Big data project on Olympic dataset

NUID:001898194

Problem statement: To draw meaningful insights from Olympic dataset using mongoDB, hdfs, hive, pig and hbase. Also compare the performances of different techniques of data storage and processing systems.

Dataset: https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results

Dataset includes all details of games and players from Athens 1896 to Rio 2016.

Summary: I will be performing following analysis on dataset

- Maximum number of participants per year
- Top 10 Medal Winners Overall using MapReduce Chaining
- Top 10 Medal Winning Country using top k filtering
- Players by country using Reduce side join to enrich the dataset
- Hosting cities for Olympics games after year 2000 using secondary sorting
- Loading data into partition table from hive table
- Average height of players by team
- Finding the youngest player of team
- Female participants per year
- Finding the distributions of gold medals across different sports in year 2000

JAVA console app to dump Olympic dataset into MongoDB

```java
String SplitBy = ",";
try (BufferedReader br = new BufferedReader(new FileReader(file2)))
{
    while ((line = br.readLine()) != null) {
        String[] tokens = line.split(SplitBy);
        Document doc = new Document("UserID", tokens[0])
                .append("Name", tokens[1])
                .append("Sex", tokens[2])
                .append("Age", tokens[3])
                .append("Height", tokens[4])
                .append("Weight", tokens[5])
                .append("Team", tokens[6])
                .append("NOC", tokens[7])
                .append("Games", tokens[8])
                .append("Year", tokens[9])
                .append("Season", tokens[10])
                .append("City", tokens[11])

                .append("Sport", tokens[12])
                .append("Event", tokens[13])
                .append("Medal", tokens[14]);
        collection.insertOne(doc);
    }
    br.close();
```

```
run:
Nov 14, 2019 4:02:41 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectic
Nov 14, 2019 4:02:41 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster description not yet available. Waiting for 30000 ms before timing out
Nov 14, 2019 4:02:41 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:1, serverValue:5}] to localhost:27017
Nov 14, 2019 4:02:41 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type
Nov 14, 2019 4:02:41 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:2, serverValue:6}] to localhost:27017
OLYMPIC DATA DUMPED INTO MONGODB
BUILD SUCCESSFUL (total time: 25 seconds)
```

Database name=projectdb

Collection name =olympic

```
> db.olympic.findOne()
{
        "_id" : ObjectId("5dcdce08f37d2221abf91e4b"),
        "UserID" : "1",
        "Name" : "A Dijiang",
        "Sex" : "M",
        "Age" : "24",
        "Height" : "180",
        "Weight" : "80",
        "Team" : "China",
        "NOC" : "CHN",
        "Games" : "1992 Summer",
        "Year" : "1992",
        "Season" : "Summer",
        "City" : "Barcelona",
        "Sport" : "Basketball",
        "Event" : "Basketball Men's Basketball",
        "Medal" : "NA"
}
```

Maximum number of participants per year

```
Var map1= function()
        {
        emit({Year:this.Year},1);
        }
Var red1= function (key,values)
        {
                var sum =0;
                for(i=0;i<values.length;i++)
                {
                sum+=values[i];
                }
                return sum;
        }
```

```
> db.top15years.find().sort({value:-1}).limit(15)
{ "_id" : { "Year" : "2000" }, "value" : 13682 }
{ "_id" : { "Year" : "1988" }, "value" : 13636 }
{ "_id" : { "Year" : "2016" }, "value" : 13443 }
{ "_id" : { "Year" : "2008" }, "value" : 13402 }
{ "_id" : { "Year" : "2004" }, "value" : 13399 }
{ "_id" : { "Year" : "1992" }, "value" : 13109 }
{ "_id" : { "Year" : "2012" }, "value" : 12524 }
{ "_id" : { "Year" : "1996" }, "value" : 11838 }
{ "_id" : { "Year" : "1972" }, "value" : 11482 }
{ "_id" : { "Year" : "1984" }, "value" : 10868 }
{ "_id" : { "Year" : "1968" }, "value" : 10203 }
{ "_id" : { "Year" : "1976" }, "value" : 9567 }
{ "_id" : { "Year" : "1964" }, "value" : 8711 }
{ "_id" : { "Year" : "1980" }, "value" : 8217 }
{ "_id" : { "Year" : "1960" }, "value" : 8038 }
```

Indexing in mongoDB for faster query performance

Before indexing

```
> db.olympic.find({Name:"A Dijiang"}).explain("executionStats")
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "projectdb.olympic",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "Name" : {
                                "$eq" : "A Dijiang"
                        }
                },
                "queryHash" : "EBFEE4C5",
                "planCacheKey" : "EBFEE4C5",
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "Name" : {
                                        "$eq" : "A Dijiang"
                                }
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 1,
                "executionTimeMillis" : 92,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 206152,
                "executionStages" : {
```

Creating Index on Name field

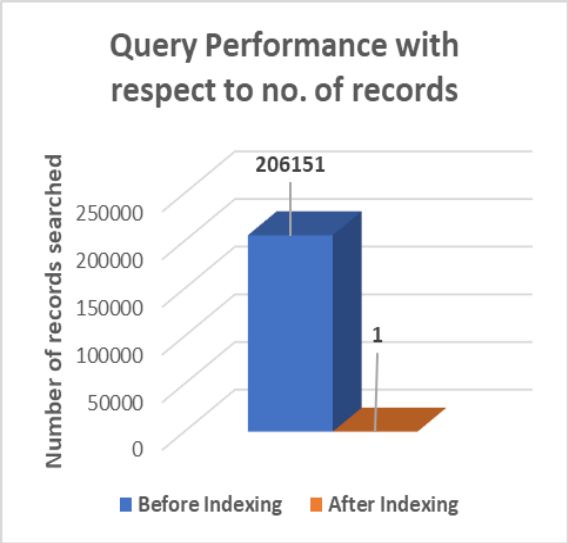
```
> db.olympic.createIndex({Name:1});
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

Here {Name:1} for name field 1 is passed as value so that indexing will be created in ascending order on name field

After Indexin

```
> db.olympic.find({Name:"A Dijiang"}).explain("executionStats")
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "projectdb.olympic",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "Name" : {
                                "$eq" : "A Dijiang"
                        }
                },
                "queryHash" : "EBFEE4C5",
                "planCacheKey" : "6D446D9E",
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "Name" : 1
                                },
                                "indexName" : "Name_1",
                                "isMultiKey" : false,
                                "multiKeyPaths" : {
                                        "Name" : [ ]
                                },
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 2,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "Name" : [
                                                "[\"A Dijiang\", \"A Dijiang\"]"
                                        ]
                                }
                        }
                },
                "rejectedPlans" : [ ]
        },
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 1,
                "executionTimeMillis" : 2,
                "totalKeysExamined" : 1,
                "totalDocsExamined" : 1,
                "executionStages" : {
```

	Before Indexing	After Indexing
Query	db.olympic.find({Name:"A Dijiang"})	db.olympic.find({Name:"A Dijiang"})
Execution time in milliseconds	92 milliseconds	2 milliseconds
Total documents scanned/examined	2,06,152	1



Top 10 Medal Winners Overall using MapReduce Chaining

Combined two jobs

1st job output-Participant name and no. of medals

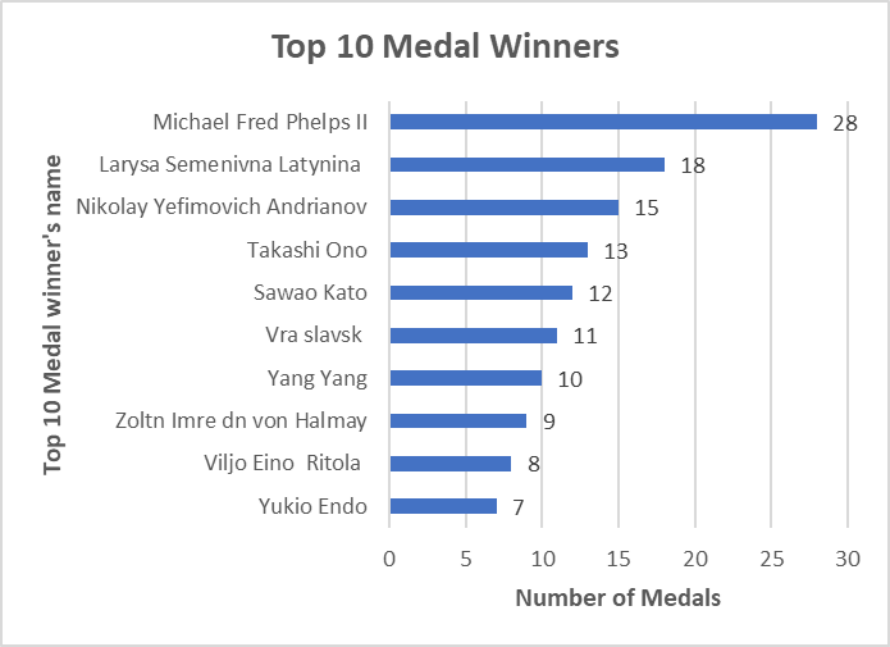
Final output- no of medals and participant name in descending order of number of medals

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/top10.jar ProjectTopMedalWinner.ProjectTopMedalWinner.Driver3 /project /top10MedalWinnersName

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/top10.jar ProjectTopMedalWinner.ProjectTopMedalWinner.Driver3 /project /top10MedalWinnersName
```

