

## Hadoop Installation Steps STEP 1 – Separate Login

Hit CTRL+ALT+T to get started. We will install Hadoop from the terminal. For new Linux users, things might get confusing while installing different programs and managing them from the same login. If you are one of them, we have a solution. Let's create a new dedicated Hadoop user. Whenever you want to use Hadoop, just use the separate login. Simple.

```
$ sudo addgroup hadoop
```

```
$ sudo adduser --ingroup hadoop hduser
```

Note: You do not have to write passwords or names. Just hit enter and press 'y' at the end.

```
dhruvil@linux:~$ sudo addgroup hadoop
[sudo] password for dhruvil:
Adding group `hadoop' (GID 1001) ...
Done.
dhruvil@linux:~$ sudo adduser --ingroup hadoop hduser
Adding user `hduser' ...
Adding new user `hduser' (1001) with group `hadoop' ...
Creating home directory `/home/hduser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for hduser
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] y
dhruvil@linux:~$ sudo adduser hduser sudo
Adding user `hduser' to group `sudo' ...
Adding user hduser to group sudo
Done.
dhruvil@linux:~$
```

Add Hadoop user to sudo group (Basically, grant it all permissions)

```
$ sudo adduser hduser sudo
```

## STEP 2 – Getting Environment Ready

In order to run perfectly, Hadoop needs basic two things in Ubuntu environment. First is *Java 1.6 or higher* because Hadoop's distributed processing and storage are written in Java. Second is *ssh(Secure Shell)* for the security of the communication between the nodes of the cluster. It is always better to update the package list before installing anything new. Let's get started. Update package list,

```
$ sudo apt-get update
```

### 2.1 Install JAVA

Next, we will install the default java development kit "OpenJDK"

```
$ sudo apt-get install default-jdk
```

Once it is installed, check the java version. I have 1.8 installed which is higher than the required 1.6 so we are good to go.

```
$ java -version
```

```
brad@Brad-Laptop:~$ java -version
openjdk version "1.8.0_45-internal"
OpenJDK Runtime Environment (build 1.8.0_45-internal-b14)
OpenJDK 64-Bit Server VM (build 25.45-b02, mixed mode)
brad@Brad-Laptop:~$
```

## 2.2 Install SSH

\$ sudo apt-get install ssh

Passwordless entry for localhost using SSH

\$ su hduser

\$ sudo ssh-keygen -t rsa

Note: When ask for file name or location, leave it blank.

\$ cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

\$ chmod 0600 ~/.ssh/authorized\_keys

```
hduser@linux:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory '/home/hduser/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:+tRQbsCSStXDG9cQ/CViWUFMwjFCVf75K+p4Dhy5uho hduser@linux
The key's randomart image is:
+----[RSA 2048]-----+
|      .+o+B%*.      |
|      . o=.B++ .    |
|      . o o*.o.o     |
|      . . ..+. . .   |
|      .   Soo   o    |
|      . .+o      .   |
|      E. .+.      .  |
|      .o. o. . .    |
|      ..oo.++. . .  |
+-----[SHA256]-----+
hduser@linux:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
hduser@linux:~$ chmod 0600 ~/.ssh/authorized_keys
hduser@linux:~$ ssh localhost
```

Check if ssh works,

\$ ssh localhost

```

hduser@linux:~$ chmod 0600 ~/.ssh/authorized_keys
hduser@linux:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:ptBvypK0h5wfmnq8L1AesWceMhw/t5T3OvCQsdSa1LY.
Are you sure you want to continue connecting (yes/no)? y
Please type 'yes' or 'no': yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 17.10 (GNU/Linux 4.13.0-21-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

138 packages can be updated.
80 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

hduser@linux:~$

```

Once we are logged in localhost, exit from this session using following command.

\$ exit

```

hduser@linux:~$ exit
logout
Connection to localhost closed.
hduser@linux:~$

```

## STEP 3 – Install Hadoop on Ubuntu

The environment is now ready to install hadoop on ubuntu. Moreover, the procedure to install hadoop on linux (for the newer versions) will remain same. Only the folder name changes from hadoop-3.0.0 to hadoop-x.y.z

### 3.1 Download Hadoop

\$ wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-3.0.0/hadoop-3.0.0.tar.gz

```

dhrumil@linux:~$ wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-3.0.0/hadoop-3.0.0.tar.gz
--2018-03-12 18:24:41-- http://mirrors.sonic.net/apache/hadoop/common/hadoop-3.0.0/hadoop-3.0.0.tar.gz
Resolving mirrors.sonic.net (mirrors.sonic.net)... 157.131.0.16
Connecting to mirrors.sonic.net (mirrors.sonic.net)|157.131.0.16|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 306392917 (292M) [application/x-gzip]
Saving to: 'hadoop-3.0.0.tar.gz'

hadoop-3.0.0.tar.gz      100%[=====] 292.20M  114KB/s   in 91m 27s

2018-03-12 19:56:09 (54.5 KB/s) - 'hadoop-3.0.0.tar.gz' saved [306392917/306392917]

dhrumil@linux:~$

```

Unzip it

```
$ tar xvzf hadoop-3.0.0.tar.gz
```

### 3.2 Hadoop Configuration

Make a directory called hadoop and move the folder 'hadoop-3.0.0' to this directory

```
$ sudo mkdir -p /usr/local/hadoop
```

```
$ cd hadoop-3.0.0/
```

```
$ sudo mv * /usr/local/hadoop
```

```
$ sudo chown -R hduser:hadoop /usr/local/hadoop
```

```
hduser@linux:/usr/local/hadoop$ cd hadoop-3.0.0
hduser@linux:/usr/local/hadoop/hadoop-3.0.0$ sudo mv * /usr/local/hadoop
hduser@linux:/usr/local/hadoop/hadoop-3.0.0$ ls
hduser@linux:/usr/local/hadoop/hadoop-3.0.0$ sudo chown -R hduser:hadoop /usr/local/hadoop
hduser@linux:/usr/local/hadoop/hadoop-3.0.0$ cd ~
hduser@linux:~$
```

### STEP 4 – Setting up Configuration files

We will change content of following files in order to complete hadoop installation.

1. ~/.bashrc
2. hadoop-env.sh
3. core-site.xml
4. hdfs-site.xml
5. yarn-site.xml

#### 4.1 ~/.bashrc

If you don't know the path where java is installed, first run the following command to locate it

```
$update-alternatives --config java
```

```
File Edit View Search Terminal Help
hduser@linux:~$ update-alternatives --config java
There is only one alternative in link group java (providing /usr/bin/java): /usr/lib/jvm/java-8-
openjdk-amd64/jre/bin/java
Nothing to configure.
hduser@linux:~$
```

Now open the ~/.bashrc file

```
$sudo nano ~/.bashrc
```

Note: I have used 'nano' editor, you can use a different one. No issues.

Now once the file is opened, append the following code at the end of file,

```
#HADOOP VARIABLES START
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```
export HADOOP_HOME=/usr/local/hadoop export
```

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



```

PATH=$PATH:$HADOOP_HOME/sbin export

HADOOP_MAPRED_HOME=$HADOOP_HOME export

HADOOP_COMMON_HOME=$HADOOP_HOME export

HADOOP_HDFS_HOME=$HADOOP_HOME export

YARN_HOME=$HADOOP_HOME

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native

export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"

#HADOOP VARIABLES END

```

GNU nano 2.8.6 File: /home/hduser/.bashrc Modified

```

# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

#HADOOP VARIABLES START
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
#HADOOP VARIABLES END

```

^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos  
 ^X Exit       ^R Read File    ^\ Replace    ^U Uncut Text    ^T To Spell    ^\_ Go To Line

Press CTRL+O to save and CTRL+X to exit from that window.

Update .bashrc file to apply changes

```
$source ~/.bashrc
```

## 4.2 hadoop-env.sh

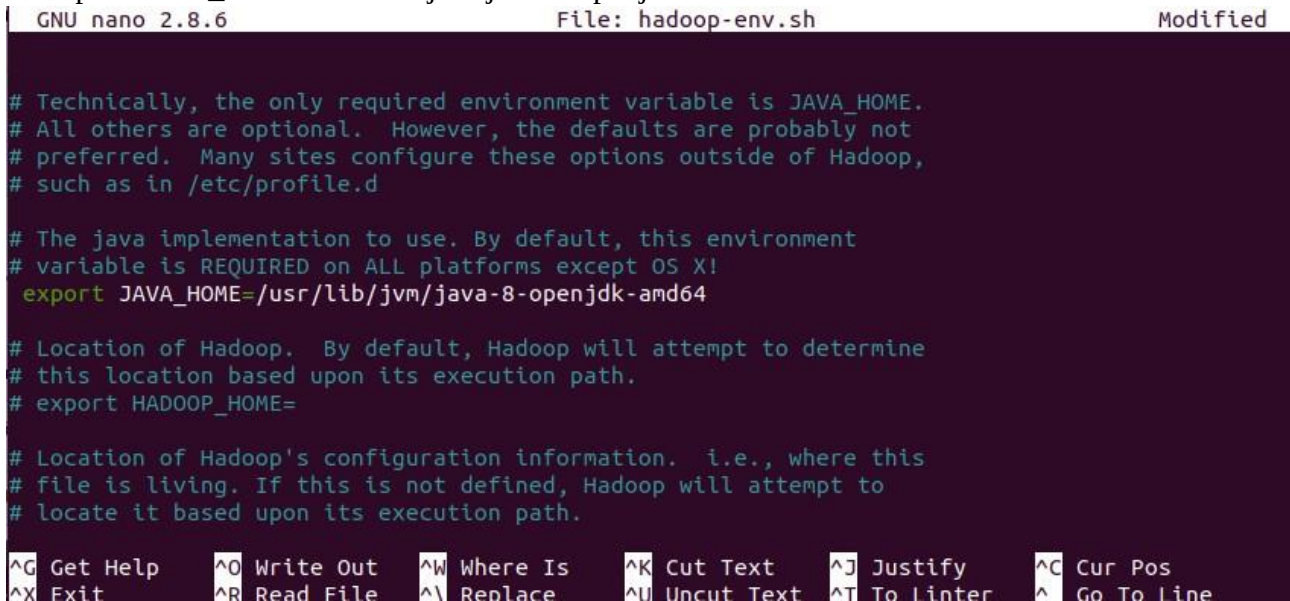
We need to tell Hadoop the path where java is installed. That's what we will do in this file, specify the path for JAVA\_HOME variable.

Open the file,

```
$sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

Now, the first variable in file will be JAVA\_HOME variable, change the value of that variable to

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```



```
GNU nano 2.8.6 File: hadoop-env.sh Modified

# Technically, the only required environment variable is JAVA_HOME.
# All others are optional.  However, the defaults are probably not
# preferred.  Many sites configure these options outside of Hadoop,
# such as in /etc/profile.d

# The java implementation to use.  By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

# Location of Hadoop.  By default, Hadoop will attempt to determine
# this location based upon its execution path.
# export HADOOP_HOME=

# Location of Hadoop's configuration information.  i.e., where this
# file is living.  If this is not defined, Hadoop will attempt to
# locate it based upon its execution path.

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Linter ^_ Go To Line
```

### 4.3 core-site.xml

Create temporary directory

```
$ sudo mkdir -p /app/hadoop/tmp
```

```
$ sudo chown hduser:hadoop /app/hadoop/tmp
```

Open the file,

```
$ sudo nano /usr/local/hadoop/etc/hadoop/core-site.xml
```

Append the following between configuration tags. Same as below.

```
<configuration>
```

```
<property>
```

```
<name>hadoop.tmp.dir</name>
```

```
<value>/app/hadoop/tmp</value>
```

```
<description>A base for other temporary directories.</description>
```

```
</property>
```

```
<property>
```

```
<name>fs.default.name</name>
```

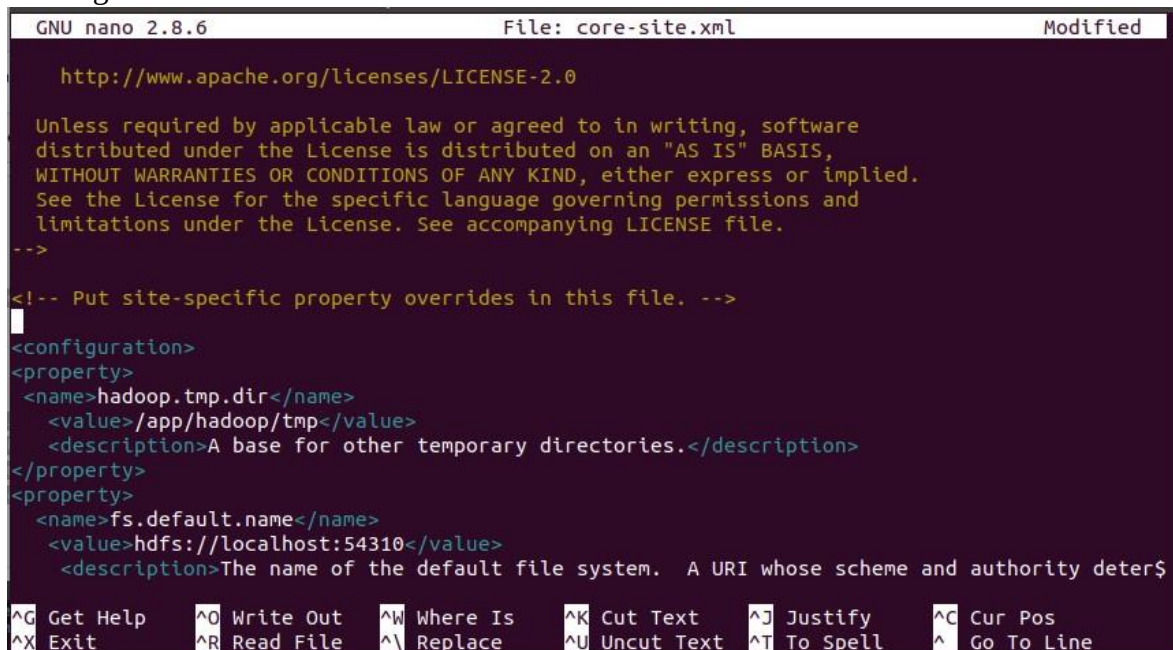
```
<value>hdfs://localhost:54310</value>
```

```
<description>The name of the default file system.  A URI whose scheme and authority
determine the FileSystem implementation.  The uri's scheme determines the config property
```

(fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.</description>

</property>

</configuration>



```
GNU nano 2.8.6 File: core-site.xml Modified

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:54310</value>
    <description>The name of the default file system. A URI whose scheme and authority deter$
  </property>
</configuration>
```

#### 4.4 hdfs-site.xml

Mainly there are two directories,

1. Name Node
2. Data Node

Make directories

```
$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
```

```
$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
```

```
$ sudo chown -R hduser:hadoop /usr/local/hadoop_store
```

Open the file,

```
$ sudo nano /usr/local/hadoop/etc/hadoop/hdfs-site.xml
```

Change the content between configuration tags shown as below.

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
    <description>Default block replication.The actual number of replications can be
specified when the file is created. The default is used if replication is not specified in create
time  </description>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
```

```

    <value>file:/usr/local/hadoop_store/hdfs/namenode</value>
  </property>
</property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/usr/local/hadoop_store/hdfs/datanode</value>
</property>
</configuration>

```

#### 4.5 yarn-site.xml

Open the file,

```
$sudo nano /usr/local/hadoop/etc/hadoop/yarn-site.xml
```

Just like the other two, add the content to configuration tags.

```

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce_shuffle</value>

</property>

</configuration>

```

#### STEP 5- Format Hadoop file system

Hadoop installation is now done. All we have to do is change format the name-nodes before using it.

```
$ hadoop namenode -format
```

```

s = 1,5,25
2018-03-13 22:15:30,631 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2018-03-13 22:15:30,631 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap
and retry cache entry expiry time is 600000 millis
2018-03-13 22:15:30,633 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2018-03-13 22:15:30,633 INFO util.GSet: VM type = 64-bit
2018-03-13 22:15:30,634 INFO util.GSet: 0.0299999999329447746% max memory 1.7 GB = 538.5 KB
2018-03-13 22:15:30,634 INFO util.GSet: capacity = 2^16 = 65536 entries
2018-03-13 22:15:30,689 INFO namenode.FSImage: Allocated new BlockPoolId: BP-254327528-127.0
.1.1-1520959530679
2018-03-13 22:15:30,828 INFO common.Storage: Storage directory /usr/local/hadoop_store/hdfs/
namenode has been successfully formatted.
2018-03-13 22:15:30,860 INFO namenode.FSImageFormatProtobuf: Saving image file /usr/local/ha
dooop_store/hdfs/namenode/current/fsimage.ckpt_00000000000000000000 using no compression
2018-03-13 22:15:31,012 INFO namenode.FSImageFormatProtobuf: Image file /usr/local/hadoop_st
ore/hdfs/namenode/current/fsimage.ckpt_00000000000000000000 of size 391 bytes saved in 0 seco
nds.
2018-03-13 22:15:31,078 INFO namenode.NNStorageRetentionManager: Going to retain 1 images wi
th txid >= 0
2018-03-13 22:15:31,084 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at linux/127.0.1.1
*****/
hduser@linux:~$

```

#### STEP 6- Start Hadoop daemons

Now that hadoop installation is complete and name-nodes are formatted, we can start hadoop by going to following directory. \$ cd /usr/local/hadoop/sbin



\$ start-all.sh

```
hduser@linux:~$ cd /usr/local/hadoop/sbin
hduser@linux:/usr/local/hadoop/sbin$ ls
distribute-exclude.sh  refresh-namenodes.sh  start-yarn.cmd  stop-secure-dns.sh
FederationStateStore  start-all.cmd         start-yarn.sh   stop-yarn.cmd
hadoop-daemon.sh      start-all.sh         stop-all.cmd   stop-yarn.sh
hadoop-daemons.sh    start-balancer.sh     stop-all.sh    workers.sh
httpfs.sh             start-dfs.cmd         stop-balancer.sh yarn-daemon.sh
kms.sh                start-dfs.sh          stop-dfs.cmd   yarn-daemons.sh
mr-jobhistory-daemon.sh start-secure-dns.sh  stop-dfs.sh
hduser@linux:/usr/local/hadoop/sbin$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hduser in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [linux]
linux: Warning: Permanently added 'linux' (ECDSA) to the list of known hosts.
2018-03-13 22:16:18,991 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
Starting resourcemanager
Starting nodemanagers
hduser@linux:/usr/local/hadoop/sbin$
```

Just check if all daemons are properly started using the following command:

\$ jps

## STEP 7 – Stop Hadoop daemons

Step 7 of hadoop installation is when you need to stop Hadoop and all its modules.

```
hduser@linux:/usr/local/hadoop/sbin$ jps
8001 SecondaryNameNode
8242 ResourceManager
8758 Jps
7735 DataNode
8569 NodeManager
7567 NameNode
hduser@linux:/usr/local/hadoop/sbin$
```

\$ stop-all.sh

Appreciate yourself because you've done it. You have completed all the Hadoop installation steps and Hadoop is now ready to run the first program.

## Let's run MapReduce job on our entirely fresh Hadoop cluster setup

Go to the following directory

\$ cd /usr/local/hadoop

Run the following command `hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduceexamples-3.0.0.jar pi 2 5`

```
Reduce input groups=2
Reduce shuffle bytes=56
Reduce input records=4
Reduce output records=0
Spilled Records=8
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=17
Total committed heap usage (bytes)=1178075136

Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=236
File Output Format Counters
  Bytes Written=97
Job Finished in 4.49 seconds
Estimated value of Pi is 3.6000000000000000000000000000
hduser@linux:/usr/local/hadoop$
```

## Hadoop Commands :

**\$ su hduser**

Password:

- **\$ start-dfs.sh**:-This command is used to start or activate Hadoop file system.
- **\$ jps**  
3424 SecondaryNameNode  
4097 Jps  
2969 NameNode  
3145 DataNode
- **\$ hadoop fs -ls /** :-This command is used to display the records present in the Hadoop.  
Found 1 items  
drwxr-xr-x - hduser supergroup 0 2024-04-05 16:23 /user
- **\$ hadoop fs -mkdir /22mca09** :- This command is used to create our own directory.   
**\$ ls**:- Displays the content present in the local file system.  
sample1 sample2   
**\$ hadoop fs -ls /**  
Found 2 items  
drwxr-xr-x - hduser supergroup 0 2024-04-05 12:02 /22mca09  
drwxr-xr-x - hduser supergroup 0 2024-04-05 16:23 /user
- **\$ pwd**  
/home/hduser/lab
- **\$ hadoop fs -copyFromLocal sample1 /22mca09**:- This command is used to copy a file from local file system to Hadoop file system.
- **\$ hadoop fs -ls /**  
Found 2 items  
drwxr-xr-x - hduser supergroup 0 2024-04-05 12:11 /22mca09  
drwxr-xr-x - hduser supergroup 0 2024-04-05 16:23 /user
- **\$ hadoop fs -ls /22mca09**  
Found 1 items  
-rw-r--r-- 1 hduser supergroup 19 2024-04-05 12:11 /22mca09/sample1
- **\$ hadoop fs -copyFromLocal sample2 /22mca09**
- **\$ hadoop fs -ls /22mca09**  
Found 2 items  
-rw-r--r-- 1 hduser supergroup 19 2024-04-05 12:11 /22mca09/sample1  
-rw-r--r-- 1 hduser supergroup 22 2024-04-05 12:14 /22mca09/sample2
- **\$ hadoop fs -ls /22mca09**  
Found 2 items  
-rw-r--r-- 1 hduser supergroup 19 2024-04-05 12:11 /22mca09/sample1  
-rw-r--r-- 1 hduser supergroup 22 2024-04-05 12:14 /22mca09/sample2
- **\$ hadoop fs -cat /22mca09/sample1** :- This command is used to display the content present in the sample1 present in the 22mca09 directory of Hadoop.  
this  
is  
hadoop  
lab
- **\$ hadoop fs -cat /22mca09/sample2**:- This command is used to display the content present in the sample2.

this

is

hdfs

commands

- **\$ hadoop fs -mv /22mca09/sample1 /22mca09/sample3:-** This command is used to rename the file sample1 to sample 3.
- **\$ hadoop fs -ls /22mca09**  
Found 2 items  
-rw-r--r-- 1 hduser supergroup 22 2024-04-05 12:14 /22mca09/sample2  
-rw-r--r-- 1 hduser supergroup 19 2024-04-05 12:11 /22mca09/sample3
- **\$hadoop fs -copyToLocal /22mca09/sample3:-** This command is used to copy the file from Hadoop file system to the local file system.
- **\$ ls**  
sample1 sample2 sample3
- **\$ start-all.sh**  
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh □
- **\$ stop -dfs.sh:-** This command is used to stop the Hadoop file

system.

