) "CustID" : "C111"], filter and display that group where the "TotAccBal" column has a value greater than 1200.


{ CustID: "C123", AccBal: 500, AccType: "S" }
{ CustID: "C123", AccBal: 900, AccType: "S" }
{ CustID: "C111", AccBal: 1200, AccType: "S" }
{ CustID: "C123", AccBal: 1500, AccType: "C"
$match
{ CustID: "C123", AccBal: 500, AccType: "S" }
{ CustID: "C123", AccBal: 900, AccType: "S" }
{ CustID: "C111", AccBal: 1200, AccType: "S" }
$group
{ _id: "C123", TotAccBal: 1400 }
{ _id: "C111", TotAccBal: 1200 }

Let us start off by creating the collection “Customers" with the above displayed four documents:

```
db.Customers.insert([{CustID:"C123",AccBal:500,AccType:"S"},
{CustID:"C123", AccBal: 900, AccType:"S"},
{CustID:"C111", AccBal: 1200, AccType:"S"},
{CustID:"C123",  AccBal:  1500,  AccType:"C"}]);
```

To confirm the presence of four documents in the "Customers" collection, use the below syntax:

```
db.Customers.find().pretty();
```

To group on "CustID" and compute the sum of "AccBal", use the below syntax:

```
db.Customers.aggregate({$group:{_id:"$CustID",TotAccBal:{$sum:"$AccBal"
}}});
```

In order to first filter on "AccType:S" and then group it on "CustID" and then compute the sum of "AccBal", use the below syntax:

```
db.Customers.aggregate(  {  $match:  {AccType:  "S"  }  },
{$group:  {  _id:  "$CustID",TotAccBal:  {  $sum  :  "$AccBal"  }  }  });
```

In order to first filter on "AccType:S" and then group it on "CustID" and then to compute the sum of "AccBal" and then filter those documents wherein the "TotAccBal" is greater than 1200, use the below syntax:

```
db.Customers.aggregate( { $match : {AccType : "S" } },
{$group:  {  _id:  "$CustID",TotAccBal:  {  $sum:  "$AccBal"  }  }  },  {  $match:
{TotAccBal  :  {  $gt:  1200  }  }});
```

To group on "CustID" and compute the average of the "AccBal" for each group:

```
db.Customers.aggregate({ $group: { _id: "$CustID", TotAccBal : { $avg: "$AccBal" } }
});
```

To group on "CustID" and determine the maximum "AccBal" for each group:
db.Customers.aggregate({ $group: { _id: "$CustID", TotAccBal: { $max : "$AccBal" } } });

To group on "CustID" and determine the minimum "AccBal" for each group:
db.Customers.aggregate({$group: { _id: "$CustID", TotAccBal: { $min : "$AccBal" } } });