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /top10MedalWinnersName/part-r-00000

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /top10MedalWinnersName/part-r-00000
28      Michael Fred Phelps II
18      Larysa Semenivna Latynina
15      Nikolay Yefimovich Andrianov
13      Takashi Ono
12      Sawao Kato
11      Vra slavsk
10      Yang Yang
9       Zoltn Imre dn von Halmay
8       Viljo Eino  Ritola
7       Yukio Endo
```



Partition records based on Season

Season=Winter and Summer

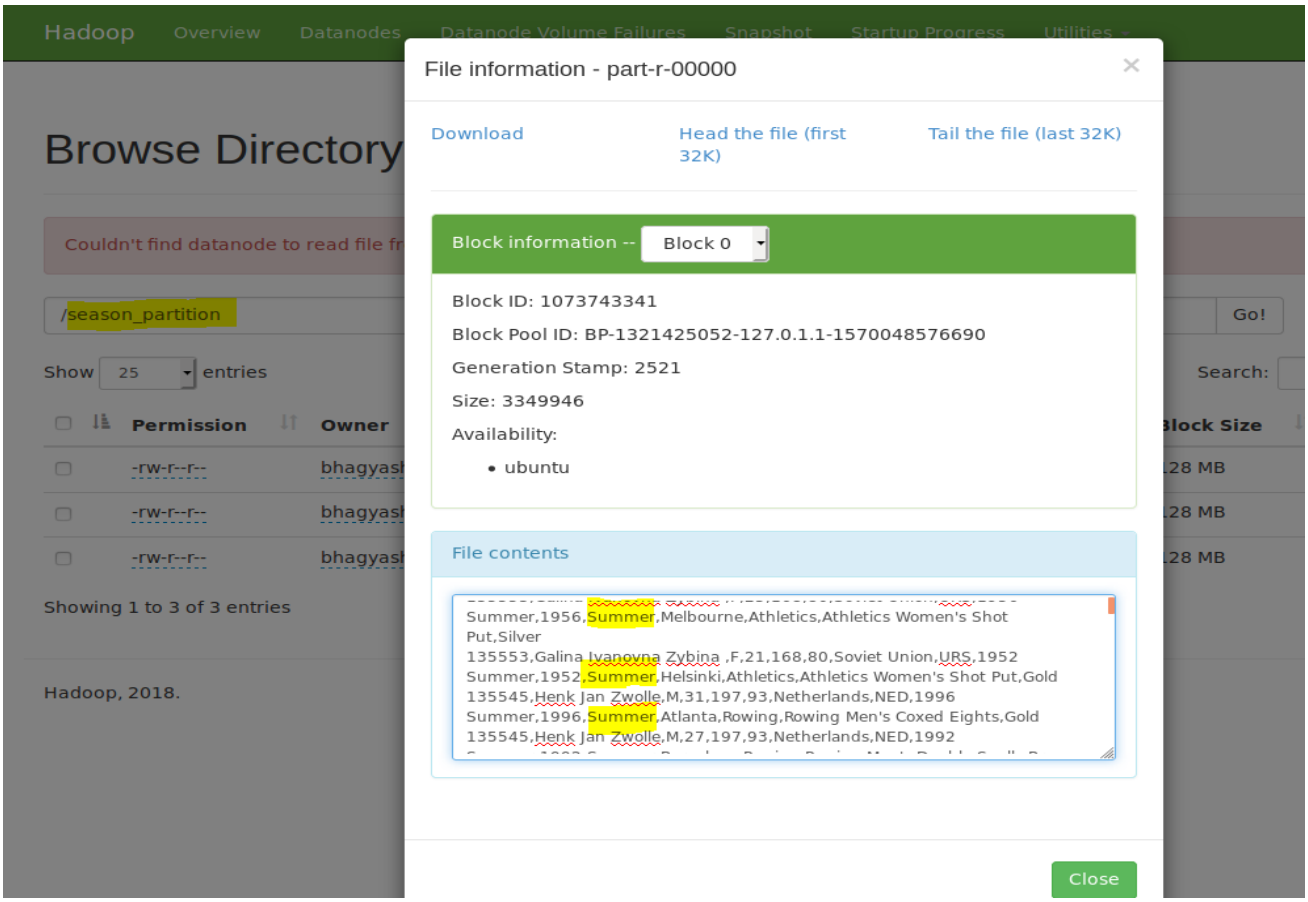
Job.setNumReduceTasks(2)

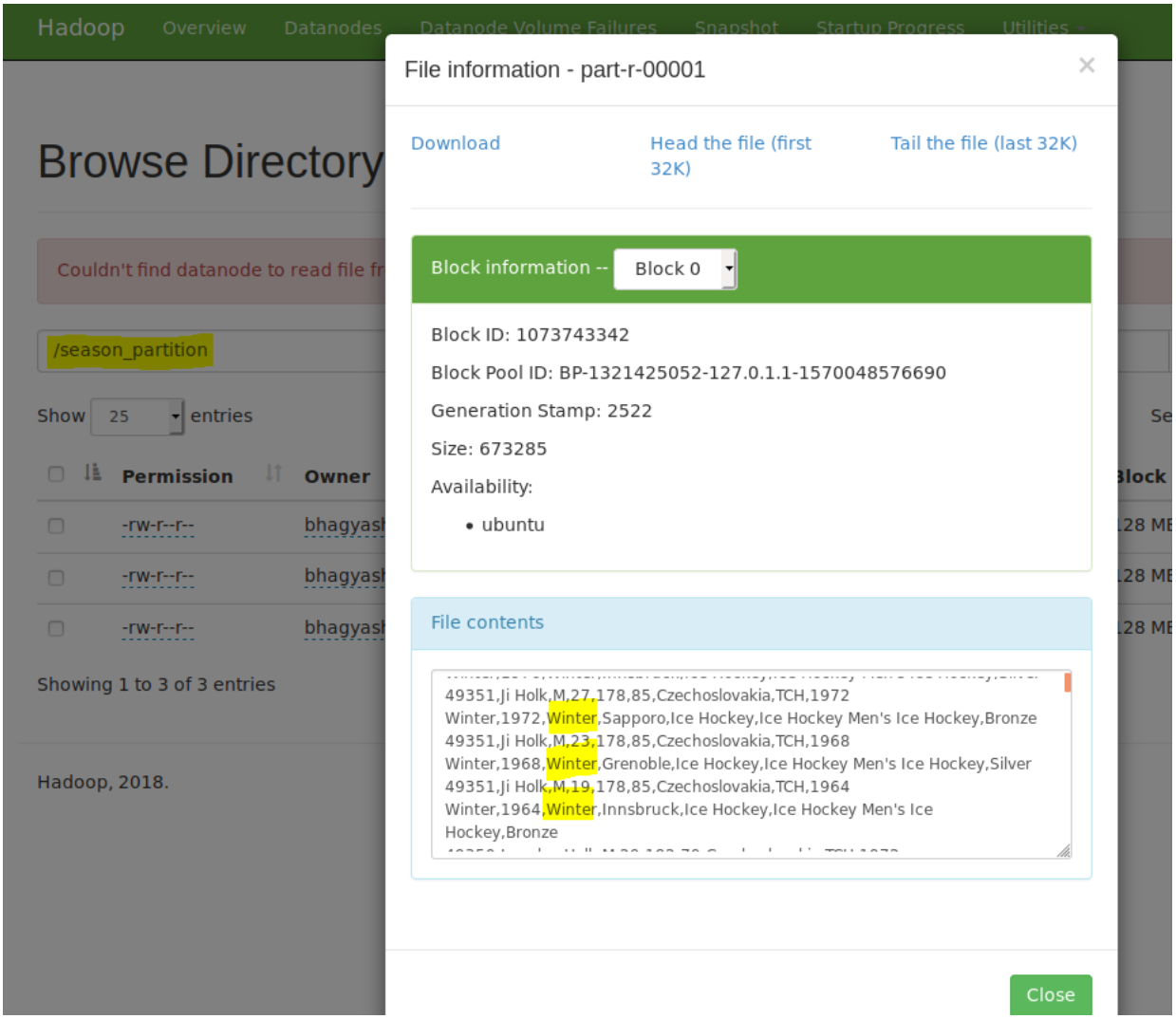
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/SeasonPartition.jar ProjectPartition.ProjectPartition.YearDriver /project/ /season_partition/

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/SeasonPartition.jar ProjectPartition.ProjectPartition.YearDriver /project/ /season_partition/
```