- **\$ hadoop fs -chmod ugo+rwX /22mca09/abc**
- **\$ hadoop fs -ls /22mca09/abc**  
-rwxrwxrwx 1 hduser supergroup 0 2024-04-18 16:26 /22mca09/abc
- **\$ hadoop fs -chmod go-rwx /22mca09/abc**
- **\$ hadoop fs -ls /22mca09/abc**  
-rwx----- 1 hduser supergroup 0 2024-04-18 16:26 /22mca09/abc
- **\$ hadoop fs -chmod go+x /22mca09/abc**
- **\$ hadoop fs -ls /22mca09/abc**  
-rwx--x--x 1 hduser supergroup 0 2024-04-18 16:26

/22mca09/abc

**\$ hadoop fs -du /22mca09 :-** It displays the size of the data of each record.

0 /22mca09/abc

0 /22mca09/lmn

108 /22mca09/naresh

108 /22mca09/naresm

108 /22mca09/naresp

22 /22mca09/sample2

19 /22mca09/sample3

- **\$ hadoop fs -du /**

365 /22mca09

0 /user

- **\$ ls\**

>

GNU bash, version 4.4.19(1)-release (x86\_64-pc-linux-gnu)

>

- **\$ ls**

Desktop Documents Downloads environments examples.desktop lab Music  
new3.py Pictures Public snap Templates Videos y165039



- **\$ cd lab**
- **\$ ls**  
abc lmn namesh namesn namesp sample1 sample2 sample3
- **\$ cat sample1** this is  
hadoop  
lab
- **\$ cat sample3** this is  
hadoop  
lab
- **\$ cat sample1** this is  
hadoop  
lab
- **\$ hadoop fs -appendToFile sample1 /22mca09/abc**
- **\$ hadoop fs -cat /22mca09/abc** this is  
hadoop  
lab
- **\$ hadoop fs -mkdir /revlab**
- **\$ hadoop fs -ls /** Found 5 items  
drwxr-xr-x - hduser supergroup 0 2024-04-18 10:02 /dummy  
drwxr-xr-x - hduser supergroup 0 2024-04-18 16:26 /22mca09  
drwxr-xr-x - hduser supergroup 0 2024-04-18 10:55 /revlab  
drwxr-xr-x - hduser supergroup 0 2024-04-18 16:23 /user  
drwxr-xr-x - hduser supergroup 0 2024-04-18 09:49 /y165006
- **\$ ls**  
a aa ab abc ac aik bbc bik bin ddc f1.txt f2.txt fin min
- **\$ hadoop fs -put f1.txt /revlab:-** This command is used to move the file from local file system to the Hadoop file system.
- **\$ hadoop fs -ls /revlab**  
Found 1 items  
-rw-r--r-- 1 hduser supergroup 36 2024-04-18 10:59 /revlab/f1.txt
- **\$ hadoop fs -copyFromLocal f2.txt /revlab**
- **\$ hadoop fs -ls /revlab**  
Found 2 items  
-rw-r--r-- 1 hduser supergroup 36 2024-04-18 10:59 /revlab/f1.txt  
-rw-r--r-- 1 hduser supergroup 52 2024-04-18 11:00 /revlab/f2.txt
- **\$ hadoop fs -cat /revlab/f1.txt** Raghu chandra sai naveen yanamadala
- **\$ hadoop fs -appendToFile f1.txt f2.txt /revlab/f3.txt**
- **\$ hadoop fs -cat /revlab/f3.txt** Raghu chandra sai naveen yanamadala what are you doing man what the hell are you doing
- **\$ hadoop fs -cp /revlab/f3.txt /revlab/f4.txt**
- **\$ hadoop fs -ls /revlab**  
Found 4 items  
-rw-r--r-- 1 hduser supergroup 36 2024-04-18 10:59 /revlab/f1.txt  
-rw-r--r-- 1 hduser supergroup 52 2024-04-18 11:00 /revlab/f2.txt  
-rw-r--r-- 1 hduser supergroup 88 2024-04-18 11:40 /revlab/f3.txt  
-rw-r--r-- 1 hduser supergroup 88 2024-04-18 11:43 /revlab/f4.txt
- **\$ hadoop fs -mv /revlab/f4.txt /revlab/f5.txt**

- **\$ hadoop fs -ls /revlab**  
Found 4 items
 

-rw-r--r--	1	hduser	supergroup	36	2024-04-18 10:59	/revlab/f1.txt
-rw-r--r--	1	hduser	supergroup	52	2024-04-18 11:00	/revlab/f2.txt
-rw-r--r--	1	hduser	supergroup	88	2024-04-18 11:40	/revlab/f3.txt
-rw-r--r--	1	hduser	supergroup	88	2024-04-18 11:43	/revlab/f5.txt
- **\$ hadoop fs -mv /revlab/f3.txt /revlab/f6.txt**
- **\$ hadoop fs -copyToLocal /revlab/f5.txt**
- **\$ hadoop fs -get /revlab/f6.txt**
- **\$ ls -l > zxc**
- **\$ hadoop fs -moveFromLocal zxc /revlab**
- **\$ hadoop fs -ls /revlab**  
Found 5 items
 

-rw-r--r--	1	hduser	supergroup	36	2024-04-18 10:59	/revlab/f1.txt
-rw-r--r--	1	hduser	supergroup	52	2024-04-18 11:00	/revlab/f2.txt
-rw-r--r--	1	hduser	supergroup	88	2024-04-18 11:43	/revlab/f5.txt
-rw-r--r--	1	hduser	supergroup	88	2024-04-18 11:40	/revlab/f6.txt
-rw-r--r--	1	hduser	supergroup	815	2024-04-18 11:58	/revlab/zxc
- **\$ hadoop fs -count /revlab**(count displays in number of directories in a given 22mca09 directory)
 

1	5	1079	/revlab
---	---	------	---------
- **\$ hadoop fs -mkdir /revlab/nlab**
- **\$ hadoop fs -ls /revlab**  
Found 6 items
 

-rw-r--r--	1	hduser	supergroup	36	2024-04-18 10:59	/revlab/f1.txt
-rw-r--r--	1	hduser	supergroup	52	2024-04-18 11:00	/revlab/f2.txt
-rw-r--r--	1	hduser	supergroup	88	2024-04-18 11:43	/revlab/f5.txt
-rw-r--r--	1	hduser	supergroup	88	2024-04-18 11:40	/revlab/f6.txt
drwxr-xr-x	-	hduser	supergroup	0	2024-04-18 12:16	/revlab/nlab
-rw-r--r--	1	hduser	supergroup	815	2024-04-18 11:58	/revlab/zxc
- **\$ hadoop fs -count /revlab**

2	5	1079	/revlab
---	---	------	---------
- **\$ hadoop fs -chmod 600 /revlab/zxc**
- **\$ hadoop fs -ls l /revlab/zxc**

-rw-----	1	hduser	supergroup	815	2024-04-18 11:58	/revlab/zxc
----------	---	--------	------------	-----	------------------	-------------

### WordCount Program in Hadoop :-

1. Open terminal first then
2. After type the pwd Command then hit the enter.
3. It shows the current directory .
4. save the below code in Current Directory name is WordCount.java **Source**

#### code of mapreduce in java:-

```
import java.io.IOException; import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration; import
org.apache.hadoop.fs.Path; import
org.apache.hadoop.io.IntWritable; import
org.apache.hadoop.io.Text; import
org.apache.hadoop.mapreduce.Job; import
org.apache.hadoop.mapreduce.Mapper; import
org.apache.hadoop.mapreduce.Reducer; import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount
{ public static class TokenizerMapperextends Mapper<Object, Text, Text, IntWritable>
    {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context) throws IOException,
        InterruptedException
        {
            StringTokenizer itr = new
            StringTokenizer(value.toString()); while
            (itr.hasMoreTokens()) { word.set(itr.nextToken());
            context.write(word, one);
            }
        }
    }
}

public static class IntSumReducerextends Reducer<Text,IntWritable,Text,IntWritable>
{ private IntWritable result = new IntWritable(); public void
    reduce(Text key, Iterable<IntWritable> values, Context
    context) throws IOException, InterruptedException
    { int sum = 0;
        for (IntWritable val : values)
        {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

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

```

Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

**After complete the above 4 steps run below codes in same terminal.**

- `echo $JAVA_HOME` (it shows the java path)
- `export PATH=${JAVA_HOME}/bin:${PATH}`
- `export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar` **Compile**

**WordCount.java and create a jar:**

- `$ hadoop com.sun.tools.javac.Main WordCount.java`
- `$ jar cf wc.jar WordCount*.class` **Assuming that:**
- `/user/joe/wordcount/input` - input directory in HDFS
- `/user/joe/wordcount/output` - output directory in HDFS **Sample text-files as**

**input:**

- `$ bin/hadoop fs -ls /user/input/`  
`/user/input/input1.txt`  
`/user/input/input2.txt`
- `$ bin/hadoop fs -cat /user/input/input1.txt` Hello World Bye World
- `$ bin/hadoop fs -cat /user/input/input2.txt`  
Hello Hadoop Goodbye Hadoop

**command is :-**

- `$ hadoop jar wc.jar WordCount /user/input.txt /output`

**Output:-**

- `hadoop fs -ls /output`  
Found 2 items  
`-rw-r--r-- 1 hduser supergroup 0 2024-05-13 23:15 /output/_SUCCESS`  
`-rw-r--r-- 1 hduser supergroup 56 2024-05-13 23:15 /output/part-r-00000`
- `hadoop fs -cat /output/part-r-00000`  
hadoop1  
hi 3  
hive 1  
is 3  
program 1  
this 3  
wountcount 1



### **PIG Installation Steps:-**

Please follow the below steps to install PIG on Ubuntu:

1. Open terminal and give the command is  
`$ cd Desktop`
2. Then command is  
`$ ls`
3. Type the command is  
`$ cp pig-0.16.0.tar.gz ..`
4. Then command is  
`$ pwd`
5. Then enter command is  
`$ tar -xzf pig-0.16.0.tar.gz`
6. Type the command  
`$ pwd`
7. Type command is `ls` to check the pig is available or not. `$ ls`
8. Type this command  
`$ nano .bashrc`
9. And then goto bottom of page in `.bashrc` file then copy type below code. #Set  
`PIG_HOME`  
`export PIG_HOME=/home/hduser/pig-0.16.0`  
`export PATH=$PATH:/home/hduser/pig-0.16.0/bin`  
`export`  
`PIG_CLASSPATH=$HADOOP_CONF_DIR`
10. Then save the file and goto terminal.
11. Enter the command is  
`$ source .bashrc`
12. Then after no error is display then enter the command is  
`$ pig -version`
13. Then complete the installation of pig .
14. After completed the installation of pig type the command in shell. `$ pig`
15. Then it opens the pig grunt shell prompt. Like below `grunt>`

## PIG Commands :

- \$ start-ll.sh**
- **\$ gedit student.tsv**
- **\$ gedit dept.tsv**
- **\$ cat student.tsv**  
1001 John 3.0  
1002 Jack 4.0  
1003 Smith 4.5  
1004 Scott 4.2  
1005 Joshi 3.5  
1006 Alex 4.5  
1007 Dravid 4.2  
1008 James 4.0  
1001 John 3.0  
1005 Joshi 3.0
- **\$ cat dept.tsv**  
1001 201 cse  
1002 201 cse  
1003 202 ece  
1004 202 ece  
1005 303 ele  
1006 303 ele  
1007 404 mec  
1008 404 mec
- **\$ hadoop fs -ls**  
Found 2 items  
-rw-r--r-- 1 hduser supergroup 0 2024-04-18 11:24 b  
-rw-r--r-- 1 hduser supergroup 0 2024-04-18 11:24 c
- **\$ hadoop fs -mkdir latin**
- **\$ hadoop fs -ls**  
Found 3 items  
-rw-r--r-- 1 hduser supergroup 0 2024-04-18 11:24 b  
-rw-r--r-- 1 hduser supergroup 0 2024-04-18 11:24 c  
drwxr-xr-x - hduser supergroup 0 2024-05-18 10:29 latin
- **\$ hadoop fs -copyFromLocal /home/hduser/student.tsv latin**
- **\$ hadoop fs -copyFromLocal /home/hduser/dept.tsv latin**
- **\$ hadoop fs -ls latin**
- **\$ pig**
  1. **grunt> A = load 'latin/student.tsv' as (rno:int,name:chararray,gpa:float);**  
(1001,John,3.0)  
(1002,Jack,4.0)  
(1003,Smith,4.5)  
(1004,Scott,4.2)  
(1005,Joshi,3.5)  
(1006,Alex,4.5)  
(1007,Dravid,4.2)  
(1008,James,4.0)

(1001,John,3.0)  
(1005,Joshi,3.0)

2. **grunt> B = DISTINCT A;**
3. **grunt> dump B;**  
(1001,John,3.0)  
(1002,Jack,4.0)  
(1003,Smith,4.5)  
(1004,Scott,4.2)  
(1005,Joshi,3.0)  
(1005,Joshi,3.5)  
(1006,Alex,4.5)  
(1007,Dravid,4.2)  
(1008,James,4.0)
4. **grunt> C = LIMIT A 3;**
5. **grunt> dump C;**  
(1001,John,3.0)  
(1002,Jack,4.0)  
(1003,Smith,4.5)
6. **grunt> D = ORDER A BY name;**
7. **grunt> dump D;**  
(1006,Alex,4.5)  
(1007,Dravid,4.2)  
(1002,Jack,4.0)  
(1008,James,4.0)  
(1001,John,3.0)  
(1001,John,3.0)  
(1005,Joshi,3.0)  
(1005,Joshi,3.5)  
(1004,Scott,4.2)  
(1003,Smith,4.5)
8. **grunt> E = FILTER A BY gpa>4.0;**
9. **grunt> dump E;**  
(1003,Smith,4.5)  
(1004,Scott,4.2)  
(1006,Alex,4.5)
10. **grunt> F = FOREACH A GENERATE UPPER(name);**
11. **grunt> dump F;**  
(JOHN)  
(JACK)  
(SMITH)  
(SCOTT)  
(JOSHI)  
(ALEX)  
(DRAVID)  
(JAMES)  
(JOHN)

```

12. grunt> Z = load 'latin/dept.tsv' as (rno:int,dno:int,dname:chararray);
13. grunt> dump Z;
    (1001,201,cse)
    (1002,201,cse)
    (1003,202,ece)
    (1004,202,ece)
    (1005,303,ele)
    (1006,303,ele)
    (1007,404,mec)
    (1008,404,mec)
14. grunt> J = JOIN A BY rno,Z BY rno;
15. grunt> dump J;
    (1001,John,3.0,1001,201,cse)
    (1001,John,3.0,1001,201,cse)
    (1002,Jack,4.0,1002,201,cse)
    (1003,Smith,4.5,1003,202,ece)
    (1004,Scott,4.2,1004,202,ece)
    (1005,Joshi,3.0,1005,303,ele)
    (1005,Joshi,3.5,1005,303,ele)
    (1006,Alex,4.5,1006,303,ele)
    (1007,Dravid,4.2,1007,404,mec)
    (1008,James,4.0,1008,404,mec)
16. grunt> U = UNION A,Z;
17. grunt> SPLIT A INTO X IF gpa==4.0, Y IF gpa<=4.0;
18. grunt> dump X;
    (1002,Jack,4.0)
    (1008,James,4.0)
19. grunt> dump Y;
    (1001,John,3.0)
    (1002,Jack,4.0)
    (1005,Joshi,3.5)
    (1008,James,4.0)
    (1001,John,3.0)
    (1005,Joshi,3.0)

```

- **\$ hadoop fs -copyFromLocal /home/hduser/lines.txt latin**
- **\$ hadoop fs -ls latin**

-rw-r--r--	1	hduser	supergroup	107	2024-05-18 10:31	latin/dept.tsv
-rw-r--r--	1	hduser	supergroup	98	2024-05-18 12:10	latin/lines.txt
-rw-r--r--	1	hduser	supergroup	148	2024-05-18 10:31	latin/student.tsv



- **\$ pig**
  1. **grunt> lines = LOAD 'latin/lines.txt' as (line:chararray);**
  2. **grunt> words=FOREACH            lines            GENERATE  
FLATTEN(TOKENIZE(line)) as word;**
  3. **grunt> grouped = GROUP words BY word;**
  4. **grunt> wordcount = FOREACH grouped GENERATE group,  
COUNT(words);**
  5. **grunt> DUMP wordcount;**
    - (to,2)
    - (yo,1)
    - (pig,1)
    - (Hive,2)
    - (haddop,1)
    - (hadoop,1)
    - (session,3)
    - (welcome,1)
    - (introduction,2)
    - (,0)

## Introduction to MongoDB

New kind of database is gaining ground in the enterprise called NoSQL (Not only SQL). We explore a NoSQL database called "MongoDB".

### WHAT IS MONGODB?

**MongoDB is**

1. Cross-platform.
2. Open source.
3. Non-relational.
4. Distributed.
5. NoSQL.
6. Document-oriented data store.

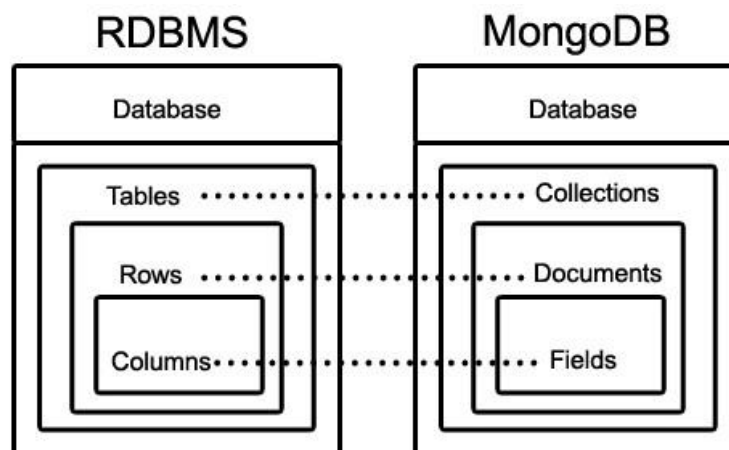
### WHY MONGODB?

- Deal with large volumes of data,
- Rich variety of data-particularly unstructured data
- Scaling of enterprise data - database that can scale out or scale horizontally
- Flexible schema
- Fault tolerant,
- Consistent and Partition tolerant (CAP theorem or Brewers theorem) - Easily distributed over a multitude of nodes in a cluster.

### Using Java Script Object Notation (JSON)

JSON is extremely expressive. MongoDB actually does not use JSON but BSON (pronounced Bee Son) - it is Binary JSON. It is an open standard, it is used to store complex data structures. JSON is very expressive. JSON provide easy to store and retrieve documents in their real form.

### TERMS USED IN RDBMS AND MONGODB



### Database

It is a collection of collections. In other words, it is like a container for collections. A single MongoDB server can house several databases.

### Collection

A collection is analogous to a table of RDBMS.

Collection is created the first time that you attempt to save a document that references it.

A collection exists within a single database. A **collection holds several MongoDB documents.**

**A collection does not enforce a schema.** The **documents within a collection can have different fields.** Even if the documents within a collection have same fields, the order of the fields can be different.

### Document

**A document is analogous to a row/record/tuple in an RDBMS table.**

**A document has a dynamic schema.**

**A document in a collection need not necessarily have the same set of fields/key-value pairs.**

### MongoDB Packages

MongoDB provides officially supported packages in their own repository:

Package Name	Description
<b>mongodb-org</b>	A meta package that will automatically install the four component packages listed below.
<b>mongodb-org-server</b>	Contains the mongod daemon, associated init script, and a configuration file (/etc/mongod.conf). You can use the initialization script to start mongod with the configuration file.
<b>mongodb-org-mongos</b>	Contains the mongos daemon.
<b>mongodb-org-shell</b>	Contains the mongo shell.
<b>mongodb-org-tools</b>	Contains the following MongoDB tools: mongoimport, bsondump, mongodump, mongoexport, mongofiles, mongorestore, mongostat, and mongotop.

### How to start MongoDB

- **\$ sudo service mongod start:-**This command is used to start the mongoDB.
- **\$ sudo service mongod stop:-** This command is used to stop the mongoDB.
- **\$ mongo:-**Before executing mongo db commands we should execute this command.

Once mongoDb is started we will get **MongoDB shell as >** as prompt

>

## CREATING DATA BASE:

The "use" command is used to create a database in MongoDB. If the database does not exist a new one will be created.

**Syntax :** use databasename

**e.g** To create a database by the name "myDB"

**>use myDB;** switched to db myDB.

## CHECKING YOUR DATA BASE:

To check your data base exists or not type the command at MongoDB shell "**db**".

run db in the shell to tell you the name of the active database

**>db;**

myDB

### Note:

**Note:** Newly created database, "myDB" does not show up in the list above. In order to list out the database needs to have at least one document in the list.

The default database in MongoDB is test. If one does not create any database, all collections are by default stored in the test database.

## COMMANDS:

### 1. Show the databases:

Syntax: **show dbs**

**>show dbs;** list out all the databases that already exist in the system.

### 2. Drop Databasesyntax: db.dropDatabase();

e.g - To drop the database, "myDB", first ensure that you are currently placed in "myDB" database and then use the db.dropDatabase() command to drop the database.

use myDB;

db.dropDatabase();

Confirm if the database "myDB" has been dropped.

**>db.dropDatabase();**

{ "dropped" : "myDB", "ok" : 1 }

**>**

**Note: If no database is selected, the default database "test" is dropped**

### 3. To display the current version of the MongoDB server:

**> db.version()**

5.0

### 4. For help, Type in db.help() in the MongoDB client to get a list of commands:

**> db.help()**

### 5. Show the collection list> show collections students food

### 6. To create a collection



To create a collection by the name "person". Let us check that collection i.e person exist or not

Syntax: db.createCollection ("person")

```
>show collections;
```

```
person students
```

```
food
```

```
>
```

## 7. To drop a collection

To drop a collection by the name "person".

First check whether collection person exist or not

```
>show collections;
```

```
person students
```

```
food
```

Syntax for drop collection: db.person.drop();

```
>db.person.drop()
```

```
true
```

```
>
```

## MONGODB QUERY LANGUAGE

CRUD (Create, Read Update, and Delete) operations in MongoDB

**Create** ==> Creation of data is done using insert() or update() or save() method.

**Read** ==> Reading the data is performed using the find() method.

**Update** ==> Update to data is accomplished using the update() method with UPSERT set to false.

**Delete** ==> a document is Deleted using the remove() method.

To Store the document(s) in to collection we had three functions

**1. insert()**

**2. update()**

**3. save()**

**1. insert()** - insert a fresh document in to collection

**2. update()** - insert a fresh document in to collection if document was not found with upsert:true option if document is found we can update the document fields

**3. save()** - insert a fresh document in to collection if document was not found with \_id if document \_id is found we can replace the document

**1. insert() - insert a fresh document in to collection student**

```
>db.student.insert({ "rollno":20201, "name":"Murali", "age":23,  
"contactno":9848929299,"email":"muralivadlamudi@gmail.com"})
```

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

Check whether document is stored in student collection

```
> db.student.find()
```

```
{ "_id" : ObjectId("5d65d54557c4c2d917f18bb3"), "rollno" : 20201, "name" : "Murali",
"age" : 23, "contactno": 9848929299, "e-mail" : "muralivadlamudi@gmail.com" }
```

### Check whether document is stored in student collection – Formatted way

```
> db.student.find().pretty()
{
  "_id" : ObjectId("5d65d54557c4c2d917f18bb3"),
  "rollno" : 20201,
  "name" : "Murali",
  "age" : 23,
  "contactno" : 9848929299,
  "e-mail" : "muralivadlamudi@gmail.com"
```

### Insert another document in to collection – student

```
>db.student.insert({"rollno":20201, "name":"Krishna", "age":23,
"contactno":9848111299,"email":"krishna@gmail.com"})
WriteResult({ "nInserted" : 1 })
```

```
> db.student.find()
{ "_id" : ObjectId("5d65d54557c4c2d917f18bb3"), "rollno" : 20201, "name" : "Murali",
"age" : 23, "contactno": 9848929299, "e-mail" : "muralivadlamudi@gmail.com" }
{ "_id" : ObjectId("5d65d89457c4c2d917f18bb4"), "rollno" : 20202, "name" : "Krishna",
"age" : 23,"contactno" : 9848111299, "e-mail" : "krishna@gmail.com" }
```

### 2. update() - insert a fresh document in to collection if document was not found with upsert 1:true option

if document is found we can update the document fields

Here we are searching for a document, if that is found we update it else we insert the document into collection

```
>db.student.update({"rollno":20203,"name":"Vamsi","age":23},
  {$set:{"contactno":9848811299,"e-mail":"vamsi@gmail.com"}},{upsert:true});
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("5d65de78386852e892928055")
})
```

**upsert** – is a boolean and Optional. If set to true, creates a new document when no document matches the query criteria. The **default value is false**, which does not insert a new document when no match is found

Now we try to update Vamsi phone number to 7988811299 as vamsi document already exists

```
>db.student.update({"rollno":17303,"name":"Vamsi","age":23},
  {$set:{"contactno":7988811299}},{upsert:true});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Now contact numner of Vamsi is updated.

**3. save()** - insert a fresh document in to collection if document was not found with \_id  
if document \_id is found we can replace the document

```
>db.student.save({"rollno":20204,"name":"Sudha","age":21,"contactno":6398193339,"e  
mail":"sudha@gmail.com"}) WriteResult({ "nInserted" : 1 })
```

```
> db.student.find().pretty()
```

```
{  
  "_id" : ObjectId("5d65dd2ade1660c49412a60d"),  
  "rollno" : 20201,  
  "name" : "Murali",  
  "age" : 23,  
  "contactno" : 9848912299,  
  "e-mail" : "muralivadlamudi@gmail.com"  
}  
{  
  "_id" : ObjectId("5d65dd58de1660c49412a60e"),  
  "rollno" : 20202,  
  "name" : "Krishna",  
  "age" : 23,  
  "contactno" : 9848191299,  
  "e-mail" : "krishna@gmail.com"  
}  
{  
  "_id" : ObjectId("5d65de78386852e892928055"),  
  "age" : 23,  
  "name" : "Vamsi",  
  "rollno" : 20203,  
  "contactno" : 7988811299,  
  "e-mail" : "vamsi@gmail.com"  
}  
{  
  "_id" : ObjectId("5d65e206de1660c49412a60f"),  
  "rollno" : 20204,  
  "name" : "Sudha",  
  "age" : 21,  
  "contactno" : 6398193339,  
  "e-mail" : "sudha@gmail.com"  
}  
>
```

To search for the documents - find() (Search for the document based on criteria)

e.g – Search for document in collection student with rollno- 20202

```
>db.student.find({rollno:20202});
```

```
{ "_id" : ObjectId("5d65dd58de1660c49412a60e"), "rollno" : 20202, "name" : "Krishna",  
  "age" : 23, "contactno" : 9848191299, "e-mail" : "krishna@gmail.com" }
```

**e.g – Search for document in collection student with name- Sudha**

> db.student.find({"name":"sudha"}).pretty() we did not get document as name is quoted as "sudha" instead of "Sudha"

```
> db.student.find({"name":"Sudha"}).pretty()
{
  "_id" : ObjectId("5d65e206de1660c49412a60f"),
  "rollno" : 20204,
  "name" : "Sudha",
  "age" : 21,
  "contactno" : 6398193339,
  "e-mail" : "sudha@gmail.com"
}
```

### Search for the document with out criteria projecting the specified fields

Now list out the specified fields in the document as per our choice selecting rollno,name,contactno only

Syntax : db.student.find({}, {rollno:1,name:1,contactno:1}).pretty()

Here we set rollno:1,name:1,contactno:1 1

indicate that field should display (true) 0

indicate that field should not display (false)

e.g

```
> db.student.find({}, {rollno:1,name:1,contactno:1})
{ "_id" : ObjectId("5d65dd2ade1660c49412a60d"), "rollno" : 20201, "name" : "Murali",
  "contactno" : 9848912299 }
{ "_id" : ObjectId("5d65dd58de1660c49412a60e"), "rollno" : 20202, "name" : "Krishna",
  "contactno" : 9848191299 }
{ "_id" : ObjectId("5d65de78386852e892928055"), "name" : "Vamsi", "rollno" : 20203,
  "contactno" : 7988811299 }
{ "_id" : ObjectId("5d65e206de1660c49412a60f"), "rollno" : 20204, "name" : "Sudha",
  "contactno" : 6398193339 }
```

### Display with out \_id number

```
> db.student.find({}, {rollno:1,name:1,contactno:1,_id:0}).pretty()
{ "rollno" : 17301, "name" : "Murali", "contactno" : 9848912299 }
{ "rollno" : 17302, "name" : "Krishna", "contactno" : 9848191299 }
{ "name" : "Vamsi", "rollno" : 17303, "contactno" : 7988811299 }
{ "rollno" : 17304, "name" : "Sudha", "contactno" : 6398193339 }
```

### Relational Operators

Name	Description
\$eq	Matches values that are equal to a specified value.
\$gt	Matches values that are greater than a specified value.
\$gte	Matches values that are greater than or equal to a specified value.
\$in	Matches any of the values specified in an array.
\$lt	Matches values that are less than a specified value.
\$lte	Matches values that are less than or equal to a specified value.

\$ne	Matches all values that are not equal to a specified value.
\$nin	Matches none of the values specified in an array.

### Adding a new field to an existing document – update method

Here's the MongoDB shell syntax for update():

**Syntax:** db.collection.update(criteria, objNew, upsert, multi )

**criteria** – Query which selects the record to update;

**objNew** – Updated object or \$ operators (e.g., \$inc) which manipulate the object **upsert** – if this should be an “upsert” operation; that is, if the record(s) do not exist, insert one.

\*Upsert only inserts a single document. **multi** – Indicates if all documents matching criteria should be updated rather than just one. Can be useful with the \$ operators below

```
> db.student.update({},{$set:{creditscore:"A"}},{multi:true});
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })
```

In the above example **we use empty criteria {}**, New field is **added to all documents**.

### Now change creditscore to person named Vamsi to “B”

```
> db.student.update({name:"Vamsi"},{$set:{creditscore:"B"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.student.find().pretty()
{
  "_id" : ObjectId("5d671076ae0af543597ceb16"),
  "rollno" : 20201,
  "name" : "Murali",
  "age" : 23,
  "contactno" : 9848912299,
  "e-mail" : "muralivadlamudi@gmail.com",
  "creditscore" : "A"
}
{
  "_id" : ObjectId("5d6710cf5ae0af543597ceb17"),
  "rollno" : 20202,
  "name" : "Krishna",
  "age" : 23,
  "contactno" : 9848191299,
  "e-mail" : "krishna@gmail.com",
  "creditscore" : "A"
}
{
  "_id" : ObjectId("5d67112bae0af543597ceb18"),
  "rollno" : 20203,
  "name" : "Vamsi",
  "age" : 22,
  "contactno" : 7988811299,
  "e-mail" : "vamsi@gmail.com",
```

```

"creditscore" : "B"
}
{
  "_id" : ObjectId("5d671175ae0af543597ceb19"),
  "rollno" : 20204,
  "name" : "Sudha",
  "age" : 21,
  "contactno" : 6398193339,
  "e-mail" : "sudha@gmail.com",
  "creditscore" : "A"
}
>

```

Other Options for update

1. collection.updateOne()
2. collection.updateMany()
3. collection.findOneAndUpdate()
4. collection.replaceOne()
5. collection.findOneAndReplace()
6. collection.findAndModify()

**>db.student.updateOne({ "age":{\$gte18} },{\$set: { "canVote": "True" },})**

This update statement updates the FIRST document that matches the given filter. Filter is any criteria based on which we want to update document.

**>db.student.updateMany({ "age":{\$gte 18} },{\$set: { "canVote": "True" },})** This command is similar to db.collection.updateOne() but the only difference is that it updates ALL the documents that match the filter in the collection. (i.e to update multiple documents).

**>db.student.remove({ "name" : "Sudha"})**

This method deletes a single document or all documents from the collection that match a specified filter.

**>db.student.deleteOne({contactno : 7988811299 } )**

This method deletes at most a single document that matches a specified filter even though multiple documents may match the specified filter.

**>db.student.deleteMany({age : 17})**

This method deletes all documents that match a specified filter. This command will delete all the documents from the 'student' collection where 'age' field is equal to '17'.

**Adding a New field "location" to the collection student**

```

>db.student.update({},{$set:{location:"Vijayawada"}},{multi:true});
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })

```

```

> db.student.find().pretty()
{
  "_id" : ObjectId("5d671076ae0af543597ceb16"),
  "rollno" : 20201,

```



```

"name" : "Murali",
"age" : 23,
"contactno" : 9848912299,
"e-mail" : "muralivadlamudi@gmail.com",
"creditscore" : "A",
"location" : "Vijayawada"
}
{
  "_id" : ObjectId("5d6710cfae0af543597ceb17"),
  "rollno" : 20202,
  "name" : "Krishna",
  "age" : 23,
  "contactno" : 9848191299,
  "e-mail" : "krishna@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada"
}
{
  "_id" : ObjectId("5d67112bae0af543597ceb18"),
  "rollno" : 20203,
  "name" : "Vamsi",
  "age" : 22,
  "contactno" : 7988811299,
  "e-mail" : "vamsi@gmail.com",
  "creditscore" : "B",
  "location" : "Vijayawada"
}
{
  "_id" : ObjectId("5d671175ae0af543597ceb19"),
  "rollno" : 20204,
  "name" : "Sudha",
  "age" : 21,
  "contactno" : 6398193339,
  "e-mail" : "sudha@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada"
}
>

```

**Here we insert two more documents into collection student**

```

>db.student.insert({"rollno": 20205,"name" : "Snigdha","age" : 21,"contactno" : 8383118383,
"e-mail" : "snigdha@gmail.com", "creditscore" : "A","location" : "Amaravati", });
WriteResult({ "nInserted" : 1 })

>db.student.insert({"rollno" : 20206,"name" : "Pravallika","age" : 20,"contactno" :
8982212383,"e-mail" : "pravallika@gmail.com","creditscore" : "B","location" : "Nellore"})
WriteResult({ "nInserted" : 1 })

```

```
>db.student.insert({"rollno":20207,"name":"Anamika","age":21,"contactno":9833194383,
"email" : "anamika@gmail.com","creditscore" : "A","location" : "Anantapuram"});
WriteResult({ "nInserted" : 1 })
```

**Here we inserted field as “hobbies” and set all as “sports”**

```
> db.student.update({},{$set:{hobbies:"sports"}},{multi:true});
WriteResult({ "nMatched" : 7, "nUpserted" : 0, "nModified" : 7 })
```

```
db.student.find().pretty()
{
  "_id" : ObjectId("5d671076ae0af543597ceb16"),
  "rollno" : 20201,
  "name" : "Murali",
  "age" : 23,
  "contactno" : 9848912299,
  "e-mail" : "muralivadlamudi@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d6710cf5ae0af543597ceb17"),
  "rollno" : 20202,
  "name" : "Krishna",
  "age" : 23,
  "contactno" : 9848191299,
  "e-mail" : "krishna@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d67112bae0af543597ceb18"),
  "rollno" : 20203,
  "name" : "Vamsi",
  "age" : 22,
  "contactno" : 7988811299,
  "e-mail" : "vamsi@gmail.com",
  "creditscore" : "B",
  "location" : "Vijayawada",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d671175ae0af543597ceb19"),
  "rollno" : 17304,
  "name" : "Sudha",
```

```

"age" : 21,
"contactno" : 6398193339,
"e-mail" : "sudha@gmail.com",
"creditscore" : "A",
"location" : "Vijayawada",
"hobbies" : "sports"
}
{
  "_id" : ObjectId("5d6729f4ae0af543597ceb1a"),
  "rollno" : 20205,
  "name" : "Snigdha",
  "age" : 21,
  "contactno" : 8383118383,
  "e-mail" : "snigdha@gmail.com",
  "creditscore" : "A",
  "location" : "Amaravati",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d672b1fae0af543597ceb1c"),
  "rollno" : 17306,
  "name" : "Pravallika",
  "age" : 20,
  "contactno" : 8982212383,
  "e-mail" : "pravallika@gmail.com",
  "creditscore" : "B",
  "location" : "Nellore",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d672ac8ae0af543597ceb1b"),
  "rollno" : 20207,
  "name" : "Anamika",
  "age" : 21,
  "contactno" : 9833194383,
  "e-mail" : "anamika@gmail.com",
  "creditscore" : "A",
  "location" : "Anantapuram",
  "hobbies" : "sports"
}

```

## TASKS

**Task1:** Find the document wherein the “Name” has value "Anamika" in students collection

```
>db.student.find({"name":  
"Anamika"}).pretty();
```

```
output: {  
  "_id" :  
  ObjectId("5d672ac8ae0af543597ceb1b"),  
  "rollno" : 20207,  
  "name" : "Anamika",  
  "age" : 21,  
  "contactno" : 9833194383,  
  "e-mail" : "anamika@gmail.com",  
  "creditscore" : "A",  
  "location" : "Anantapuram",  
  "hobbies" : "sports"  
}  
>
```

**Task2:** Find the document wherein the “creditscore” has value "A" in students collection

<pre>&gt;db.student.find({"creditscore": "A"}).pretty();</pre>	<pre>Output: {   "_id" :   ObjectId("5d671076ae0af543597ceb16"),   "rollno" : 20201,   "name" : "Murali",   "age" : 23,   "contactno" : 9848912299,   "e-mail" : "muralivadlamudi@gmail.com",   "creditscore" : "A",   "location" : "Vijayawada",   "hobbies" : "sports" } {   "_id" :   ObjectId("5d6710cfae0af543597ceb17"),   "rollno" : 20202,   "name" : "Krishna",   "age" : 23,   "contactno" : 9848191299,   "e-mail" : "krishna@gmail.com",   "creditscore" : "A",   "location" : "Vijayawada",   "hobbies" : "sports" } {   "_id" :   ObjectId("5d672ac8ae0af543597ceb1b"),   "rollno" : 20206,   "name" : "Anamika",   "age" : 21,   "contactno" : 9833194383,   "e-mail" : "anamika@gmail.com",</pre>
	<pre>"creditscore" : "A", "location" : "Anantapuram", "hobbies" : "sports" } &gt;</pre>
<p><b>Task3:</b> Find the document wherein the “creditscore” has value not equal to "A" in students collection</p>	

<pre>&gt;db.student.find({"creditscore": {\$ne:"A"}}).pretty();</pre>	<pre>Output: {   "_id" : ObjectId("5d67112bae0af543597ceb18"),   "rollno" : 20203,   "name" : "Vamsi",   "age" : 22,   "contactno" : 7988811299,   "e-mail" : "vamsi@gmail.com",   "creditscore" : "B",   "location" : "Vijayawada",   "hobbies" : "sports" } {   "_id" : ObjectId("5d672b1fae0af543597ceb1c"),   "rollno" : 20206,   "name" : "Pravallika",   "age" : 20,   "contactno" : 8982212383,   "e-mail" : "pravallika@gmail.com",   "creditscore" : "B",   "location" : "Nellore",   "hobbies" : "chess" }</pre>
---	--

### Regular Expression in MongoDB

<p><b>Task4:</b> Find the document wherein the “hobbies” are either "chess" or "acting" in students collection</p>	
<pre>&gt;db.student.find({"hobbies": {\$in: ["chess","acting"]}}).pretty(););</pre> <p>Here - and operation by {key: {\$in:[value,value]}}</p>	<pre>Output: {   "_id" : ObjectId("5d6729f4ae0af543597ceb1a"),   "rollno" : 20205,   "name" : "Snigdha",   "age" : 21,   "contactno" : 8383118383,   "e-mail" : "snigdha@gmail.com",   "creditscore" : "A",   "location" : "Amaravati",</pre>



	<pre> "hobbies" : "acting" } {   "_id" : ObjectId("5d672b1fae0af543597ceb1c"), "rollno" : 20206, "name" : "Pravallika", "age" : 20, "contactno" : 8982212383, "e-mail" : "pravallika@gmail.com", "creditscore" : "B", "location" : "Nellore", "hobbies" : "chess" } </pre>
--	--

**Task5: Find the document wherein the “hobbies” are neither "chess" nor "acting" in students collection**

<pre> &gt;db.student.find({"hobbies": {\$nin: ["chess","acting"]}}).pretty(); </pre> <p>Here - and operation by</p> <pre> { key: {\$nin:[value,value]}} </pre>	<p>Output:</p> <pre> {   "_id" : ObjectId("5d671076ae0af543597ceb16"),   "rollno" : 20201,   "name" : "Murali",   "age" : 23,   "contactno" : 9848912299,   "e-mail" : "muralivadlamudi@gmail.com",   "creditscore" : "A",   "location" : "Vijayawada",   "hobbies" : "sports" } {   "_id" : ObjectId("5d6710cfae0af543597ceb17"),   "rollno" : 20202,   "name" : "Krishna",   "age" : 23,   "contactno" : 9848191299,   "e-mail" : "krishna@gmail.com",   "creditscore" : "A",   "location" : "Vijayawada",   "hobbies" : "sports" } ... {   "_id" : ObjectId("5d672ac8ae0af543597ceb1b"),   "rollno" : 20207,   "name" : "Anamika",   "age" : 21,   "contactno" : 9833194383,   "e-mail" : "anamika@gmail.com",   "creditscore" : "A",   "location" : "Anantapuram",   "hobbies" : "sports" } &gt; </pre>
--	---

## AND condition

**Task6:** Find the document wherein the “hobbies” is sports and "location" Vijayawada in students collection

```
>db.student.find({"hobbies":"sports","location":  
"Vijayawada"}).pretty();
```

Here - and operation by separating simply by  
comma key:value,key:value

hobbies:"sports",location:"Vijayawada"

Output:

```
{  
  "_id" : ObjectId("5d671076ae0af543597ceb16"),  
  "rollno" : 20201,  
  "name" : "Murali",  
  "age" : 23,  
  "contactno" : 9848912299,  
  "e-mail" : "muralivadlamudi@gmail.com",  
  "creditscore" : "A",  
  "location" : "Vijayawada",  
  "hobbies" : "sports"  
}  
{  
  "_id" : ObjectId("5d6710cfae0af543597ceb17"),  
  "rollno" : 20202,  
  "name" : "Krishna",  
  "age" : 23,  
  "contactno" : 9848191299,  
  "e-mail" : "krishna@gmail.com",  
  "creditscore" : "A",  
  "location" : "Vijayawada",  
  "hobbies" : "sports"  
}  
...  
>
```

**Task7:** To find documents from the Students collection where the **name begins with "S"**

```
>db.student.find({"name":/^S/}).pretty();
```

/^S/ - indicate string beginning with S in  
regular expression

Note:

***Here “//” specifies that our search criteria is  
within these delimiters***

Output:

```
{  
  "_id" : ObjectId("5d671175ae0af543597ceb19"),  
  "rollno" : 20204,  
  "name" : "Sudha",  
  "age" : 21,  
  "contactno" : 6398193339,  
  "e-mail" : "sudha@gmail.com",  
  "creditscore" : "A",  
  "location" : "Vijayawada",  
  "hobbies" : "sports"  
}  
{  
  "_id" : ObjectId("5d6729f4ae0af543597ceb1a"),  
  "rollno" : 20205,  
  "name" : "Snigdha",  
  "age" : 21,  
  "contactno" : 8383118383,  
  "e-mail" : "snigdha@gmail.com",  
  "creditscore" : "A",  
  "location" : "Amaravati",  
  "hobbies" : "acting"  
}
```

	>
--	---

<b>Task8:</b> To find documents from the Students collection where the <b>name end with "i"</b>	
<pre>&gt;db.student.find({"name":/i\$/}).pretty();</pre> <p>/i\$/ - indicate string ending with i in regular expression</p>	<p>Output:</p> <pre>{   "_id" : ObjectId("5d671076ae0af543597ceb16"),   "rollno" : 20201,   "name" : "Murali",   "age" : 23,   "contactno" : 9848912299,   "e-mail" : "muralivadlamudi@gmail.com",   "creditscore" : "A",   "location" : "Vijayawada",   "hobbies" : "sports" } {   "_id" : ObjectId("5d67112bae0af543597ceb18"),   "rollno" : 20203,   "name" : "Vamsi",   "age" : 22,   "contactno" : 7988811299,   "e-mail" : "vamsi@gmail.com",   "creditscore" : "B",   "location" : "Vijayawada",   "hobbies" : "sports" } &gt;</pre>

<b>Task9:</b> To find documents from the Students collection where the name has an "v" in any position	
<pre>db.student.find({name:/v/}) .pretty (); OR db.student.find({name:/. *v.* /}) .prettyO; OR db.student.find({name:{\$regex:"v"}}).pretty();</pre>	<p>Output:</p> <pre>{   "_id" : ObjectId("5d672b1fae0af543597ceb1c"),   "rollno" : 20206,   "name" : "Pravallika",   "age" : 20,   "contactno" : 8982212383,   "e-mail" : "pravallika@gmail.com",   "creditscore" : "B",   "location" : "Nellore",   "hobbies" : "chess" }</pre>

### Using regex

<b>Task10:</b> To find documents from the Students collection where the <b>name begins with "S"</b> <b>Using regex</b>	
<pre>&gt;db.student.find({name: {\$regex:"^S"}}).pretty();</pre>	<p>Output:</p> <pre>{   "_id" : ObjectId("5d671175ae0af543597ceb19"),   "rollno" : 20204,   "name" : "Sudha",   "age" : 21,   "contactno" : 6398193339,   "e-mail" : "sudha@gmail.com",   "creditscore" : "A",   "location" : "Vijayawada",   "hobbies" : "sports" }</pre>
	<pre>{   "_id" : ObjectId("5d6729f4ae0af543597ceb1a"),   "rollno" : 20205,   "name" : "Snigdha",   "age" : 21,   "contactno" : 8383118383,   "e-mail" : "snigdha@gmail.com",   "creditscore" : "A",   "location" : "Amaravati",   "hobbies" : "acting" }</pre>
<b>Task11:</b> To find documents from the Students collection where the <b>name end with "i"</b> - <b>using regex</b>	

<pre>&gt;db.student.find({"name": {\$regex:"i\$"}}).pretty();</pre>	<pre>Output: {   "_id" : ObjectId("5d671076ae0af543597ceb16"),   "rollno" : 20201,   "name" : "Murali",   "age" : 23,   "contactno" : 9848912299,   "e-mail" : "muralivadlamudi@gmail.com",   "creditscore" : "A",   "location" : "Vijayawada",   "hobbies" : "sports" } {   "_id" : ObjectId("5d67112bae0af543597ceb18"),   "rollno" : 20203,   "name" : "Vamsi",   "age" : 22,   "contactno" : 7988811299,   "e-mail" : "vamsi@gmail.com",   "creditscore" : "B",   "location" : "Vijayawada",   "hobbies" : "sports" } &gt;</pre>
---	--

## Count, Limit, Sort, and Skip

**1. Count** - Count all document from the collection

**Task1: Count the number of the documents in the student collection.**

```
>db.student.count();
7
```

**Task2: Count the number of the documents in the student collection where location is Viajaywada.**

```
>db.student.count({location:"Vijayawada"});
4
```

**Task3: retrieve the first 2 documents in the student collection where location is Viajaywada.**

## 2.limit

**Syntax:** >db.COLLECTION\_name.find().limit(NUMBER)

```
>db.student.find({location:"Vijayawada"}).limit(2).pretty()
```

**output:**

```
{
  "_id" : ObjectId("5d671076ae0af543597ceb16"),
  "rollno" : 122mca09,
  "name" : "Murali",
  "age" : 23,
  "contactno" : 9848912299,
  "e-mail" : "muralivadlamudi@gmail.com",
  "creditscore" : "A",
```

```

    "location" : "Vijayawada",
    "hobbies" : "sports"
  }
  {
    "_id" : ObjectId("5d6710cfae0af543597ceb17"),
    "rollno" : 20202,
    "name" : "Krishna",
    "age" : 23,
    "contactno" : 9848191299,
    "e-mail" : "krishna@gmail.com",
    "creditscore" : "A",
    "location" : "Vijayawada",
    "hobbies" : "sports"
  }

```

**3. Skip** skip() which also accepts number type argument and is used to skip the number of documents.

**Syntax:** db.COLLECTION\_name.find().limit(NUMBER).skip(NUMBER)

**Note:** The default value in skip() method is 0 (zero)

**Task1: Skip first 4 documents from students collection**

```
>db.student.find().skip(4).pretty()
```

**out put:**

```

{
  "_id" : ObjectId("5d6729f4ae0af543597ceb1a"),
  "rollno" : 20205,
  "name" : "Snigdha",
  "age" : 21,
  "contactno" : 8383118383,
  "e-mail" : "snigdha@gmail.com",
  "creditscore" : "A",
  "location" : "Amaravati",
  "hobbies" : "acting"
}
{
  "_id" : ObjectId("5d672ac8ae0af543597ceb1b"),
  "rollno" : 20207,
  "name" : "Anamika",
  "age" : 21,
  "contactno" : 9833194383,
  "e-mail" : "anamika@gmail.com",
  "creditscore" : "A",
  "location" : "Anantapuram",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d672b1fae0af543597ceb1c"),
  "rollno" : 20206,
  "name" : "Pravallika",
  "age" : 20,

```



```

    "contactno" : 8982212383,
    "e-mail" : "pravallika@gmail.com",
    "creditscore" : "B",
    "location" : "Nellore",
    "hobbies" : "chess"
  }
>

```

## Task2: Skip first 4 documents from students collection limit to 2 documents

```
>db.student.find().skip(4).limit(2).pretty()
```

### out put:

```

{
  "_id" : ObjectId("5d6729f4ae0af543597ceb1a"),
  "rollno" : 17305,
  "name" : "Snigdha",
  "age" : 21,
  "contactno" : 8383118383,
  "e-mail" : "snigdha@gmail.com",
  "creditscore" : "A",
  "location" : "Amaravati",
  "hobbies" : "acting"
}
{
  "_id" : ObjectId("5d672ac8ae0af543597ceb1b"),
  "rollno" : 17307,
  "name" : "Anamika",
  "age" : 21,
  "contactno" : 9833194383,
  "e-mail" : "anamika@gmail.com",
  "creditscore" : "A",
  "location" : "Anantapuram",
  "hobbies" : "sports"
}
>

```

Here we got documents from 5 th onwards by skip(4) and listing only 2 documents as limit(2)

## Task3: To display the last 2 records from the student collection.

db.student.find().pretty().skip(db.student.count()-2) Here  
skip(db.student.count()-2) display last two documents

```

>db.student.find().pretty().skip(db.student.count()-2)
{
  "_id" : ObjectId("5d69db5b027d622d21d17ada"),
  "rollno" : 20206,
  "age" : 20,
  "contactno" : 8982212383,
  "e-mail" : "pravallika@gmail.com",
  "creditscore" : "B",
  "location" : "Nellore",
  "hobbies" : "chess",
  "name" : "Pravallika"
}
{
  "_id" : ObjectId("5d69dbe1027d622d21d17adb"),

```

```

    "rollno" : 20207,
    "age" : 21,
    "contactno" : 9833194383,
    "e-mail" : "anamika@gmail.com",
    "creditscore" : "A",
    "location" : "Anantapuram",
    "hobbies" : "sports",
    "name" : "Anamika"
  }
>

```

**Task 4:** To retrieve the **third, fourth, and fifth document** from the student collection.

```
db.student.find().pretty().skip(2).limit(3)
```

Here we are skipping first two documents with skip(2), then displaying next three documents by limit(3)

```

>db.student.find().pretty().skip(2).limit(3)
{
  "_id" : ObjectId("5d69d6e18cd7279cccb0f494"),
  "age" : 23,
  "name" : "Vamsi",
  "rollno" : 20203,
  "contactno" : 7988811299,
  "e-mail" : "vamsi@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d69d792027d622d21d17ad8"),
  "rollno" : 20204,
  "name" : "Sudha",
  "age" : 21,
  "contactno" : 6398193339,
  "e-mail" : "sudha@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d69dad7027d622d21d17ad9"),
  "rollno" : 20205,
  "age" : 21,
  "contactno" : 8383118383,
  "e-mail" : "snigdha@gmail.com",
  "creditscore" : "A",
  "location" : "Amaravati",
  "hobbies" : "acting",
  "name" : "Snigdha"
}
>

```

#### 4. Sort

**Task 1:** To sort the documents from the Student collection name in ascending order

```
db.student.find().sort({name:1}).pretty();
```

```

{
  "_id" : ObjectId("5d69dbe1027d622d21d17adb"),
  "rollno" : 20207,
  "name" : "Anamika",
  "age" : 21,
  "contactno" : 9833194383,
  "e-mail" : "anamika@gmail.com",
  "creditscore" : "A",
  "location" : "Anantapuram",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d69d6bf027d622d21d17ad7"),
  "rollno" : 20202,
  "name" : "Krishna",
  "age" : 23,
  "contactno" : 9848111299,
  "e-mail" : "krishna@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d69d63d4dfd101ef5eb9bb1"),
  "rollno" : 20201,
  "name" : "Murali",
  "age" : 23,
  "contactno" : 9848929299,
  "e-mail" : "muralivadlamudi@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports"
}
{
  "_id" : ObjectId("5d69db5b027d622d21d17ada"),
  "rollno" : 20206,
  "name" : "Pravallika",
  "age" : 20,
  "contactno" : 8982212383,
  "e-mail" : "pravallika@gmail.com",
  "creditscore" : "B",
  "location" : "Nellore",
  "hobbies" : "chess"
}
{
  "_id" : ObjectId("5d69dad7027d622d21d17ad9"),
  "rollno" : 20205,
  "name" : "Snigdha",
  "age" : 21,

```

```

    "contactno" : 8383118383,
    "e-mail" : "snigdha@gmail.com",
    "creditscore" : "A",
    "location" : "Amaravati",
    "hobbies" : "acting"
  }

```

**Task2: To sort the documents from the student collection first on name in ascending order and then on Hobbies in descending order.**

```
db.student.find().sort({name: 1, hobbies: -1}).pretty()
```

Here documents are sorted first on name in ascending order(1) then hobbies in descending order (-1)

```

>db.student.find().sort({name: 1, hobbies: -1}).pretty()
{
  "_id" : ObjectId("5d69dbe1027d622d21d17adb"),
  "rollno" : 20207,
  "age" : 21,
  "contactno" : 9833194383,
  "e-mail" : "anamika@gmail.com",
  "creditscore" : "A",
  "location" : "Anantapuram",
  "hobbies" : "sports",
  "name" : "Anamika"
}
{
  "_id" : ObjectId("5d69d6bf027d622d21d17ad7"),
  "rollno" : 20202,
  "age" : 23,
  "contactno" : 9848111299,
  "e-mail" : "krishna@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports",
  "name" : "Krishna"
}
{
  "_id" : ObjectId("5d69d63d4dfd101ef5eb9bb1"),
  "rollno" : 20201,
  "age" : 23,
  "contactno" : 9848929299,
  "e-mail" : "muralivadlamudi@gmail.com",
  "creditscore" : "A",
  "location" : "Vijayawada",
  "hobbies" : "sports",
  "name" : "Murali"
}
{
  "_id" : ObjectId("5d69db5b027d622d21d17ada"),
  "rollno" : 20206,
  "age" : 20,
  "contactno" : 8982212383,
  "e-mail" : "pravallika@gmail.com",
  "creditscore" : "B",

```

```

        "location" : "Nellore",
        "hobbies" : "chess",
        "name" : "Pravallika"
    }
    {
        "_id" : ObjectId("5d69dad7027d622d21d17ad9"),
        "rollno" : 20205,
        "age" : 21,
        "contactno" : 8383118383,
        "e-mail" : "snigdha@gmail.com",
        "creditscore" : "A",
        "location" : "Amaravati",
        "hobbies" : "acting",
        "name" : "Snigdha"
    }
    {
        "_id" : ObjectId("5d69d792027d622d21d17ad8"),
        "rollno" : 20204,
        "name" : "Sudha",
        "age" : 21,
        "contactno" : 6398193339,
        "e-mail" : "sudha@gmail.com",
        "creditscore" : "A",
        "location" : "Vijayawada",
        "hobbies" : "sports"
    }
    {
        "_id" : ObjectId("5d69d6e18cd7279cccb0f494"),
        "age" : 23,
        "name" : "Vamsi",
        "rollno" : 20203,
        "contactno" : 7988811299,
        "e-mail" : "vamsi@gmail.com",
        "creditscore" : "A",
        "location" : "Vijayawada",
        "hobbies" : "sports"
    }
}
>

```

**Task3:** To sort the documents from student collection by name and skip the first two documents from sorted list

**Syntax :** `db.student.find().skip(2).pretty().sort({name:1})`

```

> db.student.find().skip(2).pretty().sort({name:1})
{
    "_id" : ObjectId("5d69d63d4dfd101ef5eb9bb1"),
    "rollno" : 20201,
    "age" : 23,
    "contactno" : 9848929299,
    "e-mail" : "muralivadlamudi@gmail.com",
    "creditscore" : "A",
    "location" : "Vijayawada",
    "hobbies" : "sports",
    "name" : "Murali"
}
{

```

```

    "_id" : ObjectId("5d69db5b027d622d21d17ada"),
    "rollno" : 20206,
    "age" : 20,
    "contactno" : 8982212383,
    "e-mail" : "pravallika@gmail.com",
    "creditscore" : "B",
    "location" : "Nellore",
    "hobbies" : "chess",
    "name" : "Pravallika"
  }
  {
    "_id" : ObjectId("5d69dad7027d622d21d17ad9"),
    "rollno" : 20205,
    "age" : 21,
    "contactno" : 8383118383,
    "e-mail" : "snigdha@gmail.com",
    "creditscore" : "A",
    "location" : "Amaravati",
    "hobbies" : "acting",
    "name" : "Snigdha"
  }
  {
    "_id" : ObjectId("5d69d792027d622d21d17ad8"),
    "rollno" : 20204,
    "name" : "Sudha",
    "age" : 21,
    "contactno" : 6398193339,
    "e-mail" : "sudha@gmail.com",
    "creditscore" : "A",
    "location" : "Vijayawada",
    "hobbies" : "sports"
  }
  {
    "_id" : ObjectId("5d69d6e18cd7279cccb0f494"),
    "age" : 23,
    "name" : "Vamsi",
    "rollno" : 17303,
    "contactno" : 7988811299,
    "e-mail" : "vamsi@gmail.com",
    "creditscore" : "A",
    "location" : "Vijayawada",
    "hobbies" : "sports"
  }
}
>

```

### **Creation of bulk documents in to collection at one stretch**

#### **Step1: Create a variable (emp) at mongo prompt**

```

> var emp =
... [
...   {
...     "eno":1001,
...     "ename":"Murali",
...     "age":30,

```

```

...     "qual": "M.Sc(Maths)",
...     "role": "Lecturer",
...     "dept": "Mathematics",
...     "exp": 7,
...     "basic": 12000
... },
... {
...     "eno": 1002,
...     "ename": "Krishna",
...     "age": 33,
...     "qual": "M.Sc(Maths)",
...     "role": "Lecturer",
...     "dept": "Mathematics",
...     "exp": 5,
...     "basic": 10000
... },
... {
...     "eno": 1003,
...     "ename": "Vamsi",
...     "age": 33,
...     "qual": "M.Sc(Physics)",
...     "role": "Lecturer",
...     "dept": "Physics",
...     "exp": 6,
...     "basic": 11000
... },
... {
...     "eno": 1004,
...     "ename": "Sai Krishna",
...     "age": 33,
...     "qual": "M.Sc(Computers)",
...     "role": "Lecturer",
...     "dept": "Computers",
...     "exp": 8,
...     "basic": 12000
... },
... {
...     "eno": 1005,
...     "ename": "Snigdha",
...     "age": 33,
...     "qual": "MCA",
...     "role": "Lecturer",
...     "dept": "Computers",
...     "exp": 8,
...     "basic": 12000
... },
... {
...     "eno": 1006,
...     "ename": "Anamika",
...     "age": 32,
...     "qual": "MCA",
...     "role": "Lecturer",
...     "dept": "Computers",
...     "exp": 8,
...     "basic": 12000
... },

```



```
...];  
>
```

**Step2: Assign values in to a collection(employee)in current database**

```
> db.employee.insert(emp);  
BulkWriteResult({  
  "writeErrors" : [ ],  
  "writeConcernErrors" : [ ],  
  "nInserted" : 6,  
  "nUpserted" : 0,  
  "nMatched" : 0,  
  "nModified" : 0,  
  "nRemoved" : 0,  
  "upserted" : [ ]  
})
```

**Step3: display documents in collection**

```
> db.employee.find().pretty()  
{  
  "_id" : ObjectId("5d7166896167c454bdec28fd"),  
  "eno" : 1001,  
  "ename" : "Murali",  
  "age" : 30,  
  "qual" : "M.Sc(Maths)",  
  "role" : "Lecturer",  
  "dept" : "Mathematics",  
  "exp" : 7,  
  "basic" : 12000  
}  
{  
  "_id" : ObjectId("5d7166896167c454bdec28fe"),  
  "eno" : 1002,  
  "ename" : "Krishna",  
  "age" : 33,  
  "qual" : "M.Sc(Maths)",  
  "role" : "Lecturer",  
  "dept" : "Mathematics",  
  "exp" : 5,  
  "basic" : 10000  
}  
{  
  "_id" : ObjectId("5d7166896167c454bdec28ff"),  
  "eno" : 1003,  
  "ename" : "Vamsi",  
  "age" : 33,  
  "qual" : "M.Sc(Physics)",  
  "role" : "Lecturer",  
  "dept" : "Physics",  
  "exp" : 6,  
  "basic" : 11000  
}  
{  
  "_id" : ObjectId("5d7166896167c454bdec2900"),  
  "eno" : 1004,
```

```

    "ename" : "Sai Krishna",
    "age" : 33,
    "qual" : "M.Sc(Computers)",
    "role" : "Lecturer",
    "dept" : "Computers",
    "exp" : 8,
    "basic" : 12000
  }
  {
    "_id" : ObjectId("5d7166896167c454bdec2901"),
    "eno" : 1005,
    "ename" : "Snigdha",
    "age" : 33,
    "qual" : "MCA",
    "role" : "Lecturer",
    "dept" : "Computers",
    "exp" : 8,
    "basic" : 12000
  }
  {
    "_id" : ObjectId("5d7166896167c454bdec2902"),
    "eno" : 1006,
    "ename" : "Anamika",
    "age" : 32,
    "qual" : "MCA",
    "role" : "Lecturer",
    "dept" : "Computers",
    "exp" : 8,
    "basic" : 12000
  }
}
>

```

## Arrays

These MongoDB data types stores array. A set of values are represented as an array. This data type can store multiples of values and data types.