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -ls /season_partition/

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -ls /season_partition/
Found 3 items
-rw-r--r--   1 bhagyashree supergroup          0 2019-11-17 13:51 /season_partition/_SUCCESS
-rw-r--r--   1 bhagyashree supergroup    3349946 2019-11-17 13:51 /season_partition/part-r-00000
-rw-r--r--   1 bhagyashree supergroup     673285 2019-11-17 13:51 /season_partition/part-r-00001
```





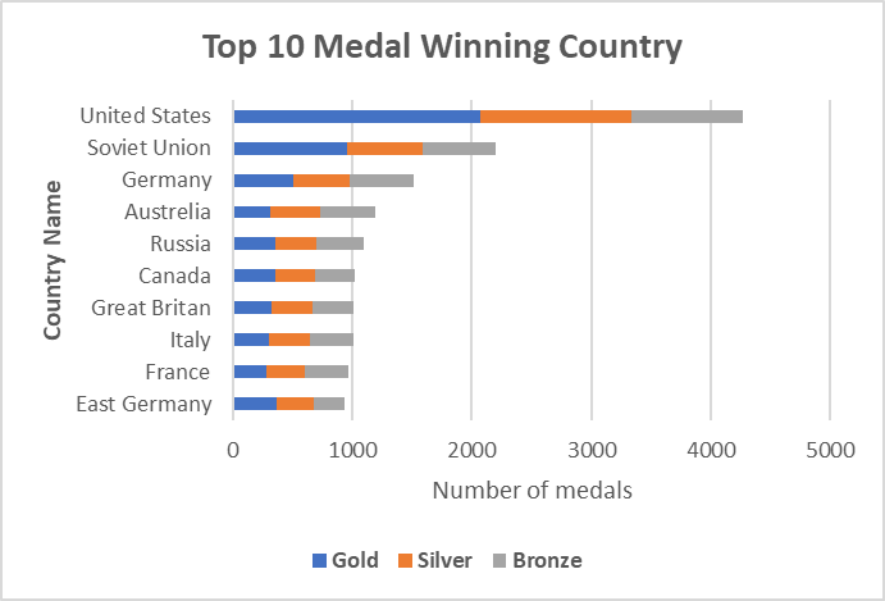
Top 10 Medal Winning Country using top k filtering

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/topTeams.jar ProjectTopTeam.ProjectTopTeam.Driver /teamMedals/ /top10Teams

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/topTeams.jar ProjectTopTeam.ProjectTopTeam.Driver /teamMedals/ /top10Teams
```

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /top10Teams/part-r-00000

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /top10Teams/part-r-00000
4273    United States   GoldCount=2075  SilverCount=1260        BronzeCount=938 Total=4273
2203    Soviet Union    GoldCount=961   SilverCount=629 BronzeCount=613 Total=2203
1518    Germany GoldCount=508   SilverCount=470 BronzeCount=540 Total=1518
1196    Australia       GoldCount=313   SilverCount=412 BronzeCount=471 Total=1196
1091    Russia  GoldCount=356   SilverCount=343 BronzeCount=392 Total=1091
1024    Canada  GoldCount=350   SilverCount=336 BronzeCount=338 Total=1024
1010    Great Britain   GoldCount=321   SilverCount=343 BronzeCount=346 Total=1010
1008    Italy   GoldCount=302   SilverCount=340 BronzeCount=366 Total=1008
965     France  GoldCount=279   SilverCount=320 BronzeCount=366 Total=965
935     East Germany    GoldCount=368   SilverCount=306 BronzeCount=261 Total=935
```



Players by country using Reduce side join to enrich the dataset

Argument 0= first input file [short form of country, players name]

Argument 1=second input file [short form of country, full form of country]

Argument 2=inner

Argument 3=output folder [player name, full form of country]

Used MultipleInputs class

```
  MultipleInputs.addInputPath(job, new
Path(args[0]),TextInputFormat.class,JoinMapper1.class);
  MultipleInputs.addInputPath(job, new
Path(args[1]),TextInputFormat.class,JoinMapper2.class);
  job.getConfiguration().set("join.type",args[2]);
  FileOutputFormat.setOutputPath(job, new Path(args[3]));
```

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar home/bhagyashree/Desktop/join3.jar ProjectJoin.ProjectJoin.JoinDriver /project /projectInputJoin inner /project_inner_join_outpu

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/join3.jar ProjectJoin.ProjectJoin.JoinDriver /project /projectInputJoin inner /project_inner_join_output
```

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /project_inner_join_output/part-r-00000

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /project_inner_join_output/part-r-00000
```

```
Emilija Eri     Serbia
Perica Buki     Serbia
Perica Buki     Serbia
Radmila  Savi   Serbia
Zlatko  Saraevi Serbia
Mirko Sandi     Serbia
Mirko Sandi     Serbia
Uro Marovi      Serbia
Aziz Salihu     Serbia
Ace Rusevski    Serbia
Zoran Mustur    Serbia
Vlado aplji     Serbia
Milan Mukatirovi        Serbia
Vlade Divac     Serbia
Slavica Djuki   Serbia
Samuel Matete   Zambia
Helen Volk      Zimbabwe
AnnMary Gwynne Grant    Zimbabwe
Patricia Jean  McKillop         Zimbabwe
Patricia Joan  Davies   Zimbabwe
Brenda Joan Phillips    Zimbabwe
Gillian  Cowley Zimbabwe
Kirsty Leigh Coventry   Zimbabwe
Alexandra  Chick        Zimbabwe
Kirsty Leigh Coventry   Zimbabwe
Anthea Dorine Stewart   Zimbabwe
Sarah English   Zimbabwe
Kirsty Leigh Coventry   Zimbabwe
Kirsty Leigh Coventry   Zimbabwe
Elizabeth  Chase        Zimbabwe
Maureen Jean George     Zimbabwe
Kirsty Leigh Coventry   Zimbabwe
Sonia Robertson         Zimbabwe
Kirsty Leigh Coventry   Zimbabwe
Christine Seraphine  Prinsloo   Zimbabwe
Kirsty Leigh Coventry   Zimbabwe
Linda Margaret Watson   Zimbabwe
Susan  Huggett  Zimbabwe
```

Hosting cities for Olympics games after year 2000 using secondary sorting

Primary key =city

Secondary key=year (descending order)

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/secSorting.jar ProjectSecSorting.ProjectSecSorting.App /project /secondarySortingOutput

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hadoop jar /home/bhagyashree/Desktop/secSorting.jar ProjectSecSorting.ProjectSecSorting.App /project /secondarySortingOutput
```

bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /secondarySortingOutput/part-r-00000

```
bhagyashree@ubuntu:/usr/local/bin/hadoop-2.9.2/bin$ hdfs dfs -cat /secondarySortingOutput/part-r-00000
CompositeKeyWritable [city=Rio de Janeiro, year=2016]   Summer
CompositeKeyWritable [city=Sochi, year=2014]    Winter
CompositeKeyWritable [city=London, year=2012]   Summer
CompositeKeyWritable [city=Vancouver, year=2010]        Winter
CompositeKeyWritable [city=Beijing, year=2008]  Summer
CompositeKeyWritable [city=Torino, year=2006]   Winter
CompositeKeyWritable [city=Athina, year=2004]   Summer
CompositeKeyWritable [city=Salt Lake City, year=2002]   Winter
CompositeKeyWritable [city=Sydney, year=2000]   Summer
```