```

> db.food.insertMany([ { fruits: ["banana","apple","cherry"]}, { fruits:
["orange","butterfruit","mango"]}, { fruits: ["pineapple","strawberry","grapes"]}, {
fruits: ["banana","strawberry","grapes"]}, { fruits: ["orange","grapes"]} ]);
{
  "acknowledged" : true,
  "insertedIds" : [ObjectId("5d716e876167c454bdec2903"),
ObjectId("5d716e876167c454bdec2904"),
ObjectId("5d716e876167c454bdec2905"),
ObjectId("5d716e876167c454bdec2906"),
ObjectId("5d716e876167c454bdec2907")
]
}

```

**> db.food.find()**

```

{"_id" : ObjectId("5d716e876167c454bdec2903"),"fruits" : [ "banana","apple", "cherry" ] }
{"_id" : ObjectId("5d716e876167c454bdec2904"),"fruits": [ "orange","butter fruit","mango" ] }
{"_id":ObjectId("5d716e876167c454bdec2905"),"fruits":["pineapple","strawberry","grapes" ] }

```

```
{ "_id": ObjectId("5d716e876167c454bdec2906"), "fruits": [ "banana", "strawberry", "grapes" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2907"), "fruits": [ "orange", "grapes" ] }
```

**Task: Find out the document fruit array contain "banana", "apple", "cherry" in food collection**

```
> db.food.find({fruits:"banana","apple","cherry"}).pretty()
{
  "_id" : ObjectId("5d716e876167c454bdec2903"),
  "fruits" : [
    "banana",
    "apple",
    "cherry"
  ]
}
```

**Task: To list documents from the "food" collection which has the "fruits" array having "banana", as an element.**

```
> db.food.find({fruits:"banana"})
{ "_id" : ObjectId("5d716e876167c454bdec2903"), "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2906"), "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

**Task: To find documents from the "food" collection where "grapes" is present in the 1<sup>st</sup> index position of the "fruits" array** > db.food.find({"fruits.1":"grapes"}).pretty()

```
{
  "_id" : ObjectId("5d716e876167c454bdec2907"),
  "fruits" : [
    "orange",
    "grapes"
  ]
}
>
```

**Task: To find documents from the "food" collection where "grapes" is present in the 2<sup>nd</sup> index position of the "fruits" array** > db.food.find({"fruits.2":"grapes"})

```
{ "_id" : ObjectId("5d716e876167c454bdec2905"), "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2906"), "fruits" : [ "banana", "strawbeery", "grapes" ] }
>
```

**Task: To find and list 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}})

```
{ "_id" : ObjectId("5d716e876167c454bdec2907"), "fruits" : [ "orange", "grapes" ] }
>
```

**Task: To find those documents from the "food" collection where the size of the array is three. Array with 3 values.** > db.food.find({fruits:{\$size:3}})

```
{ "_id" : ObjectId("5d716e876167c454bdec2903"), "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2904"), "fruits" : [ "orange", "butterfruit", "mango"
] }
{ "_id" : ObjectId("5d716e876167c454bdec2905"), "fruits" : [ "pineapple", "strawberry",
"grapes" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2906"), "fruits" : [ "banana", "strawbeery",
"grapes" ] }
>
```

**Task: To find the documents from array, whose size is 3 from the "food" collection and display the first two elements > db.food.find({fruits:{\$size:3}},{fruits:{\$slice:2}})**

```
{ "_id" : ObjectId("5d716e876167c454bdec2903"), "fruits" : [ "banana", "apple" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2904"), "fruits" : [ "orange", "butterfruit" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2905"), "fruits" : [ "pineapple", "strawberry" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2906"), "fruits" : [ "banana", "strawbeery" ] }
>
```

**Task: To find those documents from the "food" collection which have the element "orange" in the 0<sup>th</sup> index position in the array "fruits". Display two elements, starting with the element at 1<sup>st</sup> index position.**

```
> db.food.find({"fruits.0":"orange"},{fruits:{$slice:[1,2]}})
{ "_id" : ObjectId("5d716e876167c454bdec2904"), "fruits" : [ "butterfruit", "mango" ] }
{ "_id" : ObjectId("5d716e876167c454bdec2907"), "fruits" : [ "grapes" ] }
>
```

## Creating Bulk documents

```
> db.customer.insert([
...   {custId:"C101",name:"Murali","accType":"S",balance:10200 },
...   {custId:"C102",name:"Krishna","accType":"S",balance:15204 },
...   {custId:"C103",name:"Soumya","accType":"S",balance:13092 },
...   {custId:"C104",name:"Sailaja","accType":"S",balance:110400 },
...   {custId:"C105",name:"Anamika Textiles","accType":"C",balance:1022200 },
... ]);
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 5,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})
>
```

## **Aggregate Function**

**i. \$project :** This stage is used to **select certain fields from a collection**. We can also add, remove or reshape a key.

**Syntax:** { \$project: { <specifications> } }

### **Specification Description**

**<field>: <1 or true>** Specify the inclusion of a field.

**\_id: <0 or false>** Specify the suppression of the \_id field.

**<field>: <expression>** Add a new field or reset the value of an existing field.

**List selected fields in documents from customer collection, By default ObjectId is displayed >** **db.customer.aggregate( [ { \$project : { custId:1,name:1 } } ] );**

```
{ "_id" : ObjectId("5d745da59233b0160ec69416"), "custId" : "C101", "name" : "Murali" }
{ "_id" : ObjectId("5d745da59233b0160ec69417"), "custId" : "C102", "name" : "Krishna" }
{ "_id" : ObjectId("5d745da59233b0160ec69418"), "custId" : "C103", "name" : "Soumya" }
{ "_id" : ObjectId("5d745da59233b0160ec69419"), "custId" : "C104", "name" : "Sailaja" }
{ "_id" : ObjectId("5d745da59233b0160ec6941a"), "custId" : "C105", "name" : "Anamika Textiles" }
```

**ObjectId can be suppressed by \_id:0 >** **db.customer.aggregate( [ { \$project : { \_id:0,custId:1,name:1 } } ] );**

```
{ "custId" : "C101", "name" : "Murali" }
{ "custId" : "C102", "name" : "Krishna" }
{ "custId" : "C103", "name" : "Soumya" }
{ "custId" : "C104", "name" : "Sailaja" }
{ "custId" : "C105", "name" : "Anamika Textiles" }
```

**> ii. \$match** The MongoDB \$match operator filters the documents to pass only those documents that match the specified condition(s) to the next pipeline stage.

**> db.customer.aggregate([{\$match:{accType:"S"}}]);**

```
{ "_id" : ObjectId("5d745da59233b0160ec69416"), "custId" : "C101", "name" : "Murali",
"accType" : "S",
"balance" : 10200 }
{ "_id" : ObjectId("5d745da59233b0160ec69417"), "custId" : "C102", "name" : "Krishna",
"accType" : "S",
"balance" : 15204 }
{ "_id" : ObjectId("5d745da59233b0160ec69418"), "custId" : "C103", "name" : "Soumya",
"accType" : "S",
"balance" : 13092 }
{ "_id" : ObjectId("5d745da59233b0160ec69419"), "custId" : "C104", "name" : "Sailaja",
"accType" : "S",
"balance" : 110400 }
>
```

**iii. \$group** It does the work as the name says. It groups all documents based on some keys.

**Task: No we try to count the customers whose account Type is “S”**

```
>db.customer.aggregate([{$match:{balance:{$gt:15000}}},{ $group:{_id:null,count:
{$sum:1}}});
{ "_id" : null, "count" : 3 }
```

```
> db.customer.aggregate([{$group:{_id:null,count:{$sum:1}}}] );
{ "_id" : null, "count" : 5 }
> db.customer.aggregate([{$match:{accType:"S"}},{$group:{_id:null,count:{$sum:1}}}] );
{ "_id" : null, "count" : 4 }
```

#### **Task-1 Average balance**

```
> db.customer.aggregate([{$group:{_id:"$accType",TotBalance:{$avg:"$balance"}}}] );
{ "_id" : "C", "TotBalance" : 1022200 }
{ "_id" : "S", "TotBalance" : 37224 }
```

#### **Task-2 Maximum balance**

```
> db.customer.aggregate([{$group:{_id:"$accType",MaxBalance:{$max:"$balance"}}}] );
{ "_id" : "C", "MaxBalance" : 1022200 }
{ "_id" : "S", "MaxBalance" : 110400 }
```

#### **Task-3 Minimum balance**

```
> db.customer.aggregate([{$group:{_id:"$accType",MinBalance:{$min:"$balance"}}}] );
{ "_id" : "C", "MinBalance" : 1022200 }
{ "_id" : "S", "MinBalance" : 10200 }
>
```

#### **Task-4 First document- Customer balance by account type**

```
> db.customer.aggregate([{$group:{_id:"$accType",Balance:{$first:"$balance"}}}] );
{ "_id" : "C", "Balance" : 1022200 }
{ "_id" : "S", "Balance" : 10200 }
```

#### **Task-5 Last document – Customer balance by account type**

```
> db.customer.aggregate([{$group:{_id:"$accType",Balance:{$last:"$balance"}}}] );
{ "_id" : "C", "Balance" : 1022200 }
{ "_id" : "S", "Balance" : 110400 }
>
```

**iv. \$sort** It is used to sort all the documents.

#### **Task-1**

```
>db.customer.aggregate([{$group:{_id:"$accType",TotBalance:{$sum:"$balance"}},
{$sort:{_id:1}}]);
{ "_id" : "C", "TotBalance" : 1022200 }
{ "_id" : "S", "TotBalance" : 148896 }
```

#### **Task-2**

```
>db.customer.aggregate([{$group:{_id:"$accType",TotBalance:{$sum:"$balance"}},
{$sort:{_id:-1}}]);
{ "_id" : "S", "TotBalance" : 148896 }
{ "_id" : "C", "TotBalance" : 1022200 }
>
```

## **Map Reduce function**

Collection :author has 6 documents

```
db.author.find().pretty()
{
  "_id" : ObjectId("5d75134647bf7030afa9a26b"),
  "book_title" : "MongoDB Tutorial",
  "author_name" : "aparajita",
  "status" : "active",
  "publish_year" : "2016"
}
{
  "_id" : ObjectId("5d75139f47bf7030afa9a26c"),
  "book_title" : "Software Testing",
  "author_name" : "aparajita",
  "status" : "active",
  "publish_year" : "2015"
}
{
  "_id" : ObjectId("5d7513d247bf7030afa9a26d"),
  "book_title" : "Node.Js Tutorial",
  "author_name" : "keethika",
  "status" : "active",
  "publish_year" : "2016"
}
{
  "_id" : ObjectId("5d75141f47bf7030afa9a26e"),
  "book_title" : "PHP7 Tutorial",
  "author_name" : "aparajita",
  "status" : "passive",
  "publish_year" : "2016"
}
{
  "_id" : ObjectId("5d75144647bf7030afa9a26f"),
  "book_title" : "Parallel Computing",
  "author_name" : "rajaraman",
  "status" : "active",
  "publish_year" : "2014"
}
{
  "_id" : ObjectId("5d75147b47bf7030afa9a270"),
  "book_title" : "Fundamentals of Computers",
  "author_name" : "rajaraman",
  "status" : "passive",
  "publish_year" : "2010"
}
```

---

**> Case1 : Code with find () with active authors**

```
> db.author.mapReduce(
...   function() { emit(this.author_name,1)},
...   function(key, values) {return Array.sum(values)},
...   {query:{status:"active"},
...   out:"author_total"});
{ "result" : "author_total", "ok" : 1 }
```

>

output

:

```
>db.author_total.find();
{ "_id" : "aparajita", "value" : 2 }
{ "_id" : "rajaraman", "value" : 1 }
{ "_id" : "keethika", "value" : 1 }
```

>

>

**Case2 : Code with find () with passive authors**

```
> db.author.mapReduce(
  function() { emit(this.author_name,1)},
function(key, values) {return Array.sum(values)},
  {query:{status:"passive"},
out:"author_total"});
{ "result" : "author_total", "ok" : 1 }
```

```
> db.author_total.find();
{ "_id" : "rajaraman", "value" : 1 }
{ "_id" : "aparajita", "value" : 1 }
```

>

**Mongo db import and export Mongo Import**

MongoImport used at the command prompt imports CSV (Comma Separated Values) or tsv (Tab Separated Values) files or JSON (Java Script Object Notation) documents into MongoDB.

(in Ubuntu - \$ prompt / Windows - command prompt)



## Import CSV File

You can import a CSV file by using `--type csv`.

If the CSV file has a header row, use `--headerline` to tell mongoimport to use the first line to determine the name of the fields in the resulting document.

If the CSV file doesn't have a header row, use the `--fields` parameter to set the field names.

### Case:1 -With Header Row

Here's an example of importing a document with a header row.

The contents of the CSV file:

```
_id,albumname,artistname
1,Killers,"Iron Maiden"
2,Powerslave,"Iron Maiden"
12,"Somewhere in Time","Iron Maiden"
3,"Surfing with the Alien","Joe Satriani"
10,"Flying in a Blue Dream","Joe Satriani"
11,"Black Swans and Wormhole Wizards","Joe Satriani"
6,"Out of the Loop","Mr Percival"
7,"Suck on This",Primus
8,"Pork Soda",Primus
9,"Sailing the Seas of Cheese",Primus
```

### Import the file:

```
>mongoimport --db music --collection catalog --type csv --headerline --file
/data/dump/music/catalog.csv
```

Query the collection:

```
> db.catalog.find()
{ "_id" : 2, "albumname" : "Powerslave", "artistname" : "Iron Maiden" }
{ "_id" : 1, "albumname" : "Killers", "artistname" : "Iron Maiden" }
{ "_id" : 3, "albumname" : "Surfing with the Alien", "artistname" : "Joe Satriani" }
{ "_id" : 12, "albumname" : "Somewhere in Time", "artistname" : "Iron Maiden" }
{ "_id" : 10, "albumname" : "Flying in a Blue Dream", "artistname" : "Joe Satriani" }
{ "_id" : 6, "albumname" : "Out of the Loop", "artistname" : "Mr Percival" }
{ "_id" : 7, "albumname" : "Suck on This", "artistname" : "Primus" }
{ "_id" : 8, "albumname" : "Pork Soda", "artistname" : "Primus" }
{ "_id" : 11, "albumname" : "Black Swans and Wormhole Wizards", "artistname" : "Joe Satriani" }
{ "_id" : 9, "albumname" : "Sailing the Seas of Cheese", "artistname" : "Primus" }
```

### case:2 - Without Header Row

Here's another CSV file, but this one doesn't have a header row:

```
Mutt Lange, 1948
John Petrucci, 1967
DJ Shadow, 1972
George Clinton, 1941
```

Now we'll import it and specify the field names to use:

```
> mongoimport --db music --collection producers --type csv --fields name,born --file /data/dump/music/producers.csv
```

Here we are specifying the fields name,born to the collection during import

Query the collection:

```
> db.producers.find()
```

```
{ "_id" : 1, "name" : "Bob Rock" }
{ "_id" : ObjectId("5784a3a5dfad478c015f6b72"), "name" : "John Petrucci", "born" : 1967 } {
  "_id" : ObjectId("5784a3a5dfad478c015f6b73"), "name" : "Mutt Lange", "born" : 1948 } {
  "_id" : ObjectId("5784a3a5dfad478c015f6b74"), "name" : "George Clinton", "born" :
  1941 }
```

```
{ "_id" : ObjectId("5784a3a5dfad478c015f6b75"), "name" : "DJ Shadow", "born" : 1972 }
```

You'll see that the ObjectId field has been automatically created and populated for us.

Also, we already had one document in this collection before we ran the import: { "\_id" : 1, "name" : "Bob

Rock" }. Therefore, you can see that the import has simply added to the collection (as opposed to replacing it and all its contents).

## MongoExport

This command used at the command prompt exports MongoDB JSON documents into CSV (Comma Separated Values) or TSV (Tab Separated Values) files or JSON (Java Script Object Notation) documents. In MongoDB, you can export data using the mongoexport utility.

You can use the mongoexport utility to export data from your MongoDB database, to a JSON or CSV file. The utility is located in the MongoDB bin directory (eg, /mongodb/bin). When you run the utility, provide the name of the database, the collection, and the file you want it to be exported to. To export data, first open a new Terminal/Command Prompt window, then type the applicable command.

### Export a Collection to a JSON File

Here, we use mongoexport to export the **artists** collection to a **JSON** file:

```
> mongoexport --db music --collection artists --out /data/dump/music/artists.json
```

In the above command artists is the collection in mongodb database

/data/dump/music/artists.json – is the artists.json is file name and rest is the path where it is to be stored

### Resulting message:

2022-01-12T09:57:37.613+0700 connected to: localhost 2022-

01-12T09:57:37.614+0700 exported 13 records **Resulting file:**

```
{"_id":{"$_oid":"5780fbf948ef8c6b3ffb0149"},"artistname":"The Tea Party"}
{"_id":{"$_oid":"5781c9ac48ef8c6b3ffb014a"},"artistname":"Jorn Lande"}
{"_id":1.0,"artistname":"AC/DC"}
{"_id":{"$_oid":"5781d7f248ef8c6b3ffb014d"},"artistname":"The Kooks"}
{"_id":{"$_oid":"5781d7f248ef8c6b3ffb014e"},"artistname":"Bastille"}
{"_id":{"$_oid":"5781d7f248ef8c6b3ffb014f"},"artistname":"Gang of Four"}
{"_id":{"$_oid":"5781f85d48ef8c6b3ffb0150"},"artistname":"DeepPurple","albums":
[{"album":"MachineHead","year":1972.0,"genre":"Rock"},
{"album":"Stormbringer","year":1974.0,"genre":"Rock"}]}
{"_id":{"$_oid":"578214f048ef8c6b3ffb0159"},"artistname":"MilesDavis","albums":
[{"album":"KindofBlue","year":1959.0,"genre":"Jazz"},
```

```
{ "album": "Bitches Brew", "year": 1970.0, "genre": "Jazz" } } }
{ "_id": { "$oid": "578217c248ef8c6b3ffb015a" }, "artistname": "Robben
Ford", "albums": [ { "album": "Bringing it
Back Home", "year": 2013.0, "genre": "Blues" }, { "album": "Talk to Your
Daughter", "year": 1988.0, "genre": "Blues" } ] } }
{ "_id": { "$oid": "578217c248ef8c6b3ffb015b" }, "artistname": "Snoop
Dogg", "albums": [ { "album": "Tha
Doggfather", "year": 1996.0, "genre": "Rap" },
{ "album": "Reincarnated", "year": 2013.0, "genre": "Reggae" } ] } }
{ "_id": 2.0, "artistname": "Prince", "address": { "street": "Audubon
Road", "city": "Chanhassen", "state": "Minnesota", "country": "United States" } }
{ "_id": 3.0, "artistname": "Moby", "albums":
[ { "album": "Play", "year": 1999.0, "genre": "Electronica" },
{ "album": "Long Ambients 1: Calm. Sleep.", "year": 2016.0, "genre": "Ambient" } ] }
{ "_id": 4.0, "artistname": "Rush" }
```

If you find that you can't run mongoexport, be sure that you've either exited the mongo utility, or opened a new Terminal/Command Prompt window before running mongoexport, as it is a separate utility.

The above command assumes that the MongoDB bin directory is in your PATH. If it's not, you will need to use the full path to the mongoexport file.

**For example, /mongodb/bin/mongoexport or wherever your MongoDB deployment is installed.**

If you don't provide a file path for the exported file, it will be created wherever you are located when you run the command. Either provide the full path, or navigate to where you'd like the data file to be written before you run the command.

### Export a Collection to a CSV File

To export to a CSV file, add --type=csv to the command.

You must also specify the fields in the MongoDB documents to export.

Here, we use mongoexport to export the artists collection to a CSV file. We export the \_id and artistname fields. We've also given the file name a .csv extension.

```
>mongoexport --db music --collection artists --type=csv --fields _id,artistname --out
/data/dump/music/artists.csv
```

In the above command artists is the collection in mongodb database/data/dump/music/artists.csv – is the artists.csv is file name and rest is the path where it is to be stored

### Resulting message:

```
2022-01-12T10:16:33.111+0700 connected to: localhost 2022-
01-12T10:16:33.114+0700 exported 13 records Resulting
```

### CSV file:

```
_id,artistname
ObjectId(5780fbf948ef8c6b3ffb0149),The Tea Party
ObjectId(5781c9ac48ef8c6b3ffb014a),Jorn Lande
1,AC/DC
ObjectId(5781d7f248ef8c6b3ffb014d),The Kooks
ObjectId(5781d7f248ef8c6b3ffb014e),Bastille
```

ObjectId(5781d7f248ef8c6b3ffb014f),Gang of Four  
ObjectId(5781f85d48ef8c6b3ffb0150),Deep Purple  
ObjectId(578214f048ef8c6b3ffb0159),Miles Davis  
ObjectId(578217c248ef8c6b3ffb015a),Robben Ford  
ObjectId(578217c248ef8c6b3ffb015b),Snoop Dogg  
2,Prince  
3,Moby  
4,Rush

### Export the results of a Query

You can use the --query option to specify a query to export. The query must be enclosed in single quotes.

Here, we export details on Miles Davis to a JSON file:

```
> mongoexport --db music --collection artists --query '{"artistname": "Miles Davis"}' -  
out/data/dump/music/miles_davis.json
```

**Resulting message:**

2022-01-12T10:32:19.794+0700 connected to: localhost

2022-01-12T10:32:19.795+0700 exported 1 record

### Resulting JSON file:

```
{"_id":{"$_id":"578214f048ef8c6b3ffb0159"},"artistname":"Miles Davis","albums":  
[{"album":"Kind of  
Blue","year":1959.0,"genre":"Jazz"}, {"album":"Bitches  
Brew","year":1970.0,"genre":"Jazz"}]}
```

### Other Options

The mongoexport utility provides a number of options. Here are some potentially useful ones.

#### The --limit Option

Limits the number of documents in the export. mongoexport --db music --collection artists -  
limit 3 --out /data/dump/music/3\_artists.json

**Resulting file:**

```
{"_id":{"$_id":"5780fbf948ef8c6b3ffb0149"},"artistname":"The Tea Party"}  
{"_id":{"$_id":"5781c9ac48ef8c6b3ffb014a"},"artistname":"Jorn Lande"}  
{"_id":1.0,"artistname":"AC/DC"}
```

#### The --sort Option

Specifies how the results are ordered.