----------Hive----------

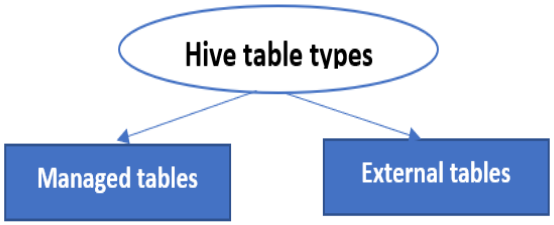
Creating database in hive

hive>create databse OlympicDatabase;

hive>show databases;

```
hive> show databases;
OK
default
olympicdatabase
sample
Time taken: 0.067 seconds, Fetched: 3 row(s)
```

hive>use OlympicDatabase;



	Managed tables	External tables
1	Data is stored in hive	Data is not actually stored in hive. It just refers data stored in hdfs
2	When table is dropped all data and metadata will be lost	Only metadata will be lost from metastore but data will still remain in hdfs
Query For creating table	**hive**> create table Olympics(ID Int,Name String,Sex String,Age Int,Height Int,Weight Int,Team String,NOC String,Games String,Year Int,Season String ,City String,Sport String,Event String,Medal String) row format delimited fields terminated by ',' lines terminated by '\n' tblproperties ("skip.header.line.count"="1");	**hive**>create external table OlympicsExternal(ID String,Name String,Sex String,Age String,Height String,Weight String,Team String,NOC String,Games String,Year String,Season String ,City String,Sport String,Event String,Medal String) row format delimited fields terminated by ',' location '/project';
Loading Data Into table	**hive**> LOAD DATA INPATH "hdfs://localhost:9000/project" OVERWRITE INTO TABLE Olympics;	Data is just referred
Query to find Location Description	**hive**> describe formatted Olympics;	**hive**> describe formatted olympicsexternal;
Location	hdfs://localhost:9000 /user/hive/warehouse /olympicdatabase.db/olympics	hdfs://localhost:9000/project
	 hive> describe olympics; OK id int name string sex string age int height int weight int team string noc string games string year int season string city string sport string event string medal string	 hive> describe olympicsexternal; OK id string name string sex string age string height string weight string team string noc string games string year string season string city string sport string event string medal string

Export hive output to csv file

bhagyashree@ubuntu:/usr/local/bin/apache-hive-2.3.6-bin/bin$ hive -e "select * from olympicdatabase.Olympics limit 5" > ~/Desktop/sample.csv

ields

Column type:

	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard
1	1	A Dijiang	M	24	180	80	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NA
2	2	A Lamusi	M	23	170	60	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NA
3	5	Christine Jacoba Aaftink	F	21	185	82	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NA
4	5	Christine Jacoba Aaftink	F	21	185	82	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 1000 metres	NA
5	5	Christine Jacoba Aaftink	F	25	185	82	Netherlands	NED	1992 Winter	1992	Winter	Albertville	Speed Skating	Speed Skating Women's 500 metres	NA

Partition in hive

If we perform select * from Olympics where year=2016; then whole table will be scanned to fetch records from table. But if we directly store records year wise then query response will be faster as records are stored year wise.

Divide the records based on column=hive partitioning

Distinct value in columns=no of partitions will be created

hive> set hive.exec.dynamic.partition=true;

hive> set hive.exec.dynamic.partition.mode=nonstrict;

hive> set hive.exec.max.dynamic.partitions.pernode=50;

hive>create table **partition_Olympics**(ID Int,Name String,Sex String,Age Int,Height Int,Weight Int,Team String,NOC String,Games String,Season String,City String,Sport String,Event String,Medal String) **partitioned by** (Year Int) stored as textfile;

```
hive> describe partition_olympics;
OK
id                      int
name                    string
sex                     string
age                     int
height                  int
weight                  int
team                    string
noc                     string
games                   string
season                  string
city                    string
sport                   string
event                   string
medal                   string
year                    int
```

Loading data into partition table from hive table

hive> insert OVERWRITE TABLE partition_olympics partition(Year) select id,name,sex,age,height,weight,team,noc,games,season,city,sport,event,medal,year from olympics;

```
ive> insert OVERWRITE TABLE partition_olympics partition(Year) select id,name,sex,age,height,weight,team,noc,games,season,city,sport,event,medal,year from olympics;
ARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using H
uery ID = bhagyashree_20191126201036_4c34479f-dace-4c2f-a511-3d32162bbd50
otal jobs = 3
aunching Job 1 out of 3
umber of reduce tasks is set to 0 since there's no reduce operator
tarting Job = job_1574713965497_0007, Tracking URL = http://ubuntu:8088/proxy/application_1574713965497_0007/
ill Command = /usr/local/bin/hadoop-2.9.2/bin/hadoop job  -kill job_1574713965497_0007
adoop job information for Stage-1: number of mappers: 1; number of reducers: 0
019-11-26 20:10:52,739 Stage-1 map = 0%,  reduce = 0%
019-11-26 20:11:12,018 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 6.51 sec
apReduce Total cumulative CPU time: 6 seconds 510 msec
nded Job = job_1574713965497_0007
tage-4 is selected by condition resolver.
tage-3 is filtered out by condition resolver.
tage-5 is filtered out by condition resolver.
oving data to directory hdfs://localhost:9000/user/hive/warehouse/olympicdatabase.db/partition_olympics/.hive-staging_hive_2019-11-26_20-10-36_241_3481767389622411
oading data to table olympicdatabase.partition_olympics partition (year=null)

oaded : 35/35 partitions.
        Time taken to load dynamic partitions: 6.648 seconds
        Time taken for adding to write entity : 0.039 seconds
apReduce Jobs Launched:
tage-Stage-1: Map: 1   Cumulative CPU: 6.51 sec   HDFS Read: 26972577 HDFS Write: 25729906 SUCCESS
otal MapReduce CPU Time Spent: 6 seconds 510 msec
K
ime taken: 46.876 seconds
```

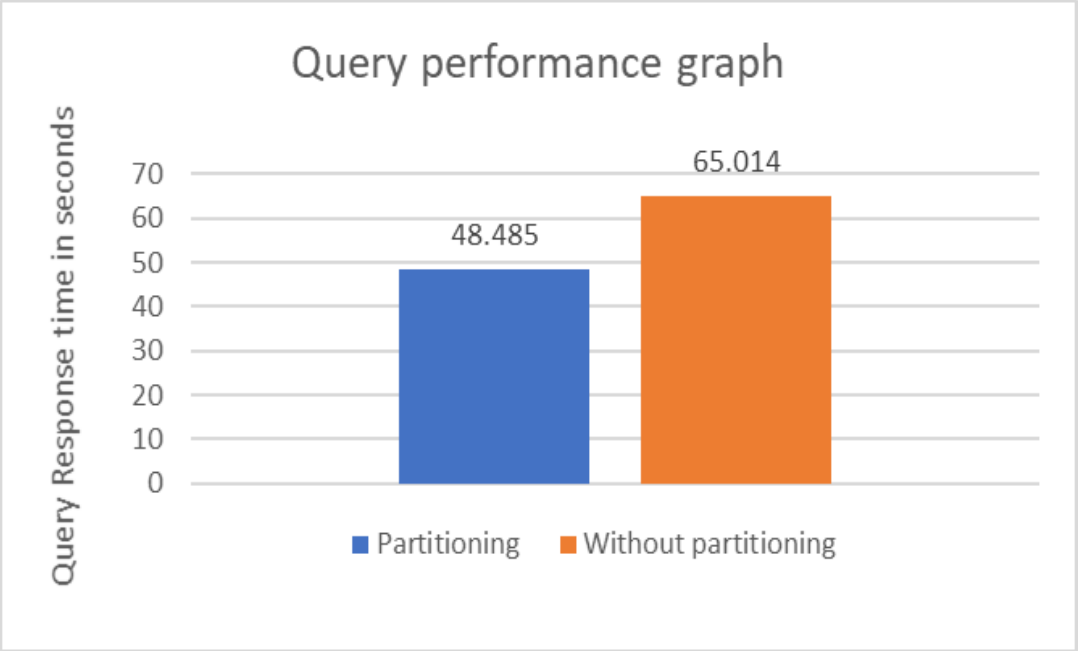
/user/hive/warehouse/olympicdatabase.db/partition_olympics Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1896
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1900
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1904
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1906
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1908
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1912
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1920
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1924
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1928
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1932
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1936
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1948
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1952
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1956
drwxr-xr-x	bhagyashree	supergroup	0 B	Nov 26 20:11	0	0 B	year=1960

	Partitioning	Bucketing
	Partitioning is done based on unique value of the column	Bucketing is done based on **number of records.** **We can use bucketing inside partitioning or separately.**

Partitioning	Without partitioning
select Team,avg(height) from **partition_olympics** where year=2016 group by team;	select Team,avg(height) from **olympics** where year=2016 group by team**;**
Afghanistan 173.66666666666666 Albania 176.16666666666666 Algeria 174.02702702702703 American Samoa 176.75 Andorra 171.5 Angola 174.5 Antigua and Barbuda 176.75 Argentina 178.5438596491228 Argentina-1 186.5 Argentina-2 184.0 Armenia 170.3235294117647 Aruba 174.28571428571428 Australia 178.61904761904762 Australia-1 177.0 Australia-2 175.25 Austria 176.30864197530863 Austria-1 184.5 Austria-2 193.0 Azerbaijan 175.66666666666666 Bahamas 176.78125 Bahrain 171.32142857142858 Bangladesh 166.5 Barbados 180.69230769230768 Belarus 176.55633802816902 Belgium 176.2246376811594 Belize 183.66666666666666 Benin 179.33333333333334 Bermuda 174.125 Bhutan 164.0 Bolivia 171.33333333333334 Bosnia and Herzegovina 180.18181818 Botswana 179.4	Afghanistan 173.66666666666666 Albania 176.16666666666666 Algeria 174.02702702702703 American Samoa 176.75 Andorra 171.5 Angola 174.5 Antigua and Barbuda 176.75 Argentina 178.5438596491228 Argentina-1 186.5 Argentina-2 184.0 Armenia 170.3235294117647 Aruba 174.28571428571428 Australia 178.61904761904762 Australia-1 177.0 Australia-2 175.25 Austria 176.30864197530863 Austria-1 184.5 Austria-2 193.0 Azerbaijan 175.66666666666666 Bahamas 176.78125 Bahrain 171.32142857142858 Bangladesh 166.5 Barbados 180.69230769230768 Belarus 176.55633802816902 Belgium 176.2246376811594 Belize 183.66666666666666 Benin 179.33333333333334 Bermuda 174.125 Bhutan 164.0 Bolivia 171.33333333333334



Views are used in hive to hide the complexity of a query. View is a logical construct it does not store data.

Finding the youngest player details

grunt>create view youngestPlayer as select distinct name,age,year,team,sport from olympics where age in (select min(age) from olympics);

grunt>select * from youngestPlayer;

```
Beatrice Hutiu  11      1968    Romania Figure Skating
Liana Vicens    11      1968    Puerto Rico     Swimming
Sonja Henie     11      1924    Norway  Figure Skating
Time taken: 105.671 seconds, Fetched: 3 row(s)
```

Indexing in hive

Compact indexing stores the pair of indexed **column's value** and its **blockid** on hdfs. Hence when query is triggered first index is checked and then query jumped to corresponding block. Initially created index is empty irrespective of data present in table hence we need to rebuild index using command **ALTER INDEX index_name on olympics REBUILD**. Hence rebuild keyword used in index creation.

Hive> CREATE INDEX index_name

ON TABLE Olympics (name)

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

WITH DEFERRED REBUILD;

```
hive> CREATE INDEX index_name
    > ON TABLE Olympics (name)
    > AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
    > WITH DEFERRED REBUILD;
OK
Time taken: 0.327 seconds
```

hive> ALTER INDEX index_name on olympics REBUILD;

```
hive> show formatted index on olympics;
OK
idx_name            tab_name            col_names           idx_tab_name            idx_type


index_sport         olympics            sport               olympicdatabase__olympics_index_sport__ compact
index_name          olympics            name                olympicdatabase__olympics_index_name__  compact
```

----------**hbase**----------

hbase(main):002:0> create 'olympicdata','cfparticipants';

```
hbase(main):004:0> list
TABLE
mytab
olympicdata
2 row(s)
Took 0.0436 seconds
```

Insert data into table

```
hbase(main):001:0> put 'olympicdata','3','cfparticipants:name','Saina Nehawal'
Took 0.8559 seconds
hbase(main):002:0> put 'olympicdata','3','cfparticipants:game','Badminton'
Took 0.0097 seconds
hbase(main):003:0> put 'olympicdata','3','cfparticipants:medal','Gold'
Took 0.0157 seconds
```

Fetching records for particular row

```
hbase(main):004:0> get 'olympicdata','3'
COLUMN                                                 CELL
 cfparticipants:game                                   timestamp=1574885389523, value=Badminton
 cfparticipants:medal                                  timestamp=1574885405456, value=Gold
 cfparticipants:name                                   timestamp=1574885352982, value=Saina Nehawal
```

Scan the entire table

```
hbase(main):006:0> scan 'olympicdata'
ROW                                                    COLUMN+CELL
 1                                                     column=cfparticipants:medal, timestamp=1574883824062, value=Bronze
 1                                                     column=cfparticipants:name, timestamp=1574881823982, value=Vikas Yadhav
 1                                                     column=cfparticipants:sport, timestamp=1574881834430, value=Boxing
 2                                                     column=cfparticipants:medal, timestamp=1574884333549, value=Silver
 3                                                     column=cfparticipants:game, timestamp=1574885389523, value=Badminton
 3                                                     column=cfparticipants:medal, timestamp=1574885405456, value=Gold
 3                                                     column=cfparticipants:name, timestamp=1574885352982, value=Saina Nehawal
3 row(s)
```

By default, hbase gives us latest version of data. If we want 2 or more versions of data then we need to set VERSION property of hbase.

By default, property

```
hbase(main):007:0> describe 'olympicdata'
Table olympicdata is ENABLED
olympicdata
COLUMN FAMILIES DESCRIPTION
{NAME => 'cfparticipants', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false',
 TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER
'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
```

Alter table to set number of versions we needed

```
hbase(main):010:0> alter 'olympicdata', {NAME => 'cfparticipants', VERSIONS => 3}
Updating all regions with the new schema...
1/1 regions updated.
Done.
```

Now check whether number of versions are updated or not

```
hbase(main):011:0> describe 'olympicdata'
Table olympicdata is ENABLED
olympicdata
COLUMN FAMILIES DESCRIPTION
{NAME => 'cfparticipants', VERSIONS => '3', EVICT_BLOCKS_ON_CLOSE => 'false',
 TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOOMFILTER
'false', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
```

When we update any record both old and new versions will be saved for same rowkey with different timestamp automatically in hbase. If we want to get last two versions of data, then execute following command.