Here, we sort the file by the \_id field in ascending order (i.e. 1). To make it descending, use a -1.

```
mongoexport --db music --collection artists --limit 3 --sort '{_id: 1}' --out  
/data/dump/music/3_artists_sorted.json
```

**Resulting file:**

```
{"_id":1.0,"artistname":"AC/DC"}  
{"_id":2.0,"artistname":"Prince","address":{"street":"Audubon  
Road","city":"Chanhassen","state":"Minnesota","country":"United States"}}  
{"_id":3.0,"artistname":"Moby","albums":  
[{"album":"Play","year":1999.0,"genre":"Electronica"},  
{"album":"Long Ambients 1: Calm. Sleep.", "year":2016.0,"genre":"Ambient"}]}
```

#### The --skip Option

Allows you to instruct mongoexport to skip a number of documents before starting the export operation.

**mongoexport --db music --collection artists --limit 3 --sort '{\_id: 1}' --skip 2 --out /data/dump/music/3\_artists\_sorted\_skipped.json** Resulting file:

```
{"_id":3.0,"artistname":"Moby","albums":
[{"album":"Play","year":1999.0,"genre":"Electronica"},
{"album":"Long Ambients 1: Calm. Sleep.","year":2016.0,"genre":"Ambient"}]}
{"_id":4.0,"artistname":"Rush"}
{"_id":{"$_oid":"5780fbf948ef8c6b3ffb0149"},"artistname":"The Tea Party"}
```

### **The --pretty Option**

Outputs documents in a more readable JSON format. **mongoexport --db music --collection artists --query '{"artistname": "Miles Davis"}' --pretty --out/data/dump/music/miles\_davis\_pretty.json** Resulting file:

```
{
  "_id": {
    "$oid": "578214f048ef8c6b3ffb0159"
  },
  "artistname": "Miles Davis",
  "albums": [
    {
      "album": "Kind of Blue",
      "year": 1959.0,
      "genre": "Jazz"
    },
    {
      "album": "Bitches Brew",
      "year": 1970.0,
      "genre": "Jazz"
    }
  ]
}
```

## Spark

Apache Spark is a lightning fast real-time processing framework. It does in-memory computations to analyze data in real-time. It came into limelight as **Apache Hadoop MapReduce** was performing batch processing only and lacked a real-time processing feature. Hence, Apache Spark was introduced as it can perform stream processing in real-time and can also take care of batch processing.

Apart from real-time and batch processing, Apache Spark supports interactive queries and iterative algorithms also. Apache Spark has its own cluster manager, where it can host its application. It leverages Apache Hadoop for both storage and processing. It uses **HDFS** (Hadoop Distributed File system) for storage and it can run Spark applications on **YARN** as well.

## PySpark

PySpark is a Spark library written in Python to run Python applications using Apache Spark capabilities, using PySpark we can run applications parallelly on the distributed cluster (multiple nodes). PySpark is a Python API for Apache Spark. Apache Spark is an analytical processing engine for large scale powerful distributed data processing and machine learning applications.



Spark basically written in Scala and later on due to its industry adaptation it's API PySpark released for Python using Py4J. Py4J is a Java library that is integrated within PySpark and allows python to dynamically interface with JVM objects, hence to run PySpark you also need Java to be installed along with Python, and Apache Spark.

For the development, you can use Anaconda distribution (used in the Machine Learning community) which comes with a lot of useful tools like Spyder IDE, Jupyter notebook to run PySpark applications.

In real-time, PySpark has used a lot in the machine learning & Data scientists community; thanks to vast python machine learning libraries. Spark runs operations on billions and trillions of data on distributed clusters 100 times faster than the traditional python applications.

### USES:

PySpark is very well used in Data Science and Machine Learning community as there are many widely used data science libraries written in Python including NumPy, TensorFlow. Also used due to its efficient processing of large datasets. PySpark has been used by many organizations like Walmart, Trivago, Sanofi, Runtastic, and many more.

### Features of PySpark.

In-memory computation

- Distributed processing using parallelize
- Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c)
- Fault-tolerant

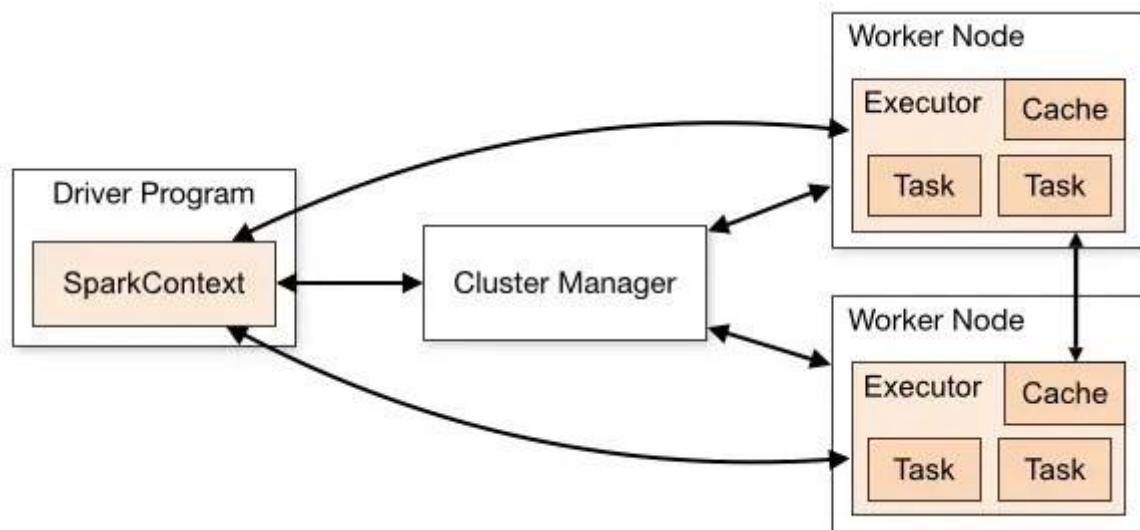
- Immutable
- Lazy evaluation
- Cache & persistence
- Inbuild-optimization when using DataFrames
- Supports ANSI SQL

### Advantages of PySpark

- PySpark is a general-purpose, in-memory, distributed processing engine that allows you to process data efficiently in a distributed fashion.
- Applications running on PySpark are 100x faster than traditional systems.
- You will get great benefits using PySpark for data ingestion pipelines.
- Using PySpark we can process data from Hadoop HDFS, AWS S3, and many file systems.
- PySpark also is used to process real-time data using Streaming and Kafka.
- Using PySpark streaming you can also stream files from the file system and also stream from the socket.
- PySpark natively has machine learning and graph libraries.

### PySpark Architecture

Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers”. When you run a Spark application, Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager.



### Cluster Manager Types

Spark supports below cluster managers:

- Standalone – a simple cluster manager included with Spark that makes it easy to set up a cluster.
- Apache Mesos – Mesos is a Cluster manager that can also run Hadoop MapReduce and PySpark applications.

- Hadoop YARN– the resource manager in Hadoop 2. This is mostly used, cluster manager.
- Kubernetes– an open-source system for automating deployment, scaling, and management of containerized applications.

local – which is not really a cluster manager but we use “local” for master() in order to run Spark on your laptop/Desktop computer.

### **PySpark Modules & Packages**

- PySpark RDD (pyspark.RDD)
- PySpark DataFrame and SQL (pyspark.sql)
- PySpark Streaming (pyspark.streaming)
- PySpark MLlib (pyspark.ml, pyspark.mllib)
- PySpark GraphFrames (GraphFrames)
- PySpark Resource (pyspark.resource) It's new in PySpark 3.0

### **Installation Steps:**

1. Login to linux computer with hduser as login
2. Update the system  
\$ sudo apt-get update
3. Upgrade the system  
\$ sudo apt-get upgrade
4. Check whether Java JDK is installed or not

```
$ java -version openjdk version
"17.0.5" 2022-10-18
OpenJDK Runtime Environment (build 17.0.5+8-Ubuntu-2ubuntu122.04)
OpenJDK 64-Bit Server VM (build 17.0.5+8-Ubuntu-2ubuntu122.04, mixed mode,
sharing)
```

Note: If JDK is not installed, install JDK and proceed to next step.

5. Install Scala. To install scala

```
$ sudo apt-get install scala -y
```

6. Verification of scala installation with

```
$ scala -version
Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL
$
```



7. Now launch scala with the command

```
$ scala
Welcome to Scala 2.11.12 (OpenJDK 64-Bit Server VM, Java 17.0.5).
Type in expressions for evaluation. Or try :help.
```

```
Scala>
```

7. To exit from Scala shell, Press CTRL+D

8. Install Apache Spark

```
$ wget https://archive.apache.org/dist/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz
or
download the above file and follow the steps
```

9. Extract the downloaded file with the following command:

```
$ tar -xvzf spark-3.3.1-bin-hadoop3.tgz
```

10. Move the extracted directory to the **/mnt** directory with the following command:

```
$ mv spark-3.3.1-bin-hadoop3 /mnt/spark
```

11. Configure the Apache Spark

In this step, we need to edit the **.bashrc** file and define the Apache Spark path.

```
$ nano ~/.bashrc
```

At end of file add the following lines:

```
export SPARK_HOME=/mnt/spark export
PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

Save and close the file

12. Then reload the environment variable with the following command.

```
$ source ~/.bashrc
```

13. Next, start the Apache Spark with the command: `$ start-master.sh`

output:

```
starting org.apache.spark.deploy.master.Master, logging to
/mnt/spark/logs/spark-root-org.apache.spark.deploy.master.Master-1spark.out
```

14. To stop the Apache Spark, run the following command: `stop-master.sh`

15. To run pyspark

```
$ pyspark
```

"help", "copyright", "credits" or "license" for more information.

```
23/02/28 12:35:25 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to
another address Setting default log level to "WARN".
```

```
23/02/28 12:35:26 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
Welcome to
```

Using Python version 3.10.6 (main, Nov 14 2022 16:10:14)

Spark context available as 'sc' (master = local[\*], app id = local1677567927751).

>>>

## Apache Spark

### 1. Create a DataFrame using SparkSession

```
# importing python libraries
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import avg

>>> spark = (SparkSession.builder.appName("AuthorsAges").getOrCreate())
23/03/01 04:38:40 WARN SparkSession: Using an existing Spark session; only runtime SQL
configurations will take effect.
```

```
# Create a DataFrame
```

```
>>> data_df = spark.createDataFrame([("Bunti", 20), ("Dileep", 31), ("Zinta", 30), ("Tirumal",
35), ("Bunti", 25)], ["name", "age"])
```

```
>>># displaying data frame
```

```
>>> data_df.show()
```

```
+-----+----+
| name|age|
+-----+----+
| Bunti| 20|
| Dileep| 31|
| Zinta| 30|
| Tirumal| 35|
| Bunti| 25|
+-----+----+
```

```
# computing average by Group the same names together, aggregate their ages
```

```
>>> avg_df = data_df.groupBy("name").agg(avg("age"))
```

```
>>> avg_df.show()
```

```
+-----+-----+
| name|avg(age)|
+-----+-----+
| Dileep| 31.0|
| Bunti| 22.5|
| Tirumal| 35.0|
| Zinta| 30.0|
+-----+-----+
```

### 2. Creating a Data frame using schema, static data and session

```
#importing libraries
```

```
# Defining schema using DDL
```

```
>>> schema = "`Id` INT, `First` STRING, `Last` STRING, `Url` STRING, `Published`
STRING, `Hits` INT, `Campaigns` ARRAY<STRING>"
```

```
# Create our static data
```

```
>>> data=[[1, "Jumanji", "Damodhar", "https://tinyurl.1", "1/4/2016", 4535,
["twitter", "LinkedIn"]],
```

```
[2, "Bunti", "Venigalla", "https://tinyurl.2", "5/5/2018", 8908, ["twitter", "LinkedIn"]],
[3, "Dundi", "Leena", "https://tinyurl.3", "6/7/2019", 7659, ["web", "twitter", "FB",
"LinkedIn"]],
[4, "Tirumal", "Das", "https://tinyurl.4", "5/12/2018", 10568, ["twitter", "FB"]],
[5, "Matha", "NitaiSevini", "https://tinyurl.5", "5/14/2014", 40578, ["web", "twitter", "FB",
"LinkedIn"]],
[6, "Ram", "Krishna", "https://tinyurl.6", "3/2/2015", 25568, ["twitter", "LinkedIn"]]
]
```

```
# Create a SparkSession
```

```
>>> # Create a SparkSes>>> # Print the schema used by Spark to process the DataFrame
```

```
# Displaying a Schema
```

```
>>> print(blogs_df.printSchema())
```

```
root
```

```
|-- Id: integer (nullable = true)
|-- First: string (nullable = true)
|-- Last: string (nullable = true)
|-- Url: string (nullable = true)
|-- Published: string (nullable = true)
|-- Hits: integer (nullable = true)
|-- Campaigns: array (nullable = true)
|   |-- element: string (containsNull = true)
```

```
None
```

```
>>>
```

```
>>>spark = SparkSession.builder.appName("sparktask_2").getOrCreate())
```

```
# Create a DataFrame using the schema defined above
```

```
>>> blogs_df = spark.createDataFrame(data, schema)
```

```
# Show the DataFrame; it should reflect our table above
```

```
>>> blogs_df.show()
```

```
+---+-----+-----+-----+-----+-----+
| Id| First|   Last|      Url|Published| Hits|      Campaigns|
+---+-----+-----+-----+-----+-----+
| 1|Jumanji| Damodhar|https://tinyurl.1| 1/4/2016| 4535| [twitter, LinkedIn]|
| 2| Bunti| Venigalla|https://tinyurl.2| 5/5/2018| 8908| [twitter, LinkedIn]|
| 3| Dundi|   Leena|https://tinyurl.3| 6/7/2019| 7659|[web, twitter, FB...|
| 4|Tirumal|   Das|https://tinyurl.4|5/12/2018|10568|   [twitter, FB]|
| 5| Matha|NitaiSevini|https://tinyurl.5|5/14/2014|40578|[web, twitter, FB...|
| 6|  Ram| Krishna|https://tinyurl.6| 3/2/2015|25568| [twitter, LinkedIn]|
+---+-----+-----+-----+-----+-----+

```

```
>>>
```

### 3. # Create the dataframe

```
#import the pyspark module
>>>import pyspark
#import SparkSession for creating a session
>>> from pyspark.sql import SparkSession
#import the col function
>>> from pyspark.sql.functions import col

#create an app named sparksidd
>>> spark_app = SparkSession.builder.appName('sparksidd').getOrCreate()

# create student data with 5 rows and 6 attributes
students=[{'rollno':'21DSC01','name':'Sravan','age':22,'height':5.79,'weight':67,'address':'Guntur'},
          {'rollno':'21DSC02','name':'Sanghi','age':22,'height':5.3,'weight':54,'address':'Tenali'},
          {'rollno':'21DSC03','name':'Gnanesh','age':22,'height':5.39,'weight':47,'address':'Paritala'},
          {'rollno':'21DSC04','name':'Rohith','age':21,'height':5.69,'weight':48,'address':'Anantapuram'},
          {'rollno':'21DSC05','name':'Srinu','age':21,'height':5.59,'weight':54,'address':'Tirupati'}
]
```

# create the dataframe

```
>>> df = spark_app.createDataFrame(students)
```

#display dataframe

```
>>> df.show()
```

```
+-----+---+-----+-----+-----+
| address|age|height| name| rollno|weight|
+-----+---+-----+-----+-----+
| Guntur| 22| 5.79| Sravan|21DSC01| 67|
| Tenali| 22| 5.3| Sanghi|21DSC02| 54|
| Paritala| 22| 5.39|Gnanesh|21DSC03| 47|
|Anantapuram| 21| 5.69| Rohith|21DSC04| 48|
| Tirupati| 21| 5.59| Srinu|21DSC05| 54|
+-----+---+-----+-----+-----+
```

### 4. Creating a row and access individual columns

```
>>> from pyspark.sql import Row
>>> blog_row = Row(6, "Ram", "Krishna", "https://tinyurl.6", "3/2/2015", 25568,["twitter",
"LinkedIn"])
>>>blog_row[0]
6
>>> blog_row[2]
'Krishna'
>>> blog_row[1]
'Ram'
```

```
>>> blog_row[4]
'3/2/2015'
>>>
```

## 5. Task #5

### # Create Data frames from Row objects

```
>>> from pyspark.sql import Row
# define a row
>>> rows = [Row("Matha NitaiSevini", "CA"), Row("Ram Krishna", "CA")]
# assign row to dataframe
>>> authors_df = spark.createDataFrame(rows, ["Authors", "State"])
# display data frame
>>> authors_df.show()
+-----+-----+
|   Authors|State|
+-----+-----+
|Matha NitaiSevini| CA|
|   Ram Krishna| CA|
+-----+-----+
```

```
>>>
```

## 6. Task #6

### # Reading an CSV file in to Data Frame - With out specifying header in reading Steps

```
# Importing libraries required
# specifying file csv file with path (absolute/relative)
# specifying schema of csv file
# creating a data frame
# Displaying a data frame
# data file
$ cat ./datasets/movies.csv
actor,title,year
pk,Bheemla Nayak,2020 nbk,akhandha,2021 prabhas,Radhe
syam,2022 bunny,ala vaikunta puramlo,2020 nani,syam singha
Rai,2022
```

### Case#1: With out specifying header in reading

```
# Importing libraries
>>> from pyspark.sql.types import *
# specifying schema
>>> schema="actor STRING, movie STRING, year INT"
# specifying file
>>> csv_file = "/home/hduser/datasets/movies.csv"
# creating a data frame
>>> movies_df = spark.read.csv(csv_file, schema=schema)
```

```
# Displaying a data frame
```

```
>>> movies_df.show()
```

```
+-----+-----+-----+
| actor|          movie|year|
+-----+-----+-----+
| actor|          title|null|
|  pk|    Bheemla Nayak|2020|
| nbk|      akhanda|2021|
|prabhas|    Radhe syam|null|
| bunny|ala vaikunta puramlo|2020|
| nani|    syam singha Rai|2022|
+-----+-----+-----+
```

### **Reading an CSV file in to Data Frame**

#### **Case#2: With specifying header in reading**

```
# Importing libraries
```

```
>>> from pyspark.sql.types import *
```

```
# specifying schema
```

```
>>> schema="actor STRING, movie STRING, year INT"
```

```
# specifying file
```

```
>>> csv_file = "/home/hduser/datasets/movies.csv"
```

```
# creating a data frame with header
```

```
>>> movies_df = spark.read.csv(csv_file, header=True,schema=schema)
```

```
>>> movies_df.show()
```

```
23/03/04 00:11:54 WARN CSVHeaderChecker: CSV header does not conform to the schema.
```

```
Header: actor, title, year
```

```
Schema: actor, movie, year
```

```
Expected: movie but found: title
```

```
CSV file: file:///home/hduser/datasets/movies.csv
```

```
+-----+-----+-----+
| actor|          movie|year|
+-----+-----+-----+
|  pk|    Bheemla Nayak|2020|
| nbk|      akhanda|2021|
|prabhas|    Radhe syam|null|
| bunny|ala vaikunta puramlo|2020|
| nani|    syam singha Rai|2022|
+-----+-----+-----+
```

```
>>>
```

#### **Case3: With specifying session with out specifying schema**

```
# Importing libraries
```

```
>>> from pyspark.sql import SparkSession
```

```
# specifying session
```

```
>>> spark = (SparkSession
```

```

.builder
.appName("PySparkmovies")
.getOrCreate()
# specifying file
>>> csv_file = "/home/hduser/datasets/movies.csv"
# creating a data frame with header
>>> movies_df = (spark.read.format("csv")
.option("inferSchema", "true")
.option("header", "true")
.load(csv_file))
>>> movies_df.show()
+-----+-----+-----+
| actor|      title| year|
+-----+-----+-----+
|  pk|    Bheemla Nayak|2020.0|
| nbk|      akhanda|2021.0|
|prabhas|    Radhe syam|2022.0|
| bunny|ala vaikunta puramlo|2020.0|
|  nani|    syam singha Rai|2022.0|
+-----+-----+-----+

>>>

```

## 7. Task #7 Reading an CSV file(firedata) in to Data Frame

```

>>> # Commaon data frame operations
>>> Case1: To read a CSV file

>>> from pyspark.sql.types import *
>>> fire_schema = StructType([StructField('CallNumber', IntegerType(), True),
    StructField('UnitID', StringType(), True),
    StructField('IncidentNumber', IntegerType(), True),
    StructField('CallType', StringType(), True),
    StructField('CallDate', StringType(), True),
    StructField('WatchDate', StringType(), True),
    StructField('CallFinalDisposition', StringType(), True),
    StructField('AvailableDtTm', StringType(), True),
    StructField('Address', StringType(), True),
    StructField('City', StringType(), True),
    StructField('Zipcode', IntegerType(), True),
    StructField('Battalion', StringType(), True),
    StructField('StationArea', StringType(), True),
    StructField('Box', StringType(), True),
    StructField('OriginalPriority', StringType(), True),
    StructField('Priority', StringType(), True),
    StructField('FinalPriority', IntegerType(), True),
    StructField('ALSUnit', BooleanType(), True),
    StructField('CallTypeGroup', StringType(), True),
    StructField('NumAlarms', IntegerType(), True),

```



```

    StructField('UnitType', StringType(), True),
    StructField('UnitSequenceInCallDispatch', IntegerType(), True),
    StructField('FirePreventionDistrict', StringType(), True),
    StructField('SupervisorDistrict', StringType(), True),
    StructField('Neighborhood', StringType(), True),
    StructField('Location', StringType(), True),
    StructField('RowID', StringType(), True),
    StructField('Delay', FloatType(), True)])

```