First update record

```
hbase(main):012:0> put 'olympicdata','3','cfparticipants:medal','Silver'
Took 0.0342 seconds
```

Check last two version

```
hbase(main):014:0> get 'olympicdata','3',{COLUMN => 'cfparticipants', VERSIONS => 2}
COLUMN                                                 CELL
 cfparticipants:game                                   timestamp=1574885389523, value=Badminton
 cfparticipants:medal                                  timestamp=1574887266721, value=Silver
 cfparticipants:medal                                  timestamp=1574885405456, value=Gold
 cfparticipants:name                                   timestamp=1574885352982, value=Saina Nehawal
 cfparticipants:name                                   timestamp=1574885275934, value=Saina Nehawal
```

Get the record by specifying timestamp

```
hbase(main):015:0> get 'olympicdata','3',{COLUMN => 'cfparticipants', TIMESTAMP => 1574885405456}
COLUMN                                                 CELL
 cfparticipants:medal                                  timestamp=1574885405456, value=Gold
1 row(s)
```

Table which will live for only 100 sec, set time to live property to 100

```
hbase(main):030:0> alter 'sampletable', {NAME =>'cfregion', VERSIONS =>1, TTL => 100}
Updating all regions with the new schema...
1/1 regions updated.
```

After 100 seconds, table will be no longer live in hbase

```
hbase(main):038:0> scan 'sampletable'
ROW                                                    COLUMN+CELL
0 row(s)
```

Fetching records for particular column[single column]

```
hbase(main):040:0> get 'olympicdata','3',{COLUMN => 'cfparticipants:medal'}
COLUMN                                                 CELL
 cfparticipants:medal                                  timestamp=1574887266721, value=Silver
```

Fetching records for two columns

```
hbase(main):045:0> get 'olympicdata','3',{COLUMN =>[ 'cfparticipants:name','cfparticipants:medal']}
COLUMN                                                 CELL
 cfparticipants:medal                                  timestamp=1574887266721, value=Silver
 cfparticipants:name                                   timestamp=1574885352982, value=Saina Nehawal
```

Status==status ‘summary’

```
hbase(main):046:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 5.0000 average load
Took 0.0586 seconds
hbase(main):047:0> status 'summary'
1 active master, 0 backup masters, 1 servers, 0 dead, 5.0000 average load
Took 0.0375 seconds
```

Status ‘detailed’ will give detailed hbase cluster

```
hbase(main):049:0> status 'detailed'
version 2.2.2
0 regionsInTransition
active master:  ubuntu:16000 1574880395255
0 backup masters
master coprocessors: []
1 live servers
    ubuntu:16020 1574880396679
        requestsPerSecond=0.0, numberOfOnlineRegions=5, usedHeapMB=29, maxHeapMB=845, numberOfStores=7, numberOfStorefiles=8, storefil
ndexSizeKB=0, readRequestsCount=264, filteredReadRequestsCount=2, writeRequestsCount=36, rootIndexSizeKB=0, totalStaticIndexSizeKB=0,
1, compactionProgressPct=1.0, coprocessors=[MultiRowMutationEndpoint]
        "hbase:meta,,1"
            numberOfStores=3, numberOfStorefiles=3, storefileUncompressedSizeMB=0, lastMajorCompactionTimestamp=1574887339048, storefi
t=34, rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=51, currentCompactedKVs=51, compaction
        "hbase:namespace,,1574721353743.6abbb3c25738dfa92a90090c9205c842."
            numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0, lastMajorCompactionTimestamp=0, storefileSizeMB=0,
SizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=NaN,
        "mytab,,1574721395536.3844599cdc4dbc9b42c2baba2515dd51."
            numberOfStores=1, numberOfStorefiles=0, storefileUncompressedSizeMB=0, lastMajorCompactionTimestamp=0, storefileSizeMB=0,
SizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=NaN,
        "olympicdata,,1574880826842.03151a6d9447db6b86dc9c6c65c25f28."
            numberOfStores=1, numberOfStorefiles=2, storefileUncompressedSizeMB=0, lastMajorCompactionTimestamp=0, storefileSizeMB=0,
SizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=NaN,
        "sampletable,,1574888493951.7ee1cf44bd831f9c43b45c5e990cecba."
            numberOfStores=1, numberOfStorefiles=2, storefileUncompressedSizeMB=0, lastMajorCompactionTimestamp=0, storefileSizeMB=0,
SizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=NaN,
0 dead servers
Took 0.1820 seconds
```

Checking version of hbase

```
hbase(main):050:0> version
2.2.2, re6513a76c91cceda95dad7af246ac81d46fa2589, Sat Oct 19 10:10:12 UTC 2019
Took 0.0024 seconds
```

Checking who is user

```
hbase(main):052:0> whoami
bhagyashree (auth:SIMPLE)
    groups: bhagyashree, adm, cdrom, sudo, dip, plugdev, lpadmin, sambashare
Took 0.1821 seconds
```

In order to delete the table, we need to disable it first. Also, once table is disable we cannot insert record into it.

```
hbase(main):053:0> disable  'olympicdata'
Took 1.4098 seconds
```

Enable the table

```
hbase(main):054:0> enable  'olympicdata'
Took 1.4196 seconds
```

Dropping table

```
hbase(main):055:0> drop  'olympicdata'
```

Deleting specific column from table for specific rowkey

```
hbase(main):055:0> delete 'olympicdata', '1','cfparticipants:name'
```

Deleting all columns for rowkey

```
hbase(main):072:0> delete  'olympicdata','2','cfparticipants'
Took 0.0282 seconds
```

Verifying existence of table

```
hbase(main):074:0> exists 'olympicdata'
Table olympicdata does exist
```

----------PIG----------

Loading data into pig

grunt> games= LOAD '/home/bhagyashree/Downloads/athlete_event.csv' USING PigStorage(',')

AS

(ID:Int,Name:Chararray,Sex:Chararray,Age:Int,Height:Int,Weight:Int,Team:Chararray,

NOC:Chararray,Games:Chararray,Year:Int,Season:Chararray,City:Chararray,Sport:Chararray,

Event:Chararray,Medal:Chararray);

No. of Total Participants per year

grouped= GROUP games BY Year;

cnt= FOREACH grouped GENERATE group AS YEAR, COUNT(games.Name) AS PARTICIPANTS;

Storing output into local directory

STORE cnt INTO '/home/bhagyashree/Documents/pig_output';

No of Female Participants per year

filterdata= FILTER games by Sex=='F';

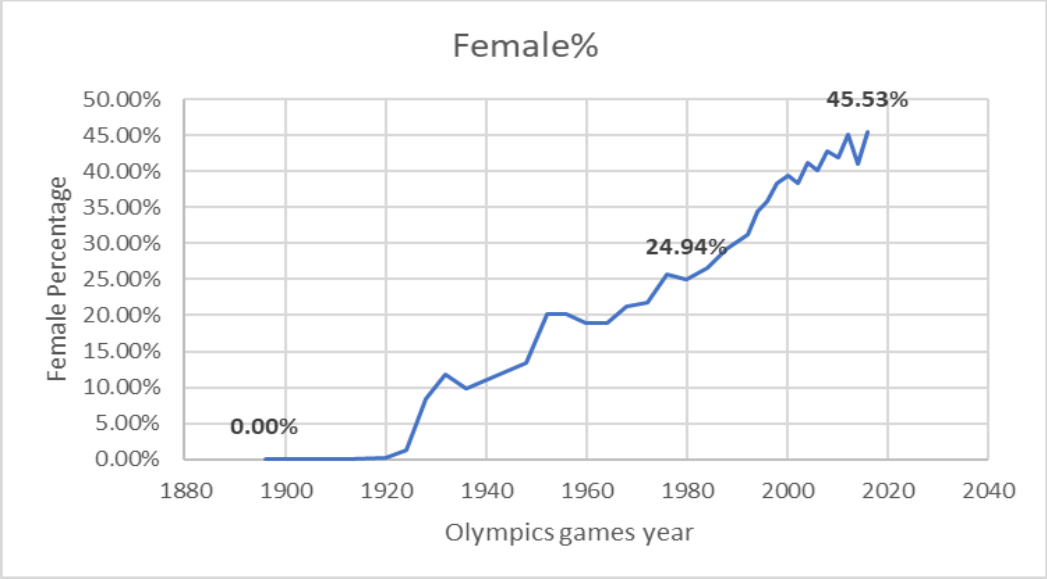
grouped2= GROUP filterdata BY Year;

cnt2= FOREACH grouped2 GENERATE group AS YEAR, COUNT(filterdata.Name) AS PARTICIPANTS;

Storing output into local directory

STORE cnt2 INTO '/home/bhagyashree/Documents/pig_output';

Total no. of **participants** per year	Total no. of **Female** participants per year
grunt>DUMP cnt;	Grunt>Dump cnt2;
(1896,40) (1900,71) (1904,119) (1906,201) (1908,365) (1912,551) (1920,428) (1924,581) (1928,665) (1932,482) (1936,896) (1948,1016) (1952,2058) (1956,2595) (1960,8038) (1964,8711) (1968,10203) (1972,11482) (1976,9567) (1980,8217) (1984,10868) (1988,13636) (1992,13109) (1994,2971) (1996,11838) (1998,3518) (2000,13682) (2002,4060) (2004,13399) (2006,4365) (2008,13402) (2010,4378) (2012,12524) (2014,4673) (2016,13443)	No Female participants 1896-1912 (1920,1) (1924,7) (1928,56) (1932,57) (1936,88) (1948,137) (1952,417) (1956,525) (1960,1516) (1964,1643) (1968,2169) (1972,2494) (1976,2463) (1980,2049) (1984,2885) (1988,4002) (1992,4085) (1994,1023) (1996,4242) (1998,1350) (2000,5386) (2002,1555) (2004,5536) (2006,1753) (2008,5739) (2010,1837) (2012,5655) (2014,1920) (2016,6121)



Percentage of female participants are increasing by the passing year.

Finding the distributions of gold medals across different sports in year 2000 (sport,no. of gold medals)

filterdata= FILTER games by Medal== 'Gold' AND Year== 2000;

groupSport= GROUP filterdata BY Sport;

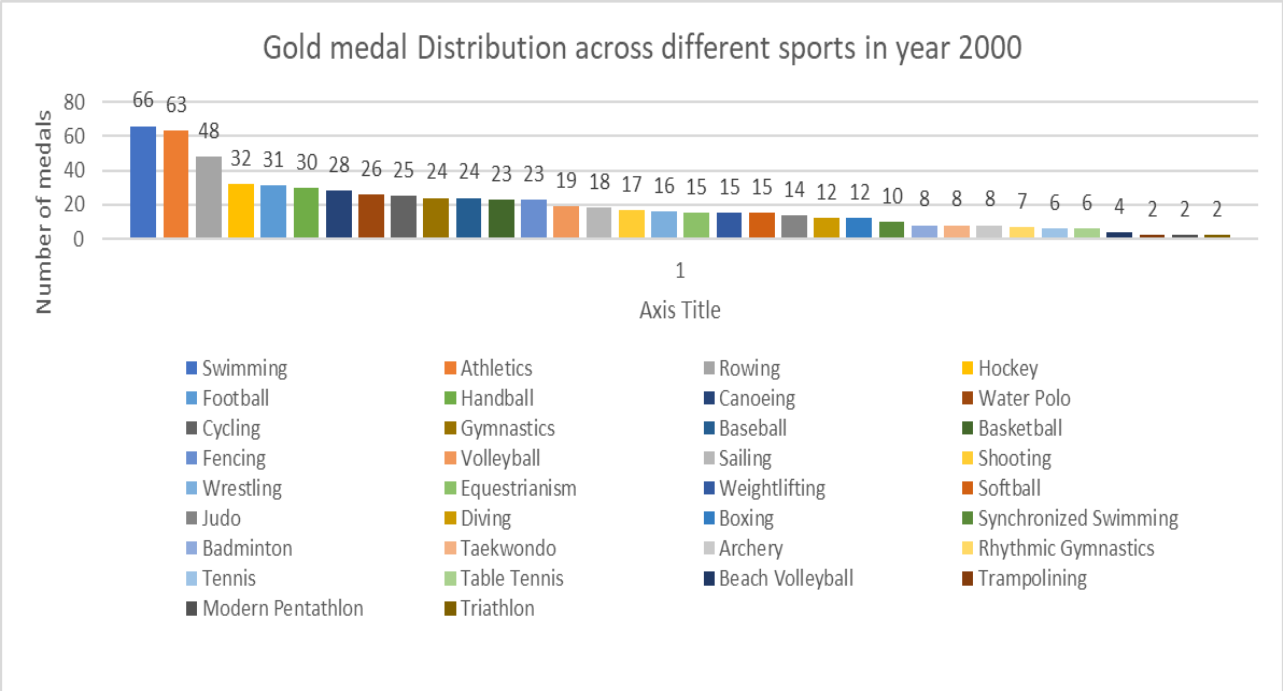
cntSport= FOREACH groupSport GENERATE group as Sport,COUNT(filterdata.Medal) as count;

orderSport= ORDER cntSport BY count desc;

STORE orderSport INTO '/home/bhagyashree/Documents/goldmedaldistribution';

grunt>DUMP orderSport;

```
(Swimming,66)
(Athletics,63)
(Rowing,48)
(Hockey,32)
(Football,31)
(Handball,30)
(Canoeing,28)
(Water Polo,26)
(Cycling,25)
(Gymnastics,24)
(Baseball,24)
(Basketball,23)
(Fencing,23)
(Volleyball,19)
(Sailing,18)
(Shooting,17)
(Wrestling,16)
(Equestrianism,15)
(Weightlifting,15)
(Softball,15)
(Judo,14)
(Diving,12)
(Boxing,12)
(Synchronized Swimming,10)
(Badminton,8)
(Taekwondo,8)
(Archery,8)
(Rhythmic Gymnastics,7)
(Tennis,6)
(Table Tennis,6)
(Beach Volleyball,4)
(Trampolining,2)
(Modern Pentathlon,2)
(Triathlon,2)
```



MongoDB	C:\Program Files\MongoDB\Server\4.2\bin\mongod C:\Program Files\MongoDB\Server\4.2\bin\mongo
HDFS	/usr/local/bin/hadoop-2.9.2/sbin$./start-all.sh /usr/local/bin/hadoop-2.9.2/sbin$./stop-all.sh /usr/local/bin/hadoop-2.9.2/bin$ Hadoop jar jarname.jar driverclass /source /destination
hive	/usr/local/bin/apache-hive-2.3.6-bin/bin$ hive Hive>quit;
hbase	/usr/local/bin/hbase-2.2.2-bin/hbase-2.2.2/bin$ ./start-hbase.sh /usr/local/bin/hbase-2.2.2-bin/hbase-2.2.2/bin$ ./hbase shell hbase(main):001:0> quit
pig	/usr/local/bin/pig-0.17.0/bin$ pig -x local grunt> quit;

Appendix

----------Top 10 Medal Winners Overall using MapReduce Chaining----------

Mapper1 code

```
public class TopMedalWinMapper extends Mapper<LongWritable,Text,Text,IntWritable>
{
IntWritable one = new IntWritable(1);
Text name = new Text();
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
String line = value.toString();
String [] tokens = line.split(",");
//tokens[14]=Medal
//tokens[1]=name
```

```
if(!tokens[14].equals("NA"))
{
name.set(tokens[1]);
context.write(name, one);
}}}
```

Reducer1 code

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

Mapper2 code

```
public class M3 extends Mapper<Text, IntWritable,IntWritable,Text> {
private TreeMap<Integer, String> tmap;
//Called once in the beginning before the method
@Override
public void setup(Context context) throws IOException, InterruptedException
{
tmap = new TreeMap<Integer, String>(Collections.reverseOrder());
}
//Called once for each key/value pair in the input split
@Override
public void map(Text key, IntWritable value, Context context) throws IOException,
InterruptedException
{
Integer mcount=Integer.parseInt(value.toString());
String name=key.toString();
tmap.put(mcount,name);

if (tmap.size() > 10)
{
tmap.remove(tmap.lastKey());
}
}
//Called once at the end of the task
@Override
```

```
public void cleanup(Context context) throws IOException, InterruptedException
{
for (Map.Entry<Integer, String> entry : tmap.entrySet())
{
Integer count = entry.getKey();
String name = entry.getValue();
context.write(new IntWritable(count),new Text(name));
}}}
```