```

>>>
>>> sf_fire_file = "/home/hduser/sf-fire-calls.csv"
>>># creating a data frame by using schema & file
>>>fire_df = spark.read.csv(sf_fire_file, header=True, schema=fire_schema)
>>> # displaying the data frame
>>> fire_df.show()

```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+
|CallNumber|UnitID|IncidentNumber|    CallType| CallDate| WatchDate|
CallFinalDisposition|    AvailableDtTm|    Address|City|Zipcode|Battalion|
StationArea| Box|OriginalPriority|Priority|FinalPriority|ALSUnit|CallTypeGroup|
NumAlarms|    UnitType|UnitSequenceInCallDispatch|FirePreventionDistrict|
SupervisorDistrict|    Neighborhood|    Location|    RowID| Delay|
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+
| 20110016| T13|    2003235| Structure Fire|01/11/2002|01/10/2002|    Other|
01/11/2002 01:51:...|2000 Block of CAL...| SF| 94109|    B04|    38|3362|    3|
3|    3| false|    null|    1|    TRUCK|    2|    4|    5|
Pacific Heights|(37.7895840679362...|020110016-T13|    2.95|
| 20110022| M17|    2003241|Medical Incident|01/11/2002|01/10/2002|    Other|
01/11/2002 03:01:...|0 Block of SILVER...| SF| 94124|    B10|    42|6495|    3|
3|    3| true|    null|    1|    MEDIC|    1|    10|
10|Bayview Hunters P...|(37.7337623673897...|020110022-M17|    4.7|
| 20110023| M41|    2003242|Medical Incident|01/11/2002|01/10/2002|    Other|
01/11/2002 02:39:...|MARKET ST/MCALLIS...| SF| 94102|    B03|    01|1455|
3|    3|    3| true|    null|    1|    MEDIC|    2|    3|
6|    Tenderloin|(37.7811772186856...|020110023-M41|2.4333334|
| 20110032| E11|    2003250| Vehicle Fire|01/11/2002|01/10/2002|    Other|
01/11/2002 04:16:...|APPLETON AV/MISSI...| SF| 94110|    B06|    32|5626|    3|
3|    3| false|    null|    1|    ENGINE|    1|    6|    9|
Bernal Heights|(37.7388432849018...|020110032-E11|    1.5|
| 20110043| B04|    2003259| Alarms|01/11/2002|01/10/2002|    Other|

```

```

01/11/2002 06:01|flights_df = spark.read.csv(csv_file, schema=schema):...|1400 Block of
SUT...| SF| 94109| B04| 03|3223| 3| 3| 3| false| null| 1|
CHIEF| 2| 4| 2| Western Addition|
(37.7872890372638...|020110043-B04|3.4833333|
| 20110072| T08| 2003279| Structure Fire|01/11/2002|01/11/2002| Other|
01/11/2002 08:03:...| BEALE ST/FOLSOM ST| SF| 94105| B03| 35|2122|
3| 3| 3| false| null| 1| TRUCK| 2| 3|
6|Financial Distric...|(37.7886866619654...|020110072-T08| 1.75|
| 20110125| E33| 2003301| Alarms|01/11/2002|01/11/2002| Other|
01/11/2002 09:46:...|0 Block of FARALL...| SF| 94112| B09| 33|8324| 3|
3| 3| false| null| 1| ENGINE| 2| 9|
11|Oceanview/Merced/...|(37.7140353531157...|020110125-E33|2.7166667|
| 20110130| E36| 2003304| Alarms|01/11/2002|01/11/2002| Other|
01/11/2002 09:58:...|600 Block of POLK ST| SF| 94102| B02| 03|3114| 3|
3| 3| false| null| 1| ENGINE| 1| 2| 6|
Tenderloin|(37.7826266328595...|020110130-E36|1.7833333|
| 20110197| E05| 2003343| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 12:06:...|1500 Block of WEB...| SF| 94115| B04| 05|3513| 3|
3| 3| false| null| 1| ENGINE| 1| 4| 5|
Japantown|(37.784958590666...|020110197-E05|1.5166667|
| 20110215| E06| 2003348| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 01:08:...|DIAMOND ST/MARKET ST| SF| 94114| B05| 06|5415|
3| 3| 3| false| null| 1| ENGINE| 1| 5|
8|Castro/Upper Market|(37.7618954753708...|020110215-E06|2.7666667|
| 20110274| M07| 2003381| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 03:31:...|2700 Block of MIS...| SF| 94110| B06| 11|5525| 1|
1| 2| true| null| 1| MEDIC| 1| 6| 9|
Mission|(37.7530339738059...|020110274-M07|2.1833334|
| 20110275| T15| 2003382| Structure Fire|01/11/2002|01/11/2002| Other|
01/11/2002 02:59:...|BRUNSWICK ST/GUTT...| SF| 94112| B09| 43|6218|
3| 3| 3| false| null| 1| TRUCK| 1| 9| 11|
Excelsior|(37.7105545807996...|020110275-T15| 2.5|
| 20110304| E03| 2003399| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 04:22:...|1000 Block of SUT...| SF| 94109| B04| 03|1557| 3|
3| 3| false| null| 1| ENGINE| 1| 4| 3|
Nob Hill|(37.7881263034393...|020110304-E03|2.4166667|
| 20110308| E14| 2003403| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 04:18:...|100 Block of 21ST...| SF| 94121| B07| 14|7173| 3|
3| 3| false| null| 1| ENGINE| 1| 7| 1|
Outer Richmond|(37.7850084431077...|020110308-E14| 4.95|
| 20110313| B10| 2003408| Structure Fire|01/11/2002|01/11/2002| Other|
01/11/2002 04:09:...|700 Block of CAPP ST| SF| 94110| B06| 07|5472| 3|
3| 3| false| null| 1| CHIEF| 6| 6| 9|
Mission|(37.7547064357942...|020110313-B10|1.4166666|
| 20110313| D3| 2003408| Structure Fire|01/11/2002|01/11/2002| Other|
01/11/2002 04:09:...|700 Block of CAPP ST| SF| 94110| B06| 07|5472| 3|
3| 3| false| null| 1| CHIEF| 4| 6| 9|

```

```

Mission|(37.7547064357942...|020110313-D3|2.5333333|
| 20110313| E32| 2003408| Structure Fire|01/11/2002|01/11/2002| Other|
01/11/2002 04:09:...|700 Block of CAPP ST| SF| 94110| B06| 07|5472| 3|
3| 3| true| null| 1| ENGINE| 8| 6| 9|
Mission|(37.7547064357942...|020110313-E32|1.8833333|
| 20110315| RC2| 2003409| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 04:34:...|200 Block of LAGU...| SF| 94116| B08| 20|8635| 3|
3| 3| true| null| 1|RESCUE CAPTAIN| 2| 8|
7| West of Twin Peaks|(37.7501117393668...|020110315-RC2| 5.35|
| 20110330| E14| 2003417| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 04:51:...|BALBOA ST/PARK PR...| SF| 94118| B07| 31|7145|
3| 3| 3| false| null| 1| ENGINE| 1| 7|
1| Inner Richmond|(37.7768682293368...|020110330-E14| 2.0|
| 20110330| M12| 2003417| Medical Incident|01/11/2002|01/11/2002| Other|
01/11/2002 04:51:...|BALBOA ST/PARK PR...| SF| 94118| B07| 31|7145| 3| 3|
3| true| null| 1| MEDIC| 2| 7|
1| Inner Richmond|(37.7768682293368...|020110330-M12|1.8166667|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+

```

only showing top 20 rows

```
>>>
```

## Task #8 # Reading an CSV file(departuredelays) in to Data Frame

### # importing Libraries

```
>>> from pyspark.sql import SparkSession
```

### # specifying the file with path

```
>>> csv_file = "./departuredelays.csv"
```

### #Specifying the schema

```
>>> f_schema="date STRING, delay INT, distance INT, origin STRING, destination
STRING"
```

### # Creating a dataframe

```
>>> flights_df = spark.read.csv(csv_file,header=True, schema=f_schema)
```

### # displaying a data frame

```
>>> flights_df.show()
```

```

+-----+-----+-----+-----+-----+
| date|delay|distance|origin|destination|
+-----+-----+-----+-----+-----+
|01011245| 6| 602| ABE| ATL|
|01020600| -8| 369| ABE| DTW|
|01021245| -2| 602| ABE| ATL|
|01020605| -4| 602| ABE| ATL|
|01031245| -4| 602| ABE| ATL|
|01030605| 0| 602| ABE| ATL|
|01041243| 10| 602| ABE| ATL|

```

```

|01040605| 28| 602| ABE| ATL|
|01051245| 88| 602| ABE| ATL|
|01050605| 9| 602| ABE| ATL|
|01061215| -6| 602| ABE| ATL|
|01061725| 69| 602| ABE| ATL|
|01061230| 0| 369| ABE| DTW|
|01060625| -3| 602| ABE| ATL|
|01070600| 0| 369| ABE| DTW|
|01071725| 0| 602| ABE| ATL|
|01071230| 0| 369| ABE| DTW|
|01070625| 0| 602| ABE| ATL|
|01071219| 0| 569| ABE| ORD|
|01080600| 0| 369| ABE| DTW| +-
-----+-----+-----+-----+-----+ only
showing top 20 rows

```

```

>>> # Query1: Find all flights whose distance is greater than 1,000 miles.
# Creating a view file
>>> flights_df.createOrReplaceTempView("us_flights_delay_tbl") >>>

```

Now we write queries on

```

# Query1: Find all flights whose distance is greater than 1,000 miles.

```

```

>>> spark.sql("""SELECT distance, origin, destination
... FROM us_flights_delay_tbl WHERE distance > 1000
... ORDER BY distance DESC""").show(10)

```

```

+-----+-----+-----+
|distance|origin|destination|
+-----+-----+-----+
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK|
| 4330| HNL| JFK| +---
-----+-----+-----+ only
showing top 10 rows

```

```

>>>

```

```

# Query1: Find all flights whose distance is less than 1,000 miles.

```

```

>>> spark.sql("""SELECT distance, origin, destination

```

```

... FROM us_flights_delay_tbl WHERE distance < 1000
... ORDER BY distance ASC""").show(10)
+-----+-----+-----+
|distance|origin|destination|
+-----+-----+-----+
| 21| IAD| DCA|
| 23| HRL| BRO|
| 27| PSG| WRG|
| 27| PSG| WRG|
| 27| PSG| WRG|
| 27| PSG| WRG|
| 27| PSG| WRG|
| 27| PSG| WRG|
| 27| PSG| WRG|
| 27| PSG| WRG|
| 27| PSG| WRG| +----
----+-----+-----+ only
showing top 10 rows >>>

```

### Task # 9 Creating a data frame from Json file

```

# data file movies.json
hduser@ramana:~/datasets$ cat movies.json
{"Id":1, "actor": "PK", "title":"Bheemla Nayak", "year":2020, "Rating":4.5}
{"Id":2, "actor": "NBK", "title":"Akanda", "year":2021, "Rating":4.6}
{"Id":3, "actor": "Prbhas", "title":"RadheSyam", "year":2022, "Rating":3.5}
{"Id":4, "actor": "Bunny", "title":"Ala Vaikuna Puramlo", "year":2020, "Rating":4.5}
{"Id":1, "actor": "Nani", "title":"Syam Singh Rai", "year":2022, "Rating":3.5}
hduser@ramana:~/datasets$

# impoting libraries

>>>from pyspark.sql import SparkSession

>>>from pyspark.sql.types      import      StructType,StructField,
StringType, IntegerType,BooleanType,DoubleType

>>>spark = SparkSession.builder.appName("read JSON").getOrCreate()

24/06/05 08:50:55 WARN SparkSession: Using an existing Spark session; only runtime SQL
configurations will take effect.
>>> movies_df = spark.read.json("./movies.json")

>>> movies_df.printSchema()
root
|-- Id: long (nullable = true)
|-- Rating: double (nullable = true)

```

```
|-- actor: string (nullable = true)
|-- title: string (nullable = true)
|-- year: long (nullable = true)
```

```
>>> movies_df.show()
+---+-----+-----+-----+
| Id|Rating| actor|          title|year|
+---+-----+-----+-----+
| 1| 4.5|  PK|    Bheemla Nayak|2020|
| 2| 4.6| NBK|        Akanda|2021|
| 3| 3.5|Prbhas|    RadheSyam|2022|
| 4| 4.5| Bunny|Ala Vaikuna Puramlo|2020|
| 1| 3.5| Nani|    Syam Singh Rai|2022|
+---+-----+-----+-----+
```

### Task # 10 - Writing data from a data frame to Json file

```
# data file movies.json
hduser@ramana:~/datasets$ cat movies.json
{"Id":1, "actor": "PK", "title":"Bheemla Nayak", "year":2020, "Rating":4.5}
{"Id":2, "actor": "NBK", "title":"Akanda", "year":2021, "Rating":4.6}
{"Id":3, "actor": "Prbhas", "title":"RadheSyam", "year":2022, "Rating":3.5}
{"Id":4, "actor": "Bunny", "title":"Ala Vaikuna Puramlo", "year":2020, "Rating":4.5} {"Id":1,
"actor": "Nani", "title":"Syam Singh Rai", "year":2022, "Rating":3.5}
hduser@ramana:~/datasets$
```

```
# impoting libraries
```

```
>>>from pyspark.sql import SparkSession
```

```
>>>from pyspark.sql.types import StructType,StructField, StringType,
IntegerType,BooleanType,DoubleType
```

```
>>>spark = SparkSession.builder \
    .master("local") \
    .appName("PySpark Read JSON") \
    .getOrCreate()
```

```
23/03/05 08:50:55 WARN SparkSession: Using an existing Spark session; only runtime SQL
configurations will take effect.
```

```
>>> movies_df = spark.read.json("/home/hduser/datasets/movies.json")
```

```
>>> movies_df.printSchema()
root
|-- Id: long (nullable = true)
|-- Rating: double (nullable = true)
```

```
|-- actor: string (nullable = true)
|-- title: string (nullable = true)
|-- year: long (nullable = true)
```

```
>>> movies_df.show()
+---+-----+-----+-----+
| Id|Rating| actor|      title|year|
+---+-----+-----+-----+
| 1| 4.5|  PK|   Bheemla Nayak|2020|
| 2| 4.6| NBK|      Akanda|2021|
| 3| 3.5|Prbhas|   RadheSyam|2022|
| 4| 4.5| Bunny|Ala Vaikuna Puramlo|2020|
| 1| 3.5| Nani|   Syam Singh Rai|2022|
+---+-----+-----+-----+
```

```
>>> # Writing PySpark dataframe into JSON File
```

```
# Saving modes
```

PySpark DataFrameWriter also has a method mode() to specify saving mode.

**overwrite** – mode is used to overwrite the existing file.

append – To add the data to the existing file.

ignore – Ignores write operation when the file already exists.

error – This is a default option when the file already exists, it returns an error.

```
>>> movies_df.write.mode('Overwrite').json("/home/hduser/datasets/wmovies.json")
>>>
```

```
#
```

```
hduser@ramana:~/datasets$ ls
```

```
blogs.json      flights      movies.csv  pincodes.json  wmovies.json
```

```
departuredelays.csv  iot_devices.json  movies.json  us_flights_delay
```

```
hduser@ramana:~/datasets$ cd wmovies.json/
```

```
hduser@ramana:~/datasets/wmovies.json$ ls part-00000-0f6e7848-882e-4cee-ab26-26b3e65ad625-c000.json
```

```
hduser@ramana:~/datasets/wmovies._SUCCESS
```

## Task # 11 Creating a data frame from Json file (large file)

```
# data file pincodes.json in /home/datasets/pincodes.json
```

```
hduser@ramana:~/datasets$ ls -l
```

```
total 50712
```

```
-rw-rw-r-- 1 hduser hduser  915 Mar  2 17:15 blogs.json
```

```
-rw-rw-r-- 1 hduser hduser 33396236 Feb 26 21:44 departuredelays.csv
```

```
drwxrwxr-x 2 hduser hduser  4096 Feb 28 08:40 flights -rw-rw-r-- 1
```

```
hduser hduser 139897 Feb 26 21:45 iot_devices.json
```

```
-rw-rw-r-- 1 hduser hduser  139 Mar  4 00:08 movies.csv
```

```
-rw-rw-r-- 1 hduser hduser    386 Mar  5 06:24 movies.json -rw-rw-r-
- 1 hduser hduser 18360059 Mar  5 06:47 pincodes.json drwxr-xr-x 2
hduser hduser    4096 Feb 28 06:59 us_flights_delay drwxr-xr-x 2
hduser hduser    4096 Mar  5 08:55 wmovies.json
hduser@ramana:~/datasets$
```

```
# impoting libraries
```

```
>>>from pyspark.sql import SparkSession
```

```
>>> spark = SparkSession.builder.appName("Read pinode JSON").getOrCreate()
23/03/05 09:03:09 WARN SparkSession: Using an existing Spark session; only runtime SQL
configurations will take effect.
```

```
>>> pincodes_df = spark.read.json("/home/hduser/datasets/pincodes.json")
```

```
>>> pincodes_df.printSchema()                                root
```

```
-- districtName: string (nullable = true)
-- officeName: string (nullable = true)
-- pincode: long (nullable = true)
-- stateName: string (nullable = true)
-- taluk: string (nullable = true)
```

```
>>> pincodes_df.show()
```

```
+-----+-----+-----+-----+-----+
|districtName|    officeName|pincode|    stateName|    taluk|
+-----+-----+-----+-----+-----+
| Adilabad|    Ada B.O| 504293|ANDHRA PRADESH| Asifabad|
| Adilabad| Andugulpet B.O| 504231|ANDHRA PRADESH|Luxettipet|
| Adilabad| Annaram B.O| 504201|ANDHRA PRADESH| Chennur|
| Adilabad| Arli (T) B.O| 504312|ANDHRA PRADESH| Adilabad|
| Adilabad| Bambara B.O| 504295|ANDHRA PRADESH| Asifabad|
| Adilabad| Bangalpet B.O| 504106|ANDHRA PRADESH| Nirmal|
| Adilabad| Bansapalli B.O| 504306|ANDHRA PRADESH| Nirmal|
| Adilabad| Bejjur B.O| 504299|ANDHRA PRADESH|Sirpur (t)|
| Adilabad| Bellalbad B.O| 504202|ANDHRA PRADESH| Khanapur|
| Adilabad|Bhainsa S.O (Adil...| 504103|ANDHRA PRADESH| Mudhole|
| Adilabad| Birvelli B.O| 504306|ANDHRA PRADESH| Nirmal|
| Adilabad| Burguda B.O| 504293|ANDHRA PRADESH| Asifabad|
| Adilabad|Chichdhari Khanap...| 504346|ANDHRA PRADESH| Utnoor|
| Adilabad| Chintaguda B.O| 504296|ANDHRA PRADESH|Sirpur (t)|
| Adilabad|Coal Chemical Com...| 504302|ANDHRA PRADESH|Mancherial|
| Adilabad| Dantanpalli B.O| 504311|ANDHRA PRADESH| Utnoor|
| Adilabad| Deepaiguda B.O| 504309|ANDHRA PRADESH| Adilabad|
| Adilabad| Dehgaon B.O| 504273|ANDHRA PRADESH| Asifabad|
| Adilabad| Dhaboli B.O| 504313|ANDHRA PRADESH| Utnoor|
| Adilabad| Dhannoor B.O| 504304|ANDHRA PRADESH| Boath|
+-----+-----+-----+-----+-----+
```

```
only showing top 20 rows
```



```
>>>
```

## Task # 12 Writing a data frame in to Json file(large file)

```
# data file pincodes.json in /home/datasets/pincodes.json
```

```
hduser@ramana:~/datasets$ ls -l
```

```
total 50712
```

```
-rw-rw-r-- 1 hduser hduser    915 Mar  2 17:15 blogs.json
-rw-rw-r-- 1 hduser hduser 33396236 Feb 26 21:44 departureDelays.csv
drwxrwxr-x 2 hduser hduser   4096 Feb 28 08:40 flights -rw-rw-r-- 1
hduser hduser  139897 Feb 26 21:45 iot_devices.json
-rw-rw-r-- 1 hduser hduser    139 Mar  4 00:08 movies.csv
-rw-rw-r-- 1 hduser hduser    386 Mar  5 06:24 movies.json -rw-rw-r-
- 1 hduser hduser 18360059 Mar  5 06:47 pincodes.json drwxr-xr-x 2
hduser hduser   4096 Feb 28 06:59 us_flights_delay drwxr-xr-x 2
hduser hduser   4096 Mar  5 08:55 wmovies.json
hduser@ramana:~/datasets$
```

```
# impoting libraries
```

```
>>>from pyspark.sql import SparkSession
```

```
>>> spark = SparkSession.builder.appName("Read pinode JSON").getOrCreate()
```

```
23/03/05 09:03:09 WARN SparkSession: Using an existing Spark session; only runtime SQL
configurations will take effect.
```

```
>>> pincodes_df = spark.read.json("/home/hduser/datasets/pincodes.json")
```

```
>>> pincodes_df.printSchema()                                root
```

```
-- districtName: string (nullable = true)
-- officeName: string (nullable = true)
-- pincode: long (nullable = true)
-- stateName: string (nullable = true)
-- taluk: string (nullable = true)
```

```
>>> pincodes_df.show()
```

```
+-----+-----+-----+-----+-----+
|districtName|officeName|pincode|stateName|taluk|
+-----+-----+-----+-----+-----+
|Adilabad|Ada B.O|504293|ANDHRA PRADESH|Asifabad|
|Adilabad|Andugulpet B.O|504231|ANDHRA PRADESH|Luxettipet|
|Adilabad|Annaram B.O|504201|ANDHRA PRADESH|Chennur|
|Adilabad|Arli (T) B.O|504312|ANDHRA PRADESH|Adilabad|
|Adilabad|Bambara B.O|504295|ANDHRA PRADESH|Asifabad|
|Adilabad|Bangalpet B.O|504106|ANDHRA PRADESH|Nirmal|
|Adilabad|Bansapalli B.O|504306|ANDHRA PRADESH|Nirmal|
|Adilabad|Bejjur B.O|504299|ANDHRA PRADESH|Sirpur (t)|
|Adilabad|Bellalbad B.O|504202|ANDHRA PRADESH|Khanapur|
|Adilabad|Bhainsa S.O (Adil...|504103|ANDHRA PRADESH|Mudhole|
```