----------Partition records based on Season----------

Mapper code

```
public class YearMapper extends Mapper<LongWritable,Text,Text,Text>  {

IntWritable one = new IntWritable(1);

Text season = new Text();

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

String line = value.toString();

String [] tokens = line.split(",");

if(!tokens[10].equals("Season"))

{

if(!tokens[14].equals("NA"))

{

season.set(tokens[10]);

context.write(season, value);

}}}}
```

Reducer code

```
public class YearReducer extends Reducer<Text,Text,Text,NullWritable>{

@Override

protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {

for(Text data: values)

{
```

```
context.write(data,NullWritable.get());
}}}
```

Practitioner code

```
public class YearPartitioner extends Partitioner <Text,Text>{
@Override
public int getPartition(Text key, Text value, int setNumReduce) {
String season=key.toString();
if(season.equals("Summer"))
{
return 0;
}
else
{
return 1;
}}}
```

---------- **Top 10 Medal Winning Country using top k filtering**----------

Mapper code

```
public class TopTeamMapper extends Mapper<LongWritable, Text, Text, LongWritable>{
private TreeMap<Long, String> tmap;
//Called once in the beginning before the method
@Override
public void setup(Context context) throws IOException,
InterruptedException
{
tmap = new TreeMap<Long, String>();
}
//Called once for each key/value pair in the input split
@Override
```

```
public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException
{
String[] tokens = value.toString().split("\t");
String [] temp2=tokens[4].split("=");
String v=temp2[1];
//String team = tokens[0];
String team = value.toString();
long no_of_medals = Long.parseLong(v);
tmap.put(no_of_medals, team);
if (tmap.size() > 10)
{
tmap.remove(tmap.firstKey());
}}
//Called once at the end of the task
@Override
public void cleanup(Context context) throws IOException,
InterruptedException
{
for (Map.Entry<Long, String> entry : tmap.entrySet())
{
long count = entry.getKey();
String team = entry.getValue();
context.write(new Text(team), new LongWritable(count));
} } }
```

Reducer code

```
public class TopTeamReducer extends Reducer<Text, LongWritable, LongWritable, Text>{
private TreeMap<Long, String> tmap2;
@Override
```

```
public void setup(Context context) throws IOException, InterruptedException
{
tmap2 = new TreeMap<Long, String>(Collections.reverseOrder());
}
@Override
public void reduce(Text key, Iterable<LongWritable> values, Context context) throws
IOException, InterruptedException
{
String team = key.toString();
long count = 0;
for (LongWritable val : values)
{
count = val.get();
}
tmap2.put(count, team);
if (tmap2.size() > 10)
{
tmap2.remove(tmap2.firstKey());
}}
@Override
public void cleanup(Context context) throws IOException,
InterruptedException
{
for (Map.Entry<Long, String> entry : tmap2.entrySet())
{
long count = entry.getKey();
String team = entry.getValue();
context.write(new LongWritable(count), new Text(team));
} } }
```

---------- Hosting cities for Olympics games after year 2000 using secondary sorting----

Mapper code

```
public class SecSortingMapper extends
Mapper<LongWritable,Text,CompositeKeyWritable,Text>

{

Text season1=new Text();

protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException

{

String line=value.toString();

String[] tokens=line.split(",");

String city=tokens[11];

String year=tokens[9];

String season=tokens[10];

season1.set(season);

String y="1999";

if(1==year.compareTo(y))

{

CompositeKeyWritable obj= new CompositeKeyWritable(city,year);

context.write(obj,season1);

}}}
```

Reducer code

```
public class SecSortingReducer extends
Reducer<CompositeKeyWritable,Text,CompositeKeyWritable,Text>{

//IntWritable sum=new IntWritable();

Text season=new Text();
```

```
public void reduce(CompositeKeyWritable key, Iterable<Text> values, Context context) throws
IOException, InterruptedException
{
String temp="";
for(Text t:values)
{
if(temp.isEmpty())
{
temp=t.toString();
}
else if(!temp.contains(t.toString()))
{
temp=temp+','+t.toString();
}}
season.set(temp);
context.write(key,season);
}}
```

Sorting comparator code

```
public class sortComparator extends WritableComparator {
public sortComparator() {
super(CompositeKeyWritable.class,true);
// TODO Auto-generated constructor stub
}
@Override
public int compare(WritableComparable a,WritableComparable b)
{
CompositeKeyWritable k1=(CompositeKeyWritable)a;
CompositeKeyWritable k2=(CompositeKeyWritable)b;
String year1=k1.getYear();
```

```
String year2=k2.getYear();
int result=year1.compareTo(year2);
return -1*result;
}}
```

Grouping comparator code

```
public class GroupKeyComparator extends WritableComparator{
public GroupKeyComparator()
{
super(CompositeKeyWritable.class,true);
// TODO Auto-generated constructor stub
}
@Override
public int compare(WritableComparable a,WritableComparable b)
{
CompositeKeyWritable k1=(CompositeKeyWritable)a;
CompositeKeyWritable k2=(CompositeKeyWritable)b;
String city1=k1.getCity();
String city2=k2.getCity();
int result=city1.compareTo(city2);
return result;
}}
```

Practitioner code

```
public class NaturalKeyPartitioner extends Partitioner <CompositeKeyWritable,Text>
{
public int        getPartition(CompositeKeyWritable key, Text value, int numPartitions)
{
return key.getCity().hashCode()%numPartitions;
}}
```

Composite key writable code

```
public class CompositeKeyWritable implements WritableComparable {
String city;
String year;
public CompositeKeyWritable() {}
public CompositeKeyWritable(String city, String year) {
super();
this.city = city;
this.year = year;
}
public String getCity() {
return city;
}
public void setCity(String city) {
this.city = city;
}
public String getYear() {
return year;
}
public void setYear(String year) {
this.year = year;
}

public void readFields(DataInput in) throws IOException {
// TODO Auto-generated method stub
city=in.readUTF();
year=in.readUTF();
```

```
}

public void write(DataOutput out) throws IOException {
// TODO Auto-generated method stub
out.writeUTF(city);
out.writeUTF(year);
}
public int compareTo(Object o) {
CompositeKeyWritable ck=(CompositeKeyWritable)o;
String thisValue = this.getYear();
String otherValue = ck.getYear();
int result=thisValue.compareTo(otherValue);
return (result< 0 ? -1 : (result == 0 ? 0 : 1));
}
@Override
public String toString() {
return "CompositeKeyWritable [city=" + city + ", year=" + year + "]";
}}
```

----------Players by country using Reduce side join to enrich the dataset----------

Mapper1 code

```
public class JoinMapper1 extends Mapper<LongWritable,Text,Text,Text>{
Text name = new Text();
Text region = new Text();
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
String line = value.toString();
String [] tokens = line.split(",");
```

```
//value----->tokens[1]--->Name of player

//key------>tokens[7]--->Region

if(!tokens[14].equals("NA"))

{

region.set(tokens[7]);

name.set("A"+tokens[1]);

context.write(region, name);

}}}
```

Mapper2 code

```
public class JoinMapper2 extends Mapper<LongWritable,Text,Text,Text>{

Text fullform = new Text();

Text region = new Text();

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

String line = value.toString();

String [] tokens = line.split(",");

//value----->tokens[1]--->full form of region

//key------>tokens[0]--->Region

region.set(tokens[0]);

fullform.set("B"+tokens[1]);

context.write(region, fullform);

}}
```

Reducer code

```
public class JoinReducer extends Reducer<Text,Text,Text,Text>{

public ArrayList<Text> listA = new ArrayList<Text>();

public ArrayList<Text> listB = new ArrayList<Text>();

Text tmp=new Text();

String jointype=null;
```

```
public void setup(Context context) {
//get type of join from configuration
jointype=context.getConfiguration().get("join.type");
}
@Override
protectd void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException
{
listA.clear();
listB.clear();
for(Text t:values)
{
if(t.charAt(0)=='A')
{
listA.add(new Text(t.toString().substring(1)));
}
else if(t.charAt(0)=='B')
{
listB.add(new Text(t.toString().substring(1)));
}
}
//now our listA and listB are ready
if(jointype.equalsIgnoreCase("inner"))
{
if(!listA.isEmpty() && !listB.isEmpty())
{
for (Text A: listA)
{for (Text B: listB)
```

```
{
context.write(A, B);
}  } } } }
```