Adilabad	Birvelli B.O	504306	ANDHRA PRADESH	Nirmal
Adilabad	Burguda B.O	504293	ANDHRA PRADESH	Asifabad
Adilabad	Chichdhari Khanap...	504346	ANDHRA PRADESH	Utnoor
Adilabad	Chintaguda B.O	504296	ANDHRA PRADESH	Sirpur (t)
Adilabad	Coal Chemical Com...	504302	ANDHRA PRADESH	Mancherial
Adilabad	Dantanpalli B.O	504311	ANDHRA PRADESH	Utnoor
Adilabad	Deepaiguda B.O	504309	ANDHRA PRADESH	Adilabad
Adilabad	Dehgaon B.O	504273	ANDHRA PRADESH	Asifabad
Adilabad	Dhaboli B.O	504313	ANDHRA PRADESH	Utnoor
Adilabad	Dhannoor B.O	504304	ANDHRA PRADESH	Boath

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

only showing top 20 rows

>>>

>>> pincodes\_df.write.mode('Overwrite').json("/home/hduser/datasets/dftjpincodes.json")

>>>

hduser@ramana:~/datasets\$ ls -l

total 50716

```
-rw-rw-r-- 1 hduser hduser    915 Mar  2 17:15 blogs.json
-rw-rw-r-- 1 hduser hduser 33396236 Feb 26 21:44 departuredelays.csv
drwxr-xr-x 2 hduser hduser   4096 Mar  5 09:07 dftjpincodes.json
drwxrwxr-x 2 hduser hduser   4096 Feb 28 08:40 flights -rw-rw-r-- 1
hduser hduser  139897 Feb 26 21:45 iot_devices.json
-rw-rw-r-- 1 hduser hduser    139 Mar  4 00:08 movies.csv
-rw-rw-r-- 1 hduser hduser    386 Mar  5 06:24 movies.json -rw-rw-r-
- 1 hduser hduser 18360059 Mar  5 06:47 pincodes.json drwxr-xr-x 2
hduser hduser   4096 Feb 28 06:59 us_flights_delay drwxr-xr-x 2
hduser hduser   4096 Mar  5 08:55 wmovies.json
hduser@ramana:~/datasets$ cd dftjpincodes.json/
hduser@ramana:~/datasets/dftjpincodes.json$ ls
part-00000-16b109e7-6c7c-44c0-be64-91cb04e98bef-c000.json _SUCCESS
hduser@ramana:~/datasets/dftjpincodes.json$
```

### Taks #13 Reading a image file(s) into a Data Frame

images are copied in to a folder trainimages in  
/home/hduser/trainimages

# including the path

```
>>> path = "/home/hduser/datasets/trainimages/"
```

```
>>> binary_files_df = (spark.read.format("binaryFile") .option("pathGlobFilter", "*.jpg")  
...   .load(path))
```

```
>>> binary_files_df.show(5)
```

```
+-----+-----+-----+-----+  
|      path|modificationTime|length|      content|  
+-----+-----+-----+-----+  
|file:/home/hduser...|2023-03-06 02:53:...| 53787|[FF D8 FF E0 00 1...|  
|file:/home/hduser...|2023-03-06 03:00:...| 53424|[FF D8 FF E0 00 1...|  
|file:/home/hduser...|2023-03-06 03:00:...| 53371|[FF D8 FF E0 00 1...|  
|file:/home/hduser...|2023-03-06 02:53:...| 53307|[FF D8 FF E0 00 1...|  
|file:/home/hduser...|2023-03-06 03:02:...| 53236|[FF D8 FF E0 00 1...|  
+-----+-----+-----+-----+
```

only showing top 5 rows

```
>>>
```

## Scala

Scala is an acronym for “Scalable Language”. It is a general-purpose programming language designed for the programmers who want to write programs in a concise, elegant, and type-safe way. Scala enables programmers to be more productive. Scala is developed as an object-oriented and functional programming language.

The design of Scala started in 2001 in the programming methods laboratory at **EPFL (École Polytechnique Fédérale de Lausanne)**. Scala made its first public appearance in January 2004 on the JVM platform and a few months later in June 2004, it was released on the .(dot)NET platform.

Scala is an object-oriented programming language. Everything in Scala is an object and any operations you perform is a method call. Scala, allow you to add new operations to existing classes with the help of implicit classes.

Scala as a programming language that has implemented major functional programming concepts. In Functional programming, every computation is treated as a mathematical function which avoids states and mutable data. The functional programming exhibits following characteristics:

- Power and flexibility
- Simplicity
- Suitable for parallel processing

Scala is a compiler based language which makes Scala execution very fast if you compare it with Python (which is an interpreted language). The compiler in Scala works in similar fashion as Java compiler. It gets the source code and generates Java byte-code that can be executed independently on any standard JVM (Java Virtual Machine).

```
hduser@ramana:~$ spark-shell
```

```
23/03/06 05:43:13 WARN Utils: Your hostname, ramana resolves to a loopback address: 127.0.1.1; using 192.168.0.9 instead (on interface enp2s0)
```

```
23/03/06 05:43:13 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
```

```
Setting default log level to "WARN".
```

```
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
```

```
23/03/06 05:43:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

```
Spark context Web UI available at http://192.168.0.9:4040
```

```
Spark context available as 'sc' (master = local[*], app id = local-1678061602635).
```

```
Spark session available as 'spark'.
```

```
Welcome to
```

```
_____
```

```

/ _/_ _ _ _/_/_
_ \V _V _ \' _/ ' _/
/_/_/ . _ ^ _ , _/_/_/_/_ \ version 3.3.1
/_/_/

```

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 17.0.6)

Type in expressions to have them evaluated.

Type :help for more information.

```
scala>
```

**# declaring a variable which is immutable**

```
scala> val rNo="21dsc01"
rNo: String = 21dsc01
```

```
scala> val sNo="1001"
sNo: String = 1001
```

```
scala> val sNo=1001
sNo: Int = 1001
```

```
# displaying a variable
scala> print("Roll Number is" + rNo)
Roll Number is21dsc01
```

```
scala> print("Roll Number is" + rNo)
Roll Number is21dsc01
```

```
scala> print("Roll Number is " + rNo)
Roll Number is 21dsc01
```

```
scala> print("Serial Number is " + sNo)
Serial Number is 1001
```

```
scala> val rNo = 210301
rNo: Int = 210301
```

```
scala> val rNo = "21dsc01"
rNo: String = 21dsc01 scala>
var rNo = "21dsc01" rNo:
String = 21dsc01
```

```

scala> print("Roll No = " + rNo)
Roll No = 21dsc01

# computation
scala> val x = 10
x: Int = 10

scala> val y = 10
y: Int = 10

scala> val sum = x+y
sum: Int = 20
# with lazy function computatio is done at runtime
scala> lazy val y = 10 y: Int = <lazy>

scala> print(sum)
20 scala>

scala> val x=10
x: Int = 10

scala> lazy val y=20
y: Int = <lazy>

scala> val sum = x+y
sum: Int = 30

scala> print(sum)
30 scala> val
y=10 y: Int = 10

scala> lazy val sum = x+y
sum: Int = <lazy>

scala> print(sum)
20 scala> var
x=10 x: Int = 10

scala> var y=10.5 y:
Double = 10.5 scala>
val sum = x+y sum:
Double = 20.5

scala> print(sum)
20.5
scala>

```

```
scala> val name="Raju"
name: String = Raju
```

```
scala> print("Mera naam " + name)
Mera naam Raju
```

These are the available **string interpolation methods**:

- **s interpolator.**
- **f interpolator.**
- **raw interpolator.**

### **String - s Interpolator**

**S Interpolator** allows the user to use the reference variables to append the data directly.

Case:With Strings

```
scala> val name="Raju"
name: String = Raju
```

```
scala> print(s"Mera naam $name")
Mera naam Raju
scala>
```

```
scala> val name="Raju"
name: String = Raju
```

```
scala> print(s"Mera naam $name")
Mera naam Raju
scala> println(s"Mera naam $name")
Mera naam Raju
```

```
scala>
```

Note: New Line is added after display

Case:With Numbers

```
scala> val x=10 x:
Int = 10
```

```
scala> val y =30
y: Int = 30
```

```
scala> print(s"Sum of $x and $y is ${x+y}")
Sum of 10 and 30 is 40
scala>
```

### **String - f Interpolator**

**f interpolator** to the string literal allows the user to create the formatted string and embed variable references directly in processed string literals.

```
scala> val book="Apache Spark"
book: String = Apache Spark
```

```
scala> val author="Singham Roy"
author: String = Singham Roy
```

```
scala> val price=320
price: Int = 320
```

```
scala> print(f"I purchased $book author $author priced at $price for Rupees
${price*((price*20)/100)}")
I purchased Apache Spark author Singham Roy priced at 320 for Rupees 256
scala>
```

```
scala> print(f"I purchased $book author $author priced at $price for Rupees
${price*((price*20)/100)}")
I purchased Apache Spark author Singham Roy priced at 320 for Rupees 256
scala> print(f"I purchased $book author $author priced at $price for Rupees
${price*((price*20)/100)}%1.1f")
I purchased Apache Spark author Singham Roy priced at 320 for Rupees 256.0
scala> print(f"I purchased $book author $author priced at $price for Rupees
${price*((price*20)/100)}%1.2f")
I purchased Apache Spark author Singham Roy priced at 320 for Rupees 256.00
scala> print(f"I purchased $book author $author priced at $price for Rupees
${price*((price*22)/100)}%1.2f")
I purchased Apache Spark author Singham Roy priced at 320 for Rupees 250.00
scala> print(f"I purchased $book author $author priced at $price for Rupees
${price*((price*21)/100)}%1.2f")
I purchased Apache Spark author Singham Roy priced at 320 for Rupees 253.00
scala> print(f"I purchased $book author $author priced at $price for Rupees
${price*((price*21.5)/100)}%1.2f")
I purchased Apache Spark author Singham Roy priced at 320 for Rupees 251.20
scala>
```

### **String - raw Interpolator**

The raw interpolator does not allow the escaping of literals.

For example, using `\n` with the raw interpolator does not return a newline character.

```
scala> print(f"I purchased $book author $author \n priced at $price for Rupees ${price-
((price*21.5)/100)}%1.2f")
I purchased Apache Spark author Singham Roy
  priced at 320 for Rupees 251.20
scala> print(raw"I purchased $book author $author \n priced at $price for Rupees ${price-
((price*21.5)/100)}%1.2f")
I purchased Apache Spark author Singham Roy \n priced at 320 for Rupees 251.20%1.2f
scala>
```



## Pattern Matching

The process of checking a pattern against a value is called pattern matching. A successful match returns a value associated with the case.

```
<reference_name> match {  
  case <option 1> => <return_value 1>  
  case <option 2> => <return_value 2>  
  case <option n> => <return_value n>  
  case <default_option> => <default return_value>  
}  
  
def chapterName(chapterNo:Int) = chapterNo match {  
  case 1 => "Scala Features"  
  case 2 => "Spark core"  
  case 3 => "Spark Streaming"  
  case _ => "Chapter not defined"  
}
```

```
scala> def chapterName(chapterNo:Int) = chapterNo match {  
  | case 1 => "Scala Features"  
  | case 2 => "Spark core"  
  | case 3 => "Spark Streaming"  
  | case _ => "Chapter not defined"  
  | }  
chapterName: (chapterNo: Int)String
```

```
scala> chapterName(6) res0:  
String = Chapter not defined
```

```
scala> chapterName(3) res1:  
String = Spark Streaming
```

```
scala> chapterName(1) res2:  
String = Scala Features
```

```
scala> chapterName(2)  
res3: String = Spark core  
scala>
```

## Scala Class vs. Object

A class is a collection of variables, functions, and objects that is defined as a blueprint for creating objects (i.e., instances).

A Scala class can be instantiated (object can be created).

## Creation of RDD

1. Using Parallelized Collection
2. From External Data Source

### 1. Using Parallelized Collection

Parallelized collections can be created by calling SparkContext's parallelize method on an existing collection.

When the parallelize method is applied on a collection, elements of the collection are copied to form a distributed data set.

```
scala> val rdd = Array(1, 2, 3, 4, 5)
rdd: Array[Int] = Array(1, 2, 3, 4, 5)
```

```
scala> val rdd1 = sc.parallelize(rdd)
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at
<console>:24
```

```
scala> rdd1.collect() res4: Array[Int]
= Array(1, 2, 3, 4, 5) scala>
```

### 2. From External Data Source

Input File: input.txt  
Apache Spark  
Big Data and Analytics using Spark  
Learning Spark  
Real time Spark Streaming  
Machine Learning using Spark  
Spark using Scala  
Pyspark  
Spark and Kafka  
Spark and R  
Spark SQL

Note: File is to be copied to the folder where scala is executed

```
scala> val rdd = sc.textFile("./input.txt")
rdd: org.apache.spark.rdd.RDD[String] = ./input.txt MapPartitionsRDD[5] at textFile at
<console>:23
```

### Creating Sample Data

```
scala> rdd.collect res2: Array[String] = Array(Apache Spark, Big Data and Analytics using
Spark, Learning Spark, Real time Spark Streaming, Machine Learning using Spark, Spark
using Scala, Pyspark, Spark and Kafka, Spark and R, Spark SQL)
```

## Creating Sample Data

First, let's generate some sample data:

```
scala> import scala.util.Random._
```

```
import scala.util.Random._ # //
```

Our case class for the Dataset

```
scala> case class Usage(uid:Int, uname:String, usage: Int)
defined class Usage
```

```
scala> val r = new scala.util.Random(42) r:
scala.util.Random = scala.util.Random@7986302f
```

```
scala>
```

```
// Create 1000 instances of scala Usage class
```

```
scala> val data = for (i <- 0 to 1000)
```

```
  | yield (Usage(i, "user-" + r.alphanumeric.take(5).mkString(""),
  | r.nextInt(1000)))
```

```
data: scala.collection.immutable.IndexedSeq[Usage] = Vector(Usage(0,user-Gpi2C,525),
Usage(1,user-DgXDi,502),      Usage(2,user-M66yO,170),      Usage(3,user-xTOn6,913),
Usage(4,user-3xGSz,246),      Usage(5,user-2aWRN,727),      Usage(6,user-EzZY1,65),
Usage(7,user-ZlZMZ,935),      Usage(8,user-VjxeG,756),      Usage(9,user-iqf1P,3),
Usage(10,user91S1q,794),      Usage(11,user-qHNj0,501),      Usage(12,user-7hb94,460),
Usage(13,userbz0WF,142),      Usage(14,user-71nwy,479),      Usage(15,user-7GZz1,823),
Usage(16,user1CSk6,140),      Usage(17,user-WPzIL,246),      Usage(18,user-VaEit,451),
Usage(19,userPSaRq,679),      Usage(20,user-0Kkzu,332),      Usage(21,user-UN3MG,172),
Usage(22,userKwwER,442),      Usage(23,user-ZnltJ,923),      Usage(24,user-IRA17,741),
Usage(25,useryNHRT,299), Usage(26,user-CJY3C,996), Usage(27,user-Yq9WW,52...
```

```
scala>
```

```
// Create a Dataset of Usage typed data
```

```
scala> val dsUsage = spark.createDataset(data) dsUsage:
org.apache.spark.sql.Dataset[Usage] = [uid: int, uname: string ... 1 more field]
```

```
scala> dsUsage.show()
```

```
+---+-----+-----+
|uid|  uname|usage| +--
-+-----+-----+
| 0|user-Gpi2C| 525|
| 1|user-DgXDi| 502|
| 2|user-M66yO| 170|
| 3|user-xTOn6| 913|
| 4|user-3xGSz| 246|
| 5|user-2aWRN| 727|
| 6|user-EzZY1| 65|
```

```
| 7|user-ZlZMZ| 935|
| 8|user-VjxeG| 756|
| 9|user-iqf1P| 3|
|10|user-91S1q| 794|
|11|user-qHNj0| 501|
|12|user-7hb94| 460|
|13|user-bz0WF| 142|
|14|user-71nwy| 479|
|15|user-7GZz1| 823|
|16|user-1CSk6| 140|
|17|user-WPzlL| 246|
|18|user-VaEit| 451|
|19|user-PSaRq| 679| +--
+-----+-----+    only
showing top 20 rows
```

```
scala> dsUsage.show(10)
+---+-----+-----+
|uid|  uname|usage| +--
+-----+-----+
| 0|user-Gpi2C| 525|
| 1|user-DgXDi| 502|
| 2|user-M66yO| 170|
| 3|user-xTOn6| 913|
| 4|user-3xGSz| 246|
| 5|user-2aWRN| 727|
| 6|user-EzZY1| 65|
| 7|user-ZlZMZ| 935|
| 8|user-VjxeG| 756|
| 9|user-iqf1P| 3|
+---+-----+-----+ only
showing top 10 rows

scala>
```

#

### Higher-order functions and functional programming

For a simple example, let's use `filter()` to return all the users in our `dsUsage` Dataset whose usage exceeds 900 minutes. One way to do this is to use a functional expression as an argument to the `filter()` method:

```
scala> import org.apache.spark.sql.functions._
import org.apache.spark.sql.functions._

scala> dsUsage res2: org.apache.spark.sql.Dataset[Usage] = [uid: int, uname: string
... 1 more field]
```

```
scala> .filter(d => d.usage > 900) res3: org.apache.spark.sql.Dataset[Usage] =
[uid: int, uname: string ... 1 more field] scala> .orderBy(desc("usage"))
res4: org.apache.spark.sql.Dataset[Usage] = [uid: int, uname: string ... 1 more field]

scala> dsUsage res5: org.apache.spark.sql.Dataset[Usage] = [uid: int, uname: string
... 1 more field]

scala> .filter(d => d.usage > 900) res6: org.apache.spark.sql.Dataset[Usage] =
[uid: int, uname: string ... 1 more field]

scala> .orderBy(desc("usage")) res7: org.apache.spark.sql.Dataset[Usage] = [uid:
int, uname: string ... 1 more field]

scala> .show(5, false)
+---+-----+-----+
|uid|uname  |usage| +-
--+-----+-----+
|561|user-5n2xY|999 |
|113|user-nnAXr|999 | |605|user-
NL6c4|999 |
|634|user-L0wci|999 |
|26 |user-CJY3C|996 | +-
--+-----+-----+ only
showing top 5 rows
```

```
>>>
>>> scala> def filterWithUsage(u: Usage) = u.usage > 900
filterWithUsage: (u: Usage)Boolean

scala> dsUsage.filter(filterWithUsage(_)).orderBy(desc("usage")).show(5)
+---+-----+-----+
|uid|  uname|usage| +--
-+-----+-----+
|113|user-nnAXr| 999| |605|user-
NL6c4| 999|
|561|user-5n2xY| 999|
|634|user-L0wci| 999|
|805|user-LX27o| 996|
+---+-----+-----+
only showing top 5 rows
```

```
scala>
```

The steps are simple:

1. Create a Scala case class or JavaBean class, UsageCost , with an additional field or column named cost .

2. Define a function to compute the cost and use it in the map() method

Here's what this looks like in Scala:

```
// In Scala
// Create a new case class with an additional field, cost
scala> case class UsageCost(uid: Int, uname:String, usage: Int, cost: Double)
defined class UsageCost scala>
```

```
// Compute the usage cost with Usage as a parameter
// Return a new object, UsageCost scala> scala> def
computeUserCostUsage(u: Usage): UsageCost = {
  | val v = if (u.usage > 750) u.usage * 0.15 else u.usage * 0.50
  | UsageCost(u.uid, u.uname, u.usage, v)
  | }
computeUserCostUsage: (u: Usage)UsageCost
```

```
scala>
// Use map() on our original Dataset
```

```
scala> dsUsage.map(u => {computeUserCostUsage(u)}).show(5)
```

```
+---+-----+-----+-----+
|uid|  uname|usage| cost|
+---+-----+-----+-----+
| 0|user-Gpi2C| 525| 262.5|
| 1|user-DgXDi| 502| 251.0|
| 2|user-M66yO| 170| 85.0|
| 3|user-xTOn6| 913|136.95|
| 4|user-3xGSz| 246| 123.0|
+---+-----+-----+-----+
only showing top 5 rows
```

```
scala>
```

## # Converting DataFrames to Datasets

For strong type checking of queries and constructs, you can convert DataFrames to Datasets.

```
import org.apache.spark.sql.Session
import org.apache.spark.sql.types.StructType
object BookApp extends App { val spark =
SparkSession.builder.appName("bookApp") .getOrCreate() }
val schema = StructType.fromDDL("id bigint, title string, pageCount integer")
import spark.implicits._
val books = spark.sqlContext.read.format("csv") .option("header", "true")
.option("inferSchema", "false") .schema(schema)
.load("Books.csv")

// DataFrame    .as[Book]
// DataSet  books.show()
// UpperCase the book names
books
.map(book => book.title.toUpperCase())
.show()
}
